StyleAI: A Deep-Learning Powered Outfit Generator

Nithya Attaluri
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge MA, 02139
nithyaa@mit.edu

Sadhana Lolla
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge MA, 02139
sadhana@mit.edu

Abstract

This work presents StyleAI, an AI-powered fashion stylist, which takes in a garment as input and outputs other clothing items that would pair well with it. This work proposes a novel model contrastive learning architecture for this application; contrastive learning is a form of self-supervised learning that enables models to easily learn associations between their inputs. Key to the functionality of this model is a novel representation for clothing and sets of compatible clothing, or outfits, in the form of a neural-network generated embedding. This work proposes a method of robustly encoding information about individual garments, and the way they fit together as a whole outfit. Ultimately, when trained on a dataset of images, StyleAI achieves a top-5 accuracy of 57%. Qualitative analysis of StyleAI's output predictions suggests that its performance as an AI-powered stylist is even higher.

1. Introduction

Computer vision models have advanced significantly in recent years. Artificial intelligence can now generate realistic images and videos of people and scenes who do not exist. It can identify objects in an image in nearly real-time, and generate brand-new artwork given a few suggestions as input [4] [3].

Amidst these advances, we pose the question: is it possible for computer vision systems to gain an eye for fashion? In addition to aiding users by providing them with AI-generated fashion advice, this problem poses a much deeper question, which is if we can teach AI models to learn completely subjective behaviors. Instilling a model with a "fashion sense" is a difficult problem. The model must identify texture, color, and pattern of garments, and learn which combinations of these factors pair well. It must also consider garment shape and length. A dress is different from a shirt. Different "cuts" of jeans (i.e. straight cut, skinny, boot cut) change the styling capabilities of the garment. The subjective and qualitative nature of this task makes it harder still. There are no quantitative metrics that describe whether a specific top pairs "better" with one pair of pants over another.

Compared to other computer vision tasks, this problem is unique in that there is no "correct answer." Even if ten fashion designers were placed in front of the same corpus of clothing and given an item around which to design an outfit, they would probably also design different outfits. Therefore, our goal here is not to create a model that has memorized the fashion sense on the dataset on which it is trained, but rather to imbue the model with a sense of what items pair well together, and a sense of color, texture, and compatibility when creating outfits.

It is difficult, unwieldy, and likely incorrect to only consider each facet of a garment individually when designing outfits. The key to developing a fashion sense, then, is identifying a good garment representation that captures all of the individual characteristics of the garment. With this representation, it is then important to determine how to combine items in a way that maximizes compatibility. Because there are no hard rules for styling clothes (it can be difficult to articulate why two items of clothing match or not), a non-human fashion stylist must learn by example, not by heuristic.

To this end, we propose StyleAI, a novel artificial-intelligence model that can generate outfits (an outfit is defined as a top, bottom, and pair of shoes) given a specific garment as a starting point.

StyleAI relies on contrastive learning to make its predictions. Contrastive learning is a self-supervised learning technique which relies on a dataset with a few image and label pairs, and a large number of positive and negative examples that relate to that class. It aims to mimic how a human learns — by learning the associations between items.

Unlike prior work, StyleAI relies on rich embeddings of clothing items generated in *two* different spaces: the item space and the outfit space. These embeddings are generated by combining the image of the garment with associated text (e.g. "Levi's light wash jeans" or "Nike Air Force 1"). Prior work has utilized just one embedding to capture information

about the garment and its compatibility with other clothing items.

Our embeddings separate this information into two different entities, with the intuition that a single embedding trying to optimize for two different objectives (item description and compatibility) is less effective than two separate embeddings. For instance, a pair of blue jeans should be clustered tightly with other blue jeans and pants in the item embedding space, to capture its clothing category, color, fit, etc. On the other hand, in the outfit embedding space blue jeans should be grouped with other categories of garments to be able to create outfits that contain one item in each category (top, bottom, shoe). It is difficult for both types of clusters to exist in the same embedding space.

Our quantitative and qualitative results show that StyleAI is successful in its role as a stylist. Quantitatively, the best version of StyleAI achieves 57% top-5 accuracy on outfit generation. Qualitatively, these results appear better. Even if StyleAI does not predict the exact garment that the input image was matched with, it predicts something very similar.

