Do these go well together? - Learning a Clothing Style Evaluator and visualizing the style space

Daniel Schappler[1]

Abstract
There is a small but important difference between visual similarity and visual *style* similarity: Two visually very different items (a leather belt and a leather purse, for example) can very well be stylistically similar, while visually similar shoes (cowboy boots, gumboots and UGG boots, for example) are hardly stylistically similar. In this project, we learn a feature transformation of clothing and accessory product images into items in a latent, 128-dimensional "style space" that understands compatibility in the sense of *stylistic similarity*. The questions we are able to answer with our trained network are not of the form *"Are these items shoes?"* or *"Is there a belt in this picture?"*, but rather *"Do these shoes match this belt?"* or *"Are these two women's dresses the same style?"*. We achieve this feature transformation by means of training a siamese convolutional neural network with example pairs, each consisting of two product images that are either similar or dissimilar in style.

Keywords
Siamese network architecture — Convolutional neural networks — Transfer learning — t-SNE

[1]**Corresponding author**: schapplerd@gmail.com

Contents

1. Definition

1.1 Project Overview

Humans have a very sophisticated sense of relationships between clothing items based on their visual appearance: Two items can be complementary, alternatives or just not matching at all.

In order to learn a similar sense from product image data, our goal of this project is to obtain an embedding for visual style similarity of clothing and jewelry items. The final system we develop should be capable of recommending which clothes and accessories go well together and which ones do not.

The field of deep learning, especially *convolutional neural networks (CNNs)*, was able to make huge improvements in recognizing objects in images. In our project, we will make use of one recent, very successful CNN and take it one step further: We learn a mapping that is able to not only detect visual similarity, but can mimic the above-mentioned human notion of visual *style similarity* as well. This means, our goal is to transfer product images into a latent style space, which meets the following two requirements:

1. Images of items that go well together are close and

2. Images of items that do not match are far apart.

The data set we use to train our system uses co-purchase data from *amazon.com* as a proxy measure of style compatibility, as it aggregates customer's preferences and therefore, and that is the underlying hypothesis, also the customers' consensus on style.

We can then use this learned system to query our own product image pairs and receive a measure of stylistic similarity.

1.2 Problem Statement

Transfer learning is a common technique used in deep learning: Instead of training a very large convolutional neural network on a very large data set from scratch, we can make use of already fully-trained CNNs that have successfully proven to be very powerful. In this project, we use the recent successful network architecture VGG16 (where VGG stands for *Visual Geometry Group*, and 16 is the amount of weight layers of the network, see Simonyan & Zisserman [9]) that was trained for object recognition on a large-scale data set (ImageNet Large-Scale Visual Recognition Challenge, ILSCV2014) and use it and its learned weights as a fixed feature extractor on our training images.

In order to convert this network into one that can be used for our embedding, we remove all fully-connected layers and the softmax layer at the end of the version that is used for object classification, and receive a 25,088-dimensional feature vector (a flattened 7x7x512 tensor) as an output for each input image. The detailed resulting architecture can be seen in Figure 1.

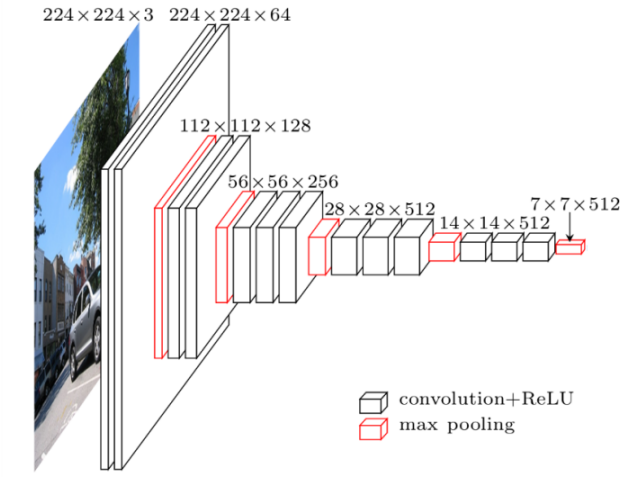


Figure 1. The pruned VGG16 network architecture (see [9]) for feature extraction will be one part of our complete network architecture.

In order to obtain our required embedding (the latent style space), Hadsell et al. [4] and Chopra et al. [2] show that an efficient method for solving this task is training a siamese network architecture. This framework, paired with the so-called *contrastive loss function* (see Section 2.3) can be used to train a similarity metric from real data. In our case, it uses two copies of the pruned VGG16 with *freezed* (=fixed) weights. On top of each of them, we place a fully-connected layer with 128 nodes that share the same parameters, both when initialized and while training. The entire network architecture (see Figure 2) can now be viewed as a new network which inputs a pair of images and outputs a similarity metric, which then can be measured against the pair's true label.

