Learning visual similarity for product design with convolutional neural networks

Sean Bell Kavita Bala
Cornell University*

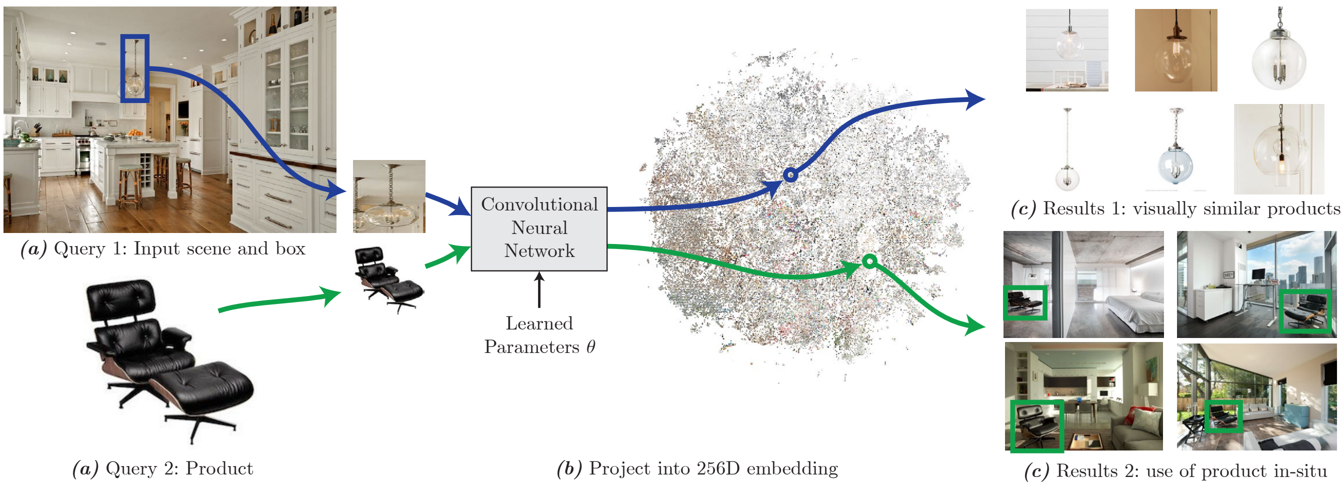


Figure 1: Visual search using a learned embedding. *Query 1: given an input box in a photo* ***(a)****, we crop and project into an embedding* ***(b)*** *using a trained convolutional neural network (CNN) and return the most visually similar products* ***(c)****. Query 2: we apply the same method to search for in-situ examples of a product in designer photographs. The CNN is trained from pairs of internet images, and the boxes are collected using crowdsourcing. The 256D embedding is visualized in 2D with t-SNE. Photo credit: Crisp Architects and Rob Karosis (photographer).*

Abstract

Popular sites like Houzz, Pinterest, and LikeThatDecor, have communities of users helping each other answer questions about products in images. In this paper we learn an embedding for visual search in interior design. Our embedding contains two different domains of product images: products cropped from internet scenes, and products in their *iconic* form. With such a multi-domain embedding, we demonstrate several applications of visual search including identifying products in scenes and finding stylistically similar products. To obtain the embedding, we train a convolutional neural network on pairs of images. We explore several training architectures including re-purposing object classifiers, using siamese networks, and using multitask learning. We evaluate our search quantitatively and qualitatively and demonstrate high quality results for search across multiple visual domains, enabling new applications in interior design.

CR Categories: I.3.8 [Computer Graphics]: Applications I.4.8 [Image Processing and Computer Vision]

Keywords: visual similarity, interior design, deep learning, search

*Authors' email addresses: {sbell, kb}@cs.cornell.edu

1 Introduction

Home owners and consumers are interested in visualizing ideas for home improvement and interior design. Popular sites like Houzz, Pinterest, and LikeThatDecor have large active communities of users that browse the sites for inspiration, design ideas and recommendations, and to pose design questions. For example, some topics and questions that come up are:

- "What is this {chair, lamp, wallpaper} in this photograph? Where can I find it?", or, "Find me {chairs, ...} similar to this one." This kind of query may come from a user who sees something they like in an online image on Flickr or Houzz, a magazine, or a friend's home.
- "How has this armchair been used in designer photos?" Users can search for the usage of a product for design inspiration.
- "Find me a compatible chair matching this table." For example, a home owner is replacing furniture in their home and wants to find a chair that matches their existing table (and bookcase).

Currently, sites like Houzz have active communities of users that answer design questions like these with (sometimes informed) guesses. Providing automated tools for design suggestions and ideas can be very useful to these users.

The common thread between these questions is the need to find *visually similar* objects in photographs. In this paper we learn a distance metric between an object *in-situ* (i.e., a sub-image of a photograph) and an *iconic* product image of that object (i.e., a clean well-lit photograph, usually with a white background). The distance is small between the in-situ object image and the iconic product image, and large otherwise. Learning such a distance metric is challenging because the in-situ object image can have many different backgrounds, sizes, orientations, or lighting when compared to the iconic product image, and, it could be significantly occluded by clutter in the scene.