Ultimately this work makes the following contributions:

- Designs a new contrastive-learning architecture, known as StyleAI, for outfit prediction that utilizes multiple embeddings to make informed decisions about compatibility in outfits
- Identifies a robust method of representing garments that captures both descriptive information and compatibility
- Demonstrates successful outfit generation by StyleAI, with 57% top-5 accuracy and higher qualitative evaluation

2. Background and Related Work

AI-powered fashion stylists have been proposed in the past. Prior work has explored using a variety of different learning techniques. Some work uses genetic search, while others learn intelligent distance metrics to represent compatibility between clothes [10] [11]. Other work explores representing an outfit as a sequence of clothes which could be predicted using LSTMs [5] [7], or using multi-task self-supervised learning to identify similarities in color or texture that could indicate a good match.

Bettaney et al generates embeddings for each clothing item, based on the image of the garment, textual descriptors, and a high-level item style category (e.g. "casual plain") [1]. Item embeddings for a proposed outfit are passed into a scorer, which utilizes a dot-product based metric to evaluate the compatibility of the items when worn together.

2.1. Encoders

There are many different ways to generate embeddings of data. An embedding is a compacted representation of the most salient features of input data. Based on the way that embeddings are generated, distances between embeddings in the overall embedding space can represent changes in different properties of the input data.

Many different models, known as encoders, have been pre-trained on large, general corpuses to embed different modalities of data. For instance, GloVe is an encoder for textual data [8]. More recently, encoders have combined multiple modalities of data. CLIP (Contrastive Language-Image Pre-Training) is trained to learn strong associations between pairs of images and text, and generate embeddings that capture this information [9]. These embeddings are often more robust and contain more information.

2.2. Self-Supervised Contrastive Learning

Similarly, there are many different ways to train a model: supervised, self-supervised, and unsupervised. Contrastive learning is a type of self-supervised learning.

While prior work has exploited self-supervised learning to learn textures and colors of clothing items, it has not utilized contrastive learning on *entire* garments as StyleAI does in the context of fashion and outfit styling.

The basic principle of contrastive learning is shown in Figure 1. Contrastive learning learns positive and negative associations like the ones shown. This is preferable to supervised learning, because we do not have to explictly pass in every association as a (data, label) pair when training the model. It is also preferable to unsupervised learning, because we do have a grouping of items that we would like to enforce.

3. Methods

Inspired by recent advances in contrastive learning, we choose to learn a two-part model representation in a manner similar to that of SimCLR and other contrastive learning frameworks. SimCLR is a state of the art contrastive learning architecture [2].

Namely, our model is comprised of two parts: an encoder $f(x)$ which outputs embeddings, and a projection head $g(x)$ which projects these embeddings into the outfit space. We then apply a contrastive loss function to maximize similarity between items of the same outfit and minimize similarity between items from different outfits. We choose to focus on outfits with three items: tops, bottoms, and shoes. The model architecture can be seen in 2.