With training, we fine-tune this new top layer with the help of our positive and negative example pairs. The loss function that is minimized while training minimizes the euclidean distance in the lambda layer (also called the cost-module) of two images of a positive pair and maximizes this euclidean distance when two images are a negative pair. Because both strands of a siamese network have the same parameters, this similarity metric is symmetric (independent of the order of the two images within the pair).

1.3 Metrics

As our goal is to give our style evaluator a pair of pictures and receive a score indicating stylistic similarity (and, based on that, a clear statement of either "these go well together" or "these do not match"), we are dealing with a binary classification task.

A common performance metric for such a task is the accuracy on the test data set defined as

$$\text{Acc} := \frac{\text{True Positives} + \text{True Negatives}}{\text{Number of all examples}}.$$

In our case, because we are dealing with a heavily unbalanced data set (roughly 94% negative pairs and 6% positive pairs, see Section 2.1), even the trivial predictor "these do not match" (so 100% negative predictions) would get a high accuracy score of 94%. Also, we would like to have some risk management metric that give us information about type-I- and type-II-errors.

Therefore, we use some related metrics that contain more detailed information. The *True Positive Rate (TPR)* is defined as

$$\text{TPR} := \frac{\text{True Positives}}{\text{Number of positive examples}},$$

and similarly the *False Positive Rate (FPR)* is calculated as

$$\text{FPR} := \frac{\text{False Positives}}{\text{Number of negative examples}}.$$

The commonly used *Receiver Operator Characteristic Curve (ROC-curve)* visualizes the trade-off between FPR and TPR as we vary the parameter for the classification criterion (the cutoff-value for predicting "positive" vs "negative"). What will ultimately give us a performance score for our models will be the area (*AUC*) under the ROC-curve for the predictions on the held out test set (see Section 4).

2. Analysis

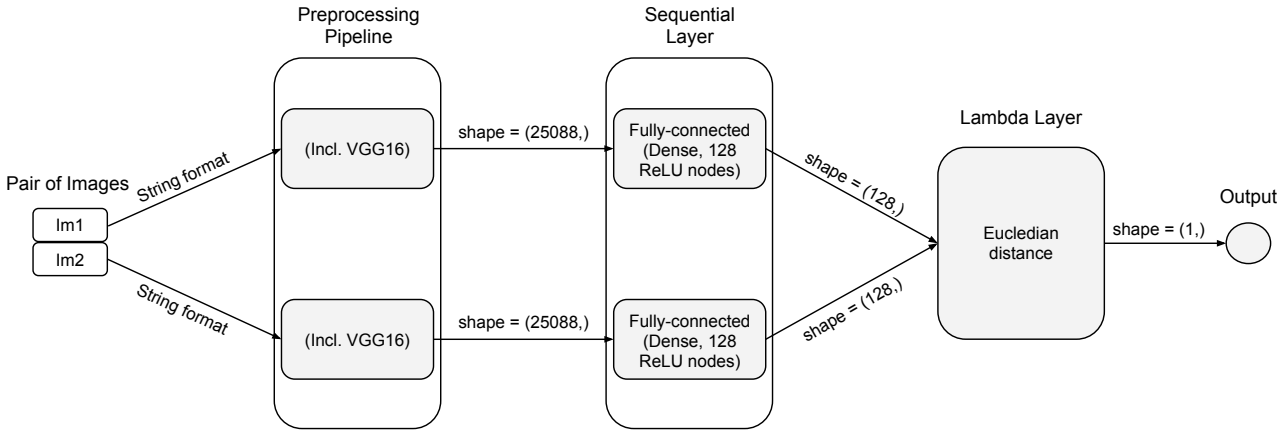


Figure 2. The siamese network architecture used in this project.

2.1 Data Exploration

A very common approach to acquire label information to an image data set is using hand-labeled annotations, which is often very infeasible considering time and effort. Because of this, these data sets are usually comparably small. But there is an alternative: Rather than relying on manually annotated images, we use a freely available source of large amount of data which is relevant and related to the true information we are looking for:

We use a processed subset of the data set used in Veit et al. [12] (available at `https://goo.gl/blgCC2`), which itself uses the large scale data set provided from McAuley et al. [7]. The latter consists of product images from *amazon.com*, their respective product categories and product co-purchase information. A large majority of the images are iconic and have a white background, only some products are shown in a full-body picture (for a visual sample of our data set see Figure 3).