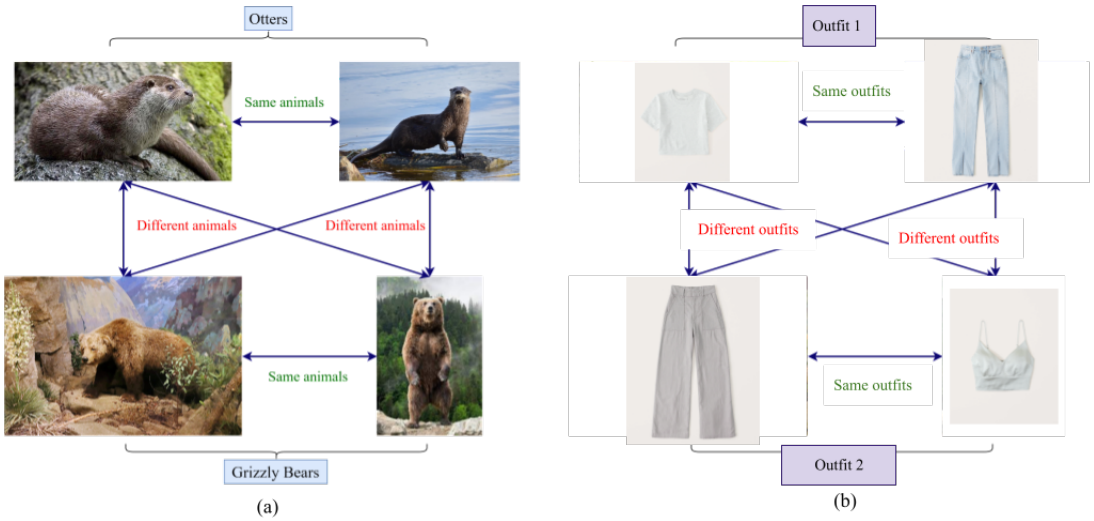

Figure 1. (a) Standard example of contrastive learning; (b) StyleAI's example of contrastive learning, where outfit groupings are taken from the dataset

3.1. Dataset

3.1.1 The Polyvore Dataset

We use the Polyvore dataset, which contains outfits created by users from the now-defunct website Polyvore.com. These outfits represent a wide range of tastes and clothing items. Each item in the dataset contains an image, the name of the item (usually descriptive of the color, fit, or other attributes of the outfit). Previous studies have capitalized on the order of the outfit items (usually top, bottom, shoes, purse, followed by accessories) to model this as a sequential learning problem, where items are chosen sequentially based on items that came before them in the tuple of items– however, we argue that these sequences are arbitrary and choose to focus on representation learning of the items themselves [7]. The Polyvore dataset contains 21000 outfits for training, which we filter based on the presence of tops, bottoms, and shoes. Our training set therefore contains around 11000 outfits, each consisting of three items. Most of the images in the Polyvore dataset are standalone images of the items themselves (i.e. not a model wearing the item in question), so we do not have to segment out the relevant parts of the image. We choose to focus on images of the clothing items only in order to reduce noise and use text embeddings to convey more information about the fit, texture, and other details of the item.

3.2. The Contrastive Learning Framework

Our contrastive learning architecture (shown in figure 1) takes in outfits. For each outfit, the encoder generates the item embedding of each garment of clothing. This embedding is passed through the projection head, which generates

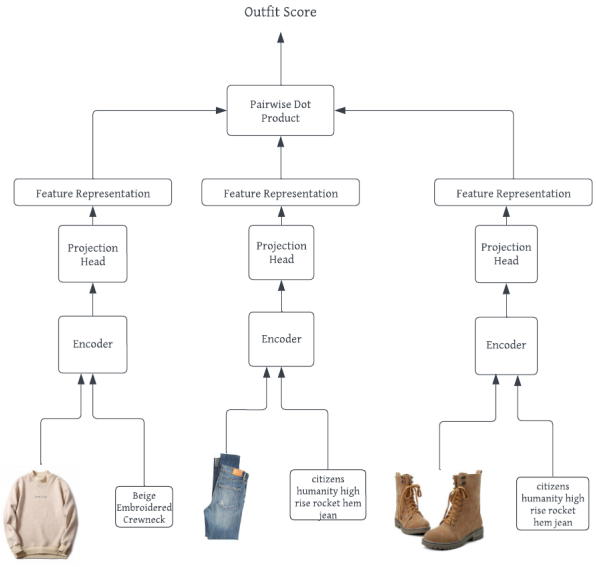

Figure 2. StyleAI Model Architecture

the feature representation in the outfit embedding space. Our goal is that items that belong in the same outfit are close to each other in this latent space, while items that are not part of the same outfit are further apart in the latent space. We then apply the contrastive loss can be applied in the outfit embedding space. The outfit embeddings are evaluated for similarity through a pairwise dot product to generate a compatibility score for the outfit.

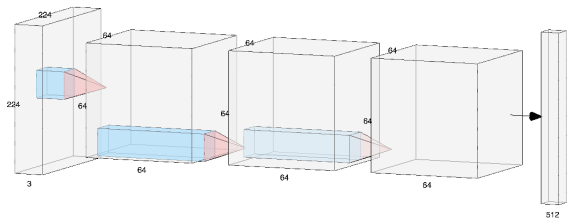

Figure 3. ConvNet Model Architecture

3.2.1 Choice of Encoder

We experiment with two different encoders: CLIP, a self-supervised method developed by OpenAI [9], which learns a representation of an image by learning the association between the image and corresponding text. CLIP applies contrastive learning to maximize the similarity between an image and its corresponding text. We compare representations learned by CLIP with a standard Resnet18, as well as a network consisting mainly of a few convolutional layers.

- **CLIP:** CLIP is a self-supervised learning algorithm, as well as a zero-shot classification algorithm, where representations of images are learned by calculating the cosine similarity between tokenized vectors of the corresponding text and image embeddings. We use CLIP to capitalize on the presence of large amounts of descriptive text associated with every image. When we use CLIP, we **concatenate the embeddings** of the images and text and pass these into the projection head later on.
- **Resnet:** Resnet is an extremely common base model network used in various self-supervised tasks, so we use it here to extract only the image embeddings.
- **Simple CNN:** Finally, we compare the performance on Resnet and CLIP with an extremely simple CNN, short for Convolutional Neural Network, (architecture shown in 3) to determine whether training from scratch achieves better results than using pretrained networks.

3.2.2 Projection Head

We develop a projection head that learns a new embedding from the original image embeddings, and apply a contrastive loss in this space, explained in section 3.2.3. We choose to apply the projection head because it has been shown that introducing nonlinearities after the encoder is more conducive to the contrastive loss and improves representation learning generally, since the contrastive loss causes a loss of information [9]. Intuitively, we consider the encoder network as a way to learn relevant features about clothing items, such as fit, color, and texture, and what they represent, and the projection head learns how to combine these embeddings in a way that maximizes compatibility.

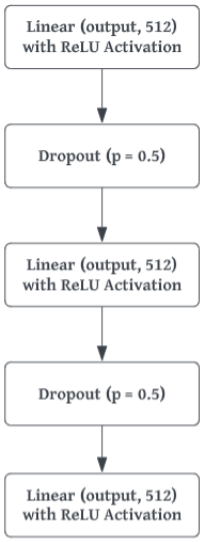

Figure 4. Projection Head Architecture

For our projection head, we use a three-layer multi-layer perceptron with ReLU activations, separated by dropout layers with 50% dropout to prevent overfitting, as shown in Figure 4.

3.2.3 Similarity Metric

Like other similarity metrics, we use the normalized dot product or cosine similarity between two embeddings. Therefore, we define the similarity between two items as:

$$s(a, b) = \frac{a \cdot b}{||a||||b||}$$

The similarity between three items is the sum of the pairwise dot product. In other words, the similarity between top t_i, bottom b_j, and shoes s_k is:

$$sim_{ijk} = s(t_i, b_j) + s(t_i, s_k) + s(b_j, s_k)$$

We do not take the similarity between items of the same category.

3.2.4 Contrastive Loss with Temperature Scaling

Unlike other outfit generation paradigms, we do not present the model with negative examples explicitly. Instead, negative examples are presented in every batch during training. For a given outfit, we define a negative example as a triple that contains at least one (but not all) of the items of the

correct outfit. For example, for a given triple (t_i, b_i, s_i), a negative example is (t_j, b_i, s_i) where $j \neq i$. Then, there are $3N(N-1)$ negative examples per outfit: there are N options for each item in the triple, of which $N-1$ are incorrect. The number of triples containing three incorrect items is $(N-1)^3$. Using complementary counting, we see that the number of triples containing at least one but not all three correct items is the total number of triples minus those containing zero correct items and those containing all three (the correct outfit), which is $N^3 - (N-1)^3 - 1 = 3N^2 - 3N = 3N(N-1)$. These negative samples are then presented to the model, and we use them to aid in training as shown in Figure 5. The larger the batch size, the more the model benefits from exposure to negative samples. We vary the batch size between 32 and 64 for this study; however, in practice, batch sizes can reach up to 1000 samples per batch. In our case, since the number of negative samples is polynomial in the size of the input, we do not increase the batch size to be greater than 64.

We calculate the pairwist dot product between all tops and all bottoms, then all tops and all shoes, and all bottoms and all shoes and add them together to obtain our final similarity matrix, as shown in 6.

We then apply temperature-scaled cross entropy loss as shown below to the outputs of the batch. There are N classes– one for each outfit, and the output of the batch-wise similarity calculation is of size $N \times N$. So, the loss function for a triplet of items is

$$l(t_i, b_j, s_k) = \frac{exp(sim_{ijk/\tau})}{\sum_{i,j,k} exp(sim_{ijk}\tau)} \quad (1)$$

τ is a tunable temperature parameter, and a smaller temperature parameter causes the distribution of similarity scores to spread out further, while a larger temperature parameter causes them to contract. We experimented with temperature parameters of 0.01, 0.1, and 1 and find that a temperature of 0.1 works the best, which is in line with previous works.