Veit et al. [12] only focus on the "Clothing, Shoes and Jewelry"-category and its respective subcategories. As they are explicitly interested in cross-category fit, they use a strategic method to sample training data where pairs of data are so-called "heterogeneous dyadic co-occurences" (the two elements of a pair belong to different high-level categories and frequently co-occur).

They create these pairs as follows: After removing duplicates and images without category labels, the positive pairs are created via co-purchases and the negative pairs are sampled randomly among those not labeled as compatible. Therefore, they assume that these negative pairs will be incompatible with high probability.

This means, the measure of compatibility is the aggregated co-purchase data from *amazon.com* – either products that are frequently bought together or products that were bought from customers who also bought another specific product. Therefore, the relationship between two products is not coming directly from the customer, but reflect amazon's recommendations which are based on item-to-item collaborative filtering (Linden et al. [6]).

This makes clear that we operate under the following hypothesis when using this data set with respect to our problem: *We expect the aggregated user behavior data to recover the compatibility relationships between products. Buying implies liking, but not buying does not necessarily imply not liking. specifically, two items that are not labeled as compatible are not necessarily incompatible* (see [12]).

Total number of image pairs	35,000
Number of positive pairs	2,040 (5.8%)
Total number of images	70,000
Total number of distinct images	55,565
# of images that only appear once	44,206
# of images that appear in 2 pairs	9,483
# of images that appear in 3 pairs	1,503
# of images that appear in 4 pairs	2,32
# of images that appear in 5 pairs	49
# of images that appear in more than 5 pairs	see Figure 3

Table 1. Summary statistics of our data set.

Table 1 shows some important summary statistics of our data set and Table 2 shows a raw sample. We see that most of the products only appear once in the whole data set, which is not ideal for the task of learning relationships of one product to many other products. The obvious solution is to use more data (meaning generating more pairs), but we refrain from doing this because of computationally scarce resources (see Improvements, Section 5.3).

pic1	pic2	score
41dTWz5u3OL._SX395_.jpg	516EJv4m1XL._SY300_.jpg	0
41JvUmCiEoL._SX395_.jpg	41B8Xw-YyUL._SX300_.jpg	0
41wRSnJ-qOL._SY445_.jpg	41i1viWuSKL._SX342_.jpg	0
31vnEoTtr0L._SY445_.jpg	41Y459ZO9GL._SY445_.jpg	1
41Jh1oxfyEL._SY395_.jpg	41QEtd5wQUL._SX342_.jpg	0

Table 2. The first five rows of our data set.

2.2 Exploratory Visualization

Figure 3 gives a visual overview of the remaining data set statistics (amount of images that appears in n pairs for $n \geq 6$). We see that the trend continues, and only a few specific product images appear in many pairs. These images will be well-placed in our resulting embedding, because they have a clear relationship to many other images. Ideally, this would be the case for all images (see Improvements, Section 5.3).

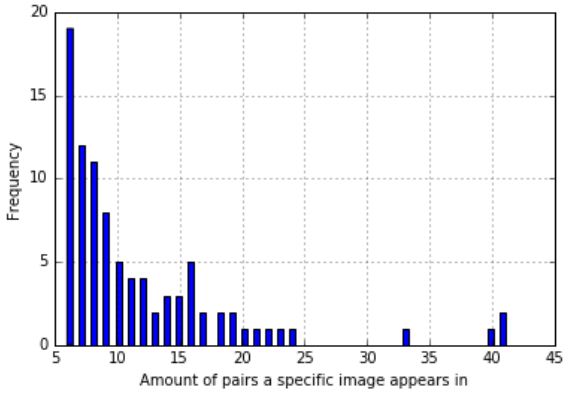


Figure 3. .

Complementary to Table 2, there is another sample from our data set visualized in Table 3, where again a score of 0 indicates a "dissimilar" style and a score of 1 means that the products shown in both images of the pair have a similar style.