3.2.5 Evaluation Metrics

Previous works evaluate the accuracy of their network by actually deploying these outfits on commercial websites and measuring consumer interaction with the outfits. However, since we do not have access to such metrics, we instead use a validation set of 1000 outfits that the model has never seen before, and evaluate on the top-5 accuracy per batch. However, it is unclear if this evaluation metric is the best way to measure accuracy. Since items can be part of multiple outfits, and items in outfits are often interchangeable, it is possible that the network assigns high similarity scores to clothing items that are very close to the "correct" item in the dataset but are not exactly the same. In this case, although the outfits generated are plausible, the accuracy might still be low. To account for this problem, we use the top-5 accuracy per batch instead of the top-1, and we also manually check the outfits for plausibility.

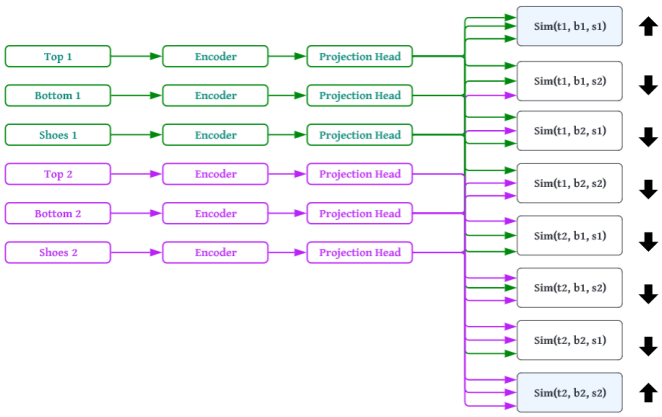

Figure 5. Contrastive Learning with batch size $N = 2$ different outfits. The `sim()` scores is the pairwise dot product of the projections of every item in the outfit. We want to maximize the similarity scores of outfit groups (e.g. `(t1, b1, s1)`) and minimize the similarity scores of non-outfit groups (e.g. `(t1, b2, s2)`)

3.3. Training Configurations

We train all models for 50 epochs, using Adam [6] as the optimizer and learning rates of 5×10^{-5} when modifying the pretrained networks and 10^{-4} as the learning rate when training the projection head. All models were trained on a single nVidia GEForce RTX 3060 using the Pytorch library.

3.4. Beam Search

To generate real outfits using the model outfit embeddings, we use an efficient searching strategy known as beam search to find the combination of top, bottom, and shoe with the smallest possible distance between them (the optimal outfit). Beam search works avoids searching over every combination of outfit possibilities, which grows exponentially. It works as follows: when given an input garment (e.g., without loss of generality, a top), it searches over the next category of garments (e.g. bottoms) and identifies the closest k garments in that category that match. It then repeats this search over the last category of garments using these top k item pairs as inputs.

Closeness is defined in this scenario as the sum of the pairwise dot products of the outfit embeddings (see eq 1). It can be defined differently, but we found that this method works best.