2.3 Algorithms and Techniques

As part of preprocessing, each image is passed through a stack of convolutional layers and max-pooling operations (see Figure 1), using filters with receptive fields of 3x3. All weight layers are equipped with *Rectified Linear (ReLU)* non-linearities. Because of their non-saturating properties, these neurons make the training of deep networks a lot faster than training with tanh or sigmoid neurons (see Krizhevsky et al. [5]). Their activation function is of the form $f(x) = \max\{0, x\}$, which is also the activation function of our choice for our fully-connected layer on top of each VGG16 in our siamese network (compare Figure 2). These fully-connected top-layers are initialized with zeros for all bias values and a Glorot-uniform-initialization (Glorot & Bengio [3]) for all other weights. The contrastive loss function consists of two penalties and has the form

$$\begin{aligned} L(y,\theta) =& y \cdot \|G_\theta(x_1) - G_\theta(x_2)\|_2^2 \\ &+ (1-y) \cdot \max^2\{0, m - \|G_\theta(x_1) - G_\theta(x_2)\|_2\} \end{aligned}$$

where x_1, x_2 are the respective images of the pair to evaluate, G_θ is the neural network with weight parameters θ and 128-dimensional output (compare Figure 2), $y \in \{0,1\}$ the binary label (score) of the pair an m the *margin* (explanation see below).

As a consequence, this function has a different impact on the two different types of training examples – positive (similar) and negative (dissimilar): Its gradient pulls together stylistically similar examples, and pushes apart stylistically dissimilar examples that are not yet at least m apart ($L = 0$ for $y = 0$ and $\|G_\theta(x_1) - G_\theta(x_2)\|_2 \geq m$). The margin m can be chosen arbitrarily, as a CNN is able to learn to scale its embedding proportional to m. Nevertheless, the downside of

pic1	pic2	score
		0
		0
		1
		1

Table 3. Visual sample of our data set.

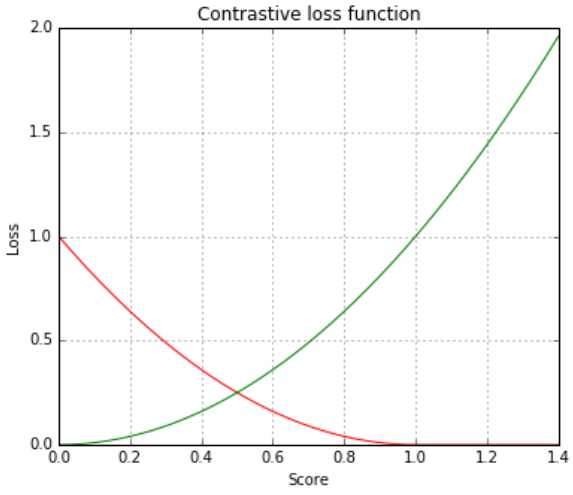


Figure 4. The contrastive loss function $L(y,\theta)$ with $m = 1$ for similar pairs (green), dissimilar pairs (red) and their score $\|G_\theta(x_1) - G_\theta(x_2)\|_2$.

making m too large is risking instabilities, and the downside making it too small is very slow learning. After testing several different values, we choose $m = 1$ in our implementation.

We train on 21,000 image pairs (60% of the whole data set, split in a stratified manner) and cross-validate on 7,000 held out pairs (20% of whole data set). The gradient of the contrastive loss function w.r.t. the parameter vector is computed via backpropagation and updated with RMSProp (a Stochastic Gradient Descent variant, updating the learning rate based on recent gradients, see Tieleman & Hinton [10]) using the sum of the gradients contributed by the two sub-nets. To speed up training even more, we use mini-batch updates with a size of 128 training pairs each.

2.4 Benchmark

As our performance metric is the area under the ROC curve on the held out test data set (see Section 1.3), the most basic benchmark to beat would be an AUC greater than 0.5 (which means our model would be, in the long run, more precise than a pure 50/50-coinflip to determine the result – "similar style" or "dissimilar style").

But knowing the background and nature of VGG16, which is part of our preprocessing and will not be trained, we can make the following argument: As VGG16 has at least some kind of notion of color and shape(s), we expect the initialized – but untrained – model to have an AUC that is slightly greater than 0.5.

Therefore we choose the initialized, untrained model's AUC on the test data set as our benchmark to beat.

3. Methodology

3.1 Data Preprocessing

Figure 5 and Table 4 show a step-by-step explanation of our preprocessing pipeline, whose output is the pair data which in turn is the input data to the trainable siamese network (see Figure 2).

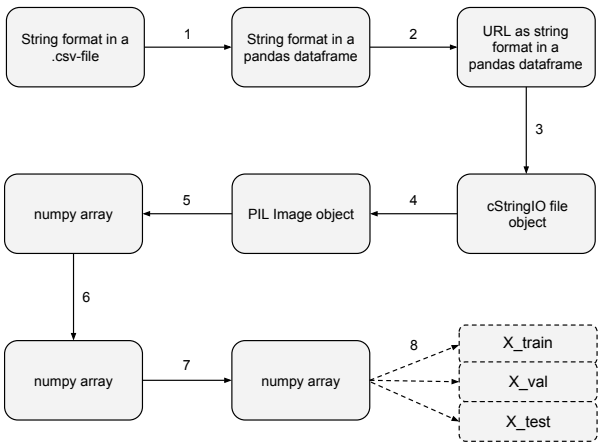


Figure 5. Preprocessing pipeline used for each image before training.

- **Steps 1-4** convert the mere strings from the csv-file into valid image-URLs and loads them as PIL image RGB objects of the required size (224x224) for the next steps.
- **Steps 4-5** convert the PIL image object into an array, including the preprocessing steps that are necessary for an input to VGG16 (subtracting the mean RGB value of the ILSVRC training set (see [8]) from each pixel).
- **Step 6** pushes the image through VGG16, which returns a 25,088-dimensional array.
- **Step 7** scales the values of this array into the $(0,1)$-interval via MinMax-scaling.
- **Step 8** assigns each image pair to either the training- (60%), validation- (20%) or test-set (20%) in a stratified manner (as we are dealing with a highly unbalanced data set). We are not using cross-validation due to feasibility reasons (see Improvements, Section 5.3).

Step	Function name	Input type (shape)
1	`read_csv()`	String in .csv
2	String concatenation	String in data frame
3	`StringIO()`	String in data frame
4	`load_img()`	cStringIO file oject
5a	`img_to_array()`	PIL image (224 x 224)
5b	`preprocess_input()`	numpy array (1,3,224,224)
6	`vgg16.predict()`	numpy array (1,3,224,224)
7	Feature scaling (MinMax)	numpy array (1,25088)
8	`train_test_split()`	numpy array (1,25088)

Table 4. Data preprocessing steps, including the convolutional neural network VGG16 (see [9]).

3.2 Implementation

The basic algorithm for training the siamese network in `siamese_net.py` can be summarized as follows:

```
while validation loss is decreasing:
    for each pair (x1,x2) in training set:
        if label == 0:
            update parameter vector to
            increase ||Gθ(x1) − Gθ(x2)||2
        if label == 1:
            update parameter vector to
            decrease ||Gθ(x1) − Gθ(x2)||2
```

Only continuing training while the validation loss is decreasing at a certain rate is called *early stopping* and is a commonly used regularization method (see Refinement, Section 3.3).

After training and refining, we save and upload the best performing model to the internet, as it is the core of the Clothing Style Evaluator application and should therefore should be reachable for anyone wanting to have their style evaluated. This application is a different script and is also embedded in a Jupyter notebook for any users who do not want to work in a command line environment.

`style_evaluator.py` loads the saved model and a certain constant value that is needed for rescaling purposes. A user can choose between loading an image from a local drive or from the internet in the form of an image-URL. After the user uploaded the two pictures of the pair he wants to get evaluated, they get preprocesses be an input to the incorporated VGG16 and then pushed through the trained siamese network. The output is a scalar value that measures the distance between the two queried images in the 128-dimensional style space. Depending on this value, combined with the findings from Section 4 below, the Evaluator prints one of the two results *"Nice, this looks good together!"* or *"Sorry, these styles do not match."*.

In order to visualize the 128-dimensional style space as a 2D-embedding (similar to the idea of Bell & Bala [1], who did the same with a data set of furniture and interior design product images), we implement a dimensionality reduction from 128D to 2D via the t-SNE algorithm in `tsne_vis.py`. We break up the siamese architecture and remove the top layer that computes the euclidean distance. We also break up the pair structure of our data, as now the images are pushed one by one through a copy of only one the siamese strands (see Figure 2). The output is distinct 2D-coordinates for each product image, so we are able to embed the original RGB images in a scatter plot visualization (see Section 5.1).

3.3 Refinement

The learning curves of the first and initial trained model (Model I) that was implemented as explained above can be seen in Figure 6 (a). It is instantly obvious that we do not need to increase the capacity of the model any more when refining, because the training loss goes down very quickly and is approaching zero quite fast. It also becomes obvious that we are substantially overfitting, as the validation loss increases again after only a few epochs of slowly decreasing and is overall on a very high level compared to the respective training loss.

For these reasons, we introduce two regularization techniques in order to avoid or prevent overfitting. *Early stopping* describes the method of only training until the validation loss is no longer decreasing for a certain amount of epochs, and *L_2-regularization* adds a penalty term to our contrastive loss function that will not allow the learned weights to get too big and therefore introduces an amount of bias.