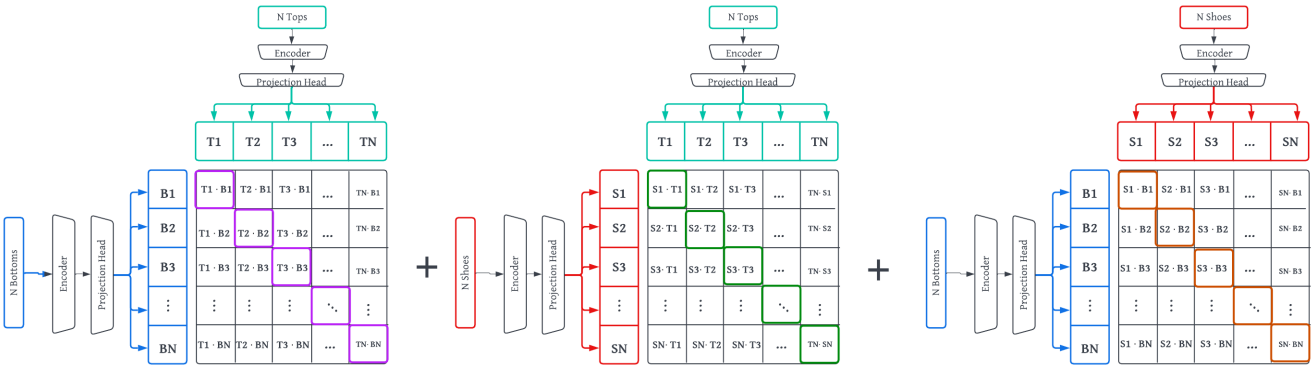

Figure 6. Similarity Matrix with N outfits

4. Results

4.1. Encoders

4.1.1 Frozen Encoder vs. Finetuned Encoder

We train a ResNet18 as the base model with the projection head as described above; we either freeze the base model or use the contrastive loss to update the weights of the pretrained model with a very small learning rate, to prevent overfitting. We find that the two models perform approximately equally, with a top-5 accuracy rate of 40% with the frozen encoder and 37% with the finetuned ResNet18. We conclude that the ResNet18 finetuning may have been helpful with more data, since the number of outfits in our training set was small. We also test on smaller networks to determine whether any overfitting was caused by complexity of the model.

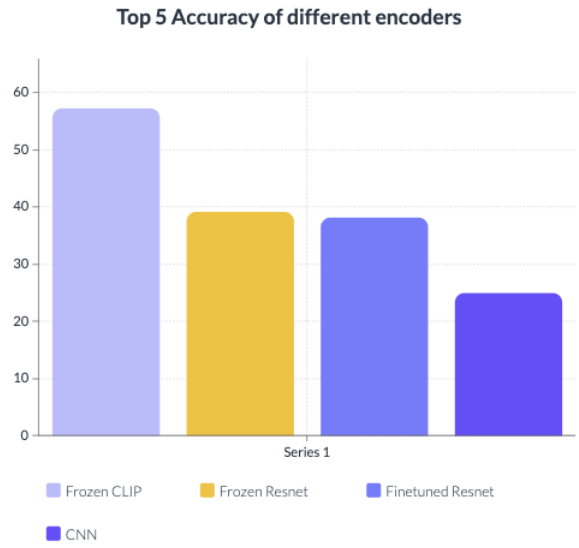

Figure 7. Top 5 Accuracy with different encoders

4.1.2 CLIP Embeddings

For our second experiment, we use the CLIP encoder with frozen weights as the initial encoder and train the contrastive component. Here, we find that the top-5 accuracy reaches 57%. As expected, CLIP outperforms both the Resnet and the CNN as shown in 7, since CLIP uses both text and image embeddings, while the other methods only use image embeddings. Since we use different evaluation metrics from other papers, which use fill-in-the-blank evaluation (where the model tries to predict a missing item in the outfit) or consumer metrics, it is difficult to determine how well StyleAI performs relative to other methods. However, we find qualitatively that the top 5 outfits are quite similar, meaning that the model is learning an accurate representation of outfits and clothing. Using the CLIP embeddings far outperforms all of the other methods of embedding, as shown in 9.

4.1.3 Convolutional Neural Network

Finally, we train a small CNN from scratch with the architecture shown in 3. The goal here is to use this CNN as a baseline for what to expect when models are trained from scratch. Additionally, results from the CNN can be used to determine how the the outputs of the pretrained encoders contribute significantly to the model's understanding of outfits. Training the CNN from scratch performs significantly worse than using either the CLIP embeddings or the ResNet, as expected, validating that our approach to embedding the clothing items is more effective. A comparison of all of the encoders is shown in 7 and 8.