As shown in Table 5, the validation loss decreases as we slowly increase the L_2-regularization step-by-step. Figure 6 (b) shows the learning curves of the model after 4 iterations of regularization increase (Model V). Both training and validation loss are reasonably low after stopping, and the gap between both is reasonably small, too – we found a good bias-variance-tradeoff.

This becomes especially obvious as we further increase the L_2-regularization parameter. Figure 6 (c) shows the learning curves of the over-regularized and underfitting Model VII. Both training and validation loss stay on a high level until stopping and cannot compete with the validation loss of the less regularized models (again see Table 5). This is an indicator of too high bias and suggests Model V as the model to use in our Clothing Style Evaluator application.

Model #	L_2-reg.	Train loss	Validation Loss
I	0	106	450
II	10^{-6}	120	422
III	10^{-5}	134	405
IV	10^{-4}	316	406
V	$1.25 \cdot 10^{-4}$	**331**	**403**
VI	$1.5 \cdot 10^{-4}$	404	413
VII	$2 \cdot 10^{-4}$	483	426

Table 5. Minimum validation loss and respective training loss for different magnitudes of L_2-regularization.

4. Results

4.1 Model Evaluation and Validation

After training, we now evaluate all seven models from Table 5 on the performance metric we defined in Section 1.3 – the ROC curves generated on another held out 7,000 image pairs (the remaining 20% of the whole data set). The resulting areas under these curves can be seen in Table 6, and mirror our findings with regard to the validation loss performances in the previous section.

Model #	L_2-reg.	Area under ROC
I	0	0.854
II	10^{-6}	0.865
III	10^{-5}	0.873
IV	10^{-4}	0.873
V	$1.25 \cdot 10^{-4}$	**0.884**
VI	$1.5 \cdot 10^{-4}$	0.875
VII	$2 \cdot 10^{-4}$	0.871

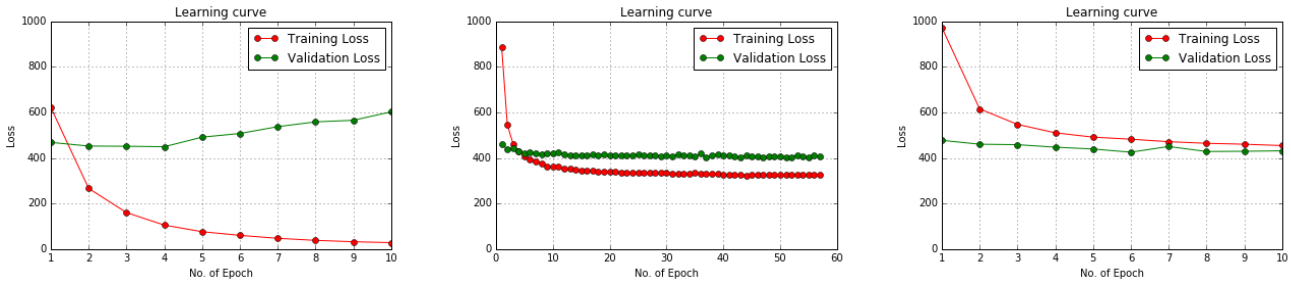
Table 6. Area under the ROC curve for different magnitudes of L_2-regularization.

Not only has Model V the highest area under the ROC curve compared to the other model versions, the absolute value of 0.884 also suggests, just like the training curves in the previous section, that Model V is able to generalize reasonably well to unseen data.

This confirms that it is not overly prone to small perturbations in input data and therefore is an acceptably robust candidate to use for our Style Evaluator application.

4.2 Justification

In Section 2.4 we chose the AUC of the initialized, untrained model as the performance benchmark. In Figure 7 we see, that its predictive power is terms of AUC is 0.584 and as expected, in the long run, indeed slightly better than random guessing. As seen in the previous Section, the fully-trained Model I (early stopping but no L_2-regularization) has an AUC of 0.854 and performs substantially better than the untrained



(a) Model I - high variance, overfitting. **(b)** Model V - a good bias-variance-tradeoff. **(c)** Model VII - high bias, underfitting.

Figure 6. Learning curves (training and validation losses over the epochs) as we increase the L_2-regularization.

model. Therefore, our best candidate, Model V (with an AUC of 0.884) does not only beat all other trained models, but the performance benchmark as well.

We have to keep in mind, that theses AUCs are a single data point for each model (as we did not implement cross-validation), so there can not be made any statements with regard to statistical significance. Nevertheless, the difference in AUC between benchmark and Model V is large enough to consider it very likely that the benchmark will also be beaten on other partitions of the data set.

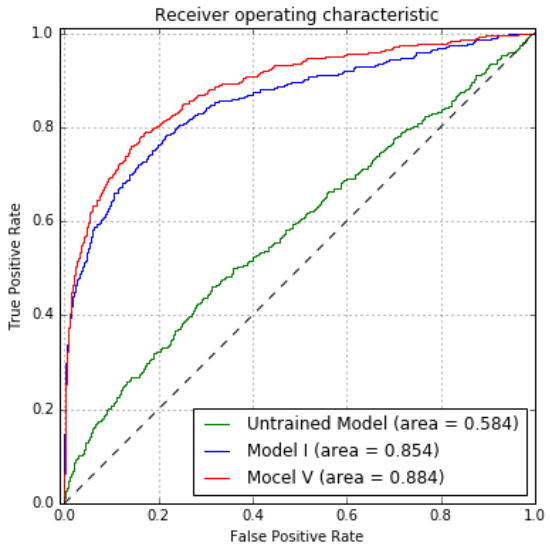


Figure 7. ROC curves of the unregularized Model I and the model with the best performance on the validation set (Model V), calculated on the held out test set.

As a final step, we choose a *cutoff score* (also called *cut-point*) c^*, which acts as the final decision boundary for the Style Evaluator's binary decision: Image pairs with distance score values on one side of the cutoff are labeled as positive (similar) and those with values on the other side are labeled negative (dissimilar).

The mathematically optimal cutoff score corresponds to the point of the ROC curve that is closest to the top left corner (where TPR $= 1$ and FPR $= 0$) and can be calculated as

$$c^* := \arg\max_c \{\text{TPR}(c) - \text{FPR}(c)\}.$$

In accordance to Figure 7, the optimal cutoff score in our case corresponds to a FPR of 0.2 and a TPR of 0.8 with the Style Evaluator labeling all product image pairs with a distance in the style space of greater than 0.91 as dissimilar.

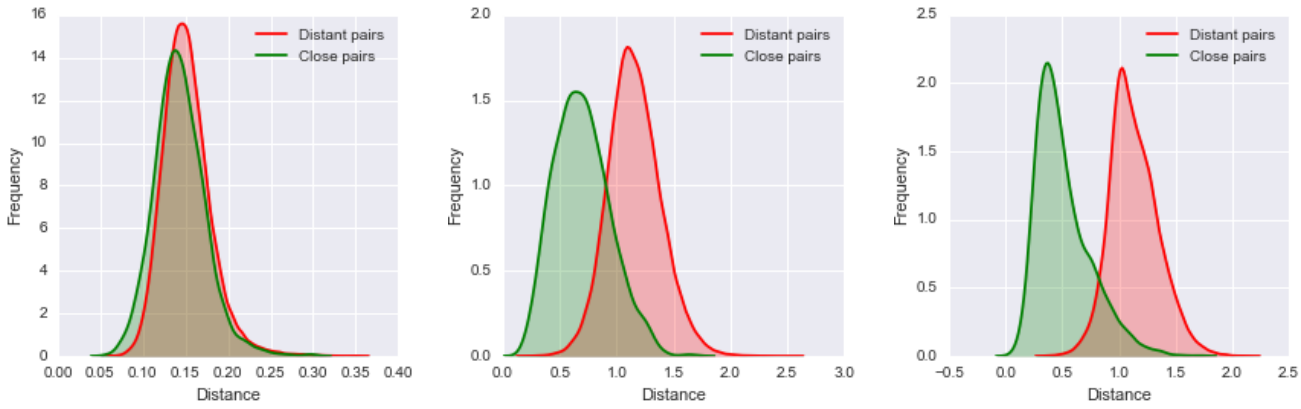
As the ROC curve describes the trade-off between TPR and FPR, we are also able to define other cutoff scores as we see fit based on our individual problem. Table 7 shows selected cutoff scores and their respective TPR and FPR.

FPR	TPR	Cutoff score
5%	59%	0.74
10%	70%	0.82
20%	80%	0.91

Table 7. Selected cutoff scores of Model V and their respective TPR and FPR for choosing the decision boundary of the Style Evaluator.

In our case, as we would like to have a reasonably small type-I-error (a low FPR) to not accidentally label items as style-similar (when in fact they are not) too often, we choose a cutoff score of 0.82 for our final implementation. The mathematically optimal solution has an FPR too high (20%) for our problem, while an even lower FPR of 5% will lead to an unsatisfyingly low TPR of 59%.