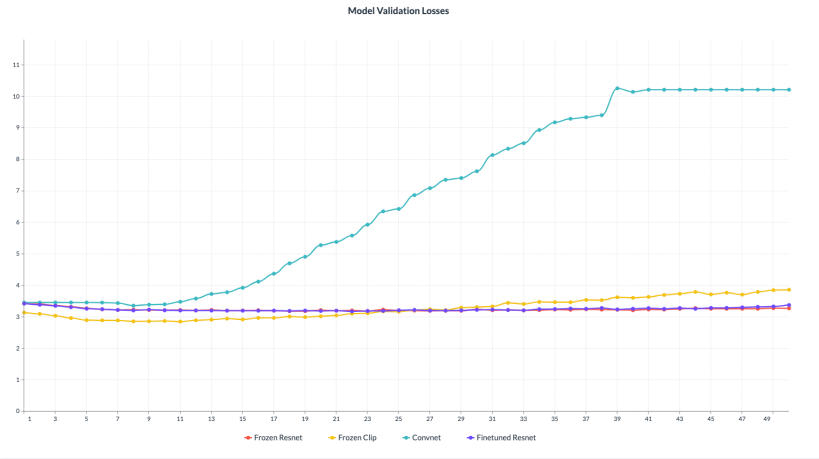

Figure 8. Validation loss of all encoders

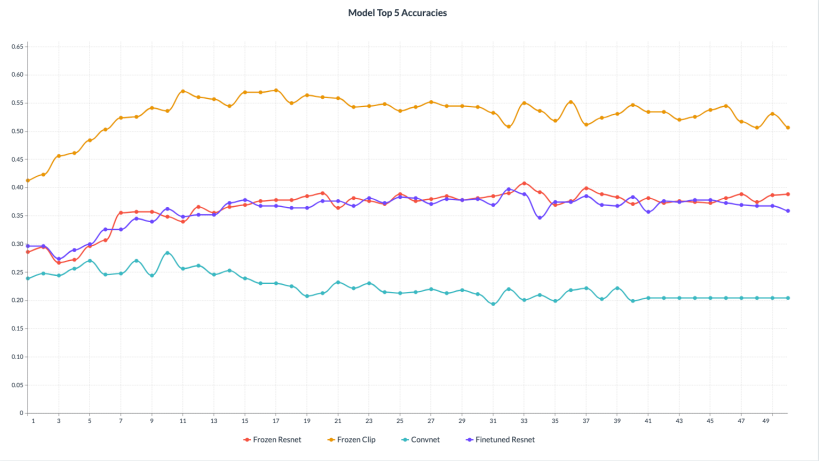

Figure 9. Validation top-5 accuracy of all encoders

4.2. Projection Head

We experiment with various projection head architectures and outputs; namely, with the output size of the projection head embedding, which represents the components of the outfits in the outfit feature space. We find that an output embedding size of 512 performs the best, over output sizes of 256, 128, and 64. With the frozen CLIP architecture, an output embedding size of 512 had 57% accuracy, while smaller embeddings performed worse (a 256-dimensional output projection embedding had a 54% accuracy).

4.3. Batch Size

We find that the batch size has a very large impact on the training performance. We vary the batch size between 32 and 64 and find that with a batch size of 64, the top-5 accuracy drops drastically to around 40% when using the frozen CLIP as the encoder, as opposed to 57%. We believe that this is due to the number of negative samples presented to the model per outfit being polynomial in the input– i.e. for a batch size of 64, the model is being presented with over 12K negative samples per outfit. Since items between different outfits may look extremely similar, the likelihood that some of the "wrong" outfit items are very similar to the correct outfit items grows larger as the batch size increases, and therefore the model may not be able to learn these effectively.

4.4. Problems With Overfitting

As shown in figure 11, all of the models we trained experienced high rates of overfitting. However, in most cases, this resulted in the validation losses increasing but the validation accuracies staying constant. We believe that the reason for this peculiar overfitting is that currently, our loss and accuracy functions optimize for a single correct answer: i.e. any outfit that does not exactly contain the items shown in the dataset is penalized in the loss. However, if

Figure 10. Outfits on the left (a) are the ”correct” outfits in the dataset, while outfits on the right (b) are incorrect yet closely matching outfits predicted by the model.