In accordance to our findings so far, Figure 8 confirms that our final model does a reasonably good job in learning style similarity: While being untrained, the model's predictions of item distance in the style space for the item pairs in the full data set are hardly distinguishable between similar (close) and dissimilar (distant) product pairs. The fully-trained model (which is used in our final application) has learned to keep these two different types of pairs apart to an acceptable degree. We can also clearly see the connections between (and the values of) distance cutoff score, TPR and FPR as evaluated above.



(a) Untrained Model V. **(b)** Model V after one epoch of training. **(c)** Fully-trained Model V.

Figure 8. Densities of distances in the style space between product images that are style-similar (green) and style-dissimilar (red) before, while and after training.

Finally, Table 8 shows the final model's prediction of the distance score of two previously unseen example pairs (not in the data set), which are in accordance to the author's personal perception of style similarity.

Image 1	Image 2	Score
		0.57
		1.28

Table 8. Example results of the final Style Evaluator. A score of smaller than 0.82 indicates style similarity.

5. Conclusion

5.1 Visualization

After we determined that Model V shows the best performance out of all evaluated models, we visualize the result by projecting our 128-dimensional embedding down to two dimensions using the t-SNE algorithm (v. d. Maaten & Hinton [11]). With the help of this algorithm it is possible to create a 2D-map that reveals structure of the underlying data. The best results can be obtained with high-dimensional data that lie on related low-dimensional manifolds. As we are dealing with images of objects from multiple classes, which are known to fulfill this condition, we can hope to preserve as much of the significant structure of the high-dimensional data as possible in the low-dimensional output map.

In Figure 9 we visualize our product embedding by pasting each image in its respective 2D location. This embedding is based on 1,000 randomly sampled images from the data set and is best viewed on a monitor. For the full resolution version, see `visuals/tsne_style_space.png` in the accompanying Github repository.

Even though the algorithm does not know the subcategory labels (shoes, underwear, dress, watch, ...) at all, we see that the products are generally organized by object subcategory. This makes sense in two ways: First, objects from the same subcategory generally share many visual features. Second, as already mentioned, a part of the network consists of VGG16, which naturally separates its feature space by object categories.

But, by looking closer, we can also observe that our model learned a more subtle notion of compatibility – style similarity: Business-casual sneakers, for example, are clearly separated from more sporty sneakers, although the visual differences are minimal.

5.2 Reflection

In this project, we used a very powerful convolutional neural network as a feature extractor, in combination with the siamese network architecture, in order to learn the notion of *visual style similarity* from clothing and jewelry product images paired with co-purchase metadata from *amazon.com*.

The learned parameter weights were then implemented into a Clothing Style Evaluator application, which automatically evaluates whether two items of clothing and/or accessories are stylistically similar.

Interestingly, despite training on a slow CPU and therefore only using a small data set due to feasibility reasons, our results are surprisingly good – Bell & Bala [1], for example,

train their furniture and interior design evaluator on millions of image pairs (see Improvements, Section 5.3).

5.3 Improvement

Although our results are already relatively satisfying, there is still room for improvement. We expect the overall performance to get even better by...

- ... using a larger data set (especially increasing the number of pairs a specific image appears in),
- ... using human-labeled pair data instead of relying on amazon data,
- ... using cross-validation for more precise model selection,
- ... not freezing the weight parameters of VGG16, but making them trainable as well,
- ... using a different CNN than VGG16 altogether (for example winning networks from more recent ILSVRC competitions),
- ... using more layers and/or mode nodes in the fully-connected top layer(s) to increase model capacity.

References

[1] S. Bell and K. Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. on Graphics (SIGGRAPH)*, 34(4), 2015.

[2] S. Chopra, R. Hadsell, and Y. Lecun. Learning a similarity metric discriminatively, with application to face verification. In *In Proc. of Computer Vision and Pattern Recognition Conference*, pages 539–546. IEEE Press, 2005.

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[4] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 1735–1742. IEEE, 2006.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, pages 1106–1114, 2012.

[6] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[7] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[9] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[10] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[11] L. van der Maaten and G. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9: 2579–2605, Nov 2008.

[12] A. Veit*, B. Kovacs*, S. Bell, J. McAuley, K. Bala, and S. Belongie. Learning visual clothing style with heterogeneous dyadic co-occurrences. *International Conference on Computer Vision (ICCV)*, 2015.

Figure 9. A scatter plot visualizing the learned 2-dimensional style space via t-SNE dimensionality reduction.