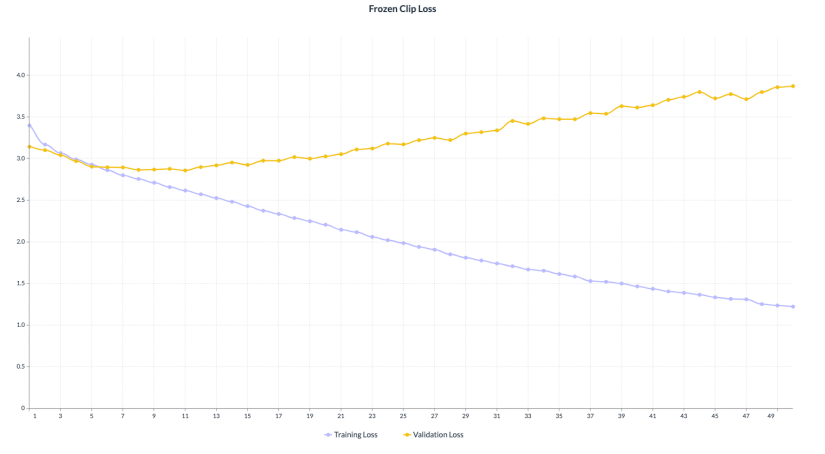

Figure 11. Training vs. Validation Loss for Frozen CLIP Encoder

the model truly learns a fashion sense, then it might generate perfectly acceptable outfits that do not exactly match the dataset. Therefore, as stated before, this problem does not have a ”single” correct answer, and forcing the model into a paradigm that enforces a correct answer causes the loss function to increase drastically. We believe that therefore, the accuracy and loss metrics used in this paper provide a good approximation of model performance, but that there are other metrics that could be used instead that might reflect model performance more accurately. As seen in figure 10, even when the model predicts the wrong outfit, the predicted outfit is often extremely similar to the actual outfit. Therefore, the loss and accuracy metrics could be further improved to take this into account. For instance, instead of solely optimizing by applying the temperature-scaled contrastive loss, we could also compute a similarity metric between the predicted generated embeddings with the original generated embeddings to determine whether the model predicts similar enough clothing items, even if they are not exactly the same. This would reduce the overfitting, since we are no longer encouraging the model to memorize outfits, and measure the model's ability to predict plausible outfits

that do not exactly match the original.

5. Conclusion

We conclude that StyleAI, a novel self-supervised method that uses contrastive learning to learn outfit compatibility, performs well as a stylist and is able to learn not only what features are crucial to understanding the representation of a clothing item, but also how items fit together to create cohesive and fashionable outfits. We develop a novel loss function that maximizes compatibility between items of the same outfit, and show that even when the model does not predict the same outfit as shown in the dataset, it predicts a plausible, fashionable outfit that shares many of the same characteristics (color, texture, etc) with the "correct" outfit items. Our model is strengthened by separate item and outfit embeddings whose spaces do not overlap — a significant improvement over prior work. Using beam search, we are able to search over the space of clothing given to find the optimal outfit given an item in the outfit. After experimenting with various encoder and projection head architectures, we obtain a 57% top-5 accuracy and even better qualitative results. Our results indicate that our model architecture and data embeddings are successful and have potential to be useful in future work in this area.

6. Contributions

Nithya Attaluri helped design, implement and debug the models. She implemented the beam search algorithm to search over the outfit embedding space and translate model outputs into outfit images. She also contributed to the writing of this report.

7. Acknowledgements

We would like to thank Prof. Freeman, Prof. Isola, and all of the 6.869 TA's for their guidance and support over the past sxemester. We would also like to thank Prof. Daniela Rus and Dr. Alexander Amini, as well as the Distributed Robotics Lab at CSAIL for providing us with access to their compute cluster for model training.

References

[1] Elaine M. Bettaney, Stephen R. Hardwick, Odysseas Zisimopoulos, and Benjamin Paul Chamberlain. Fashion outfit generation for e-commerce, 2019. 2

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020. 2

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015. 1

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. 1

[5] Xintong Han, Zuxuan Wu, Yu-Gang Jiang, and Larry S. Davis. Learning fashion compatibility with bidirectional LSTMs. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, oct 2017. 2

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 5

[7] Takuma Nakamura and Ryosuke Goto. Outfit generation and style extraction via bidirectional lstm and autoencoder, 2018. 2, 3

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. 2

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. 2, 4

[10] Devashish Shankar, Sujay Narumanchi, H. A. Ananya, Pramod Kompalli, and Krishnendu Chaudhury. Deep learning based large scale visual recommendation and search for e-commerce. *CoRR*, abs/1703.02344, 2017. 2

[11] Mariya I. Vasileva, Bryan A. Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar, and David Forsyth. Learning type-aware embeddings for fashion compatibility, 2018. 2