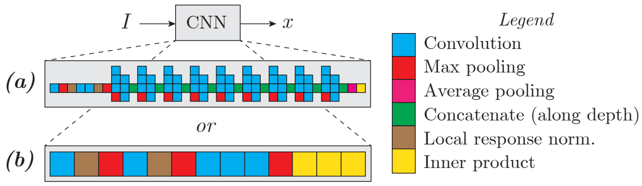


Figure 2: CNN architectures: ***(a)*** *GoogLeNet and* ***(b)*** *AlexNet. Either CNN consists of a series of simple operations that processes the input I and produces a descriptor x. Operations are performed left to right; vertically stacked operations can be computed in parallel. This is only meant to give a visual overview; see [Szegedy et al. 2015] and [Krizhevsky et al. 2012] for details including kernel sizes, layer depths, added nonlinearities, and dropout regularization. Note that there is no "softmax" layer; it has been removed so that the output is the D-dimensional vector x.*

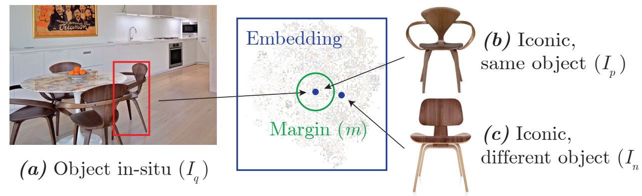


Figure 3: *Our goal is to learn an embedding such that the object in-situ* ***(a)*** *and an iconic view of an object* ***(b)*** *map to the same point. We also want different objects* ***(c)*** *to be separated by at least a margin m, even if they are similar. Photo credit: Austin Rooke.*

Recently, the area of deep learning using convolutional neural networks (CNNs) has made incredible strides in recognizing objects across a variety of viewpoints and distortions [Szegedy et al. 2015; Krizhevsky et al. 2012]. Therefore, we build our method around this powerful tool to learn functions that can reason about object similarity across wide baselines and wide changes in appearance. To apply deep learning, however, we need ground truth data to train the network. We use crowdsourcing to collect matching information between in-situ images and their corresponding iconic product images to generate the data needed to train deep networks.

We make the following contributions:

- We develop a crowdsourced pipeline to collect pairings between in-situ images and their corresponding product images.
- We show how this data can be combined with a siamese CNN to learn a high quality embedding. We evaluate several different training methodologies including: training with contrastive loss, object classification softmax loss, training with both, and the effect of normalizing the embedding vector.
- We apply this embedding to image search applications like finding a product, finding designer scenes that use a product, and finding visually similar products across categories.

Figure 1 illustrates how our visual search works. On the left are two types of queries: (1) an object in a scene marked with a box and (2) an iconic product image. The user sends the images as queries to the CNN we have learned. The queries map to different locations in our 256D learned embedding (visualized in the middle). A nearest neighbor query in the embedding produces the results on the right, our visually similar objects. For query 1, we search for iconic products, and for query 2 we search for usages of a product in designer scenes. We include complete results for visual search and embeddings in the supplemental and on our website (http://productnet.cs.cornell.edu).

2 Related Work

There are many bodies of related work; we focus on learning similarity metrics and visual search, and deep learning using CNNs.

Learning similarity metrics and visual search. Metric learning is a rich area of research; see [Kulis 2012] for a survey. One of the most successful approaches is OASIS [Chechik et al. 2010] which solves for a bi-linear similarity function given triplet judgements. In the area of graphics, metric learning has been applied to illustration style [Garces et al. 2014] and font similarity [O'Donovan et al. 2014]. Complementary to metric learning are *attributes*, which assign semantic labels to vector directions or regions of the input feature space. Whittle search uses relative attributes for product search [Parikh and Grauman 2011; Kovashka et al. 2012], Furniture-geek [Ordonez et al. 2014] understands fine-grained furniture attributes. In addition to modeling visual similarity, there are many approaches to solving the *instance* retrieval problem. For example, Girod et al. [2011] use the CHoG descriptor to match against millions of images of CDs, DVDs, and book covers. Many papers have built methods around feature representations including Fisher Vectors [Perronnin and Dance 2007] and VLAD [Jegou et al. 2012]. In contrast to the above approaches that mostly use "hand-tuned" features, we want to learn features end-to-end directly from input pixels, using convolutional neural networks.

In industry, there have been examples of instance retrieval for different problem domains like books and movies. Proprietary services like Google Goggles and Amazon Flow attempt to handle any arbitrary input image and recognize specific products. Other services like TinEye.com perform "reverse image search" to find near-duplicates across the entire internet. In contrast, we focus on modeling a higher level notion of visual similarity.

Convolutional neural networks (CNNs). A full review of deep learning and convolutional neural networks is beyond the scope of this paper; please see [Chatfield et al. 2014]. Here, we cover the most relevant background, and focus our explanation to how we use this tool for our problem. CNNs are functions for processing images, consisting of a number of stages ("layers") such as convolution, pooling, and rectification, where the parameters of each stage are learned to optimize performance on some task, given training data. While CNNs have been around for many years, with early successes such as LeNet [LeCun et al. 1989], it is only recently that they have shown competitive results for tasks such as object classification or detection. Driven by the success of Krizhevsky et al. [2012]'s "SuperVision" submission to the ILSVRC2012 image classification challenge, there has been an explosion of interest in CNNs, with many new architectures and approaches being presented.

For our work, we focus on two recent successful architectures: AlexNet (a.k.a. "SuperVision") [Krizhevsky et al. 2012] and GoogLeNet [Szegedy et al. 2015]. AlexNet has 5 convolutional layers and 3 inner product layers, and GoogLeNet has many more layers. Both networks are very efficient and can be parallelized on modern GPUs to process hundreds of images per second. Figure 2 gives a schematic of these two networks.

In addition to image classification, CNNs have been applied to many problems in computer vision and show state-of-the-art performance in areas including classifying image photography style [Karayev et al. 2014], recognizing faces [Taigman et al. 2014], and ranking images by fine-grained similarity [Wang et al. 2014]. CNNs have been shown to produce high quality image descriptors that can be used for visual instance retrieval, even if they were trained for object classification [Babenko et al. 2014; Razavian et al. 2014b].

3 Background: learning a distance metric with siamese networks

In this section, we provide a brief overview of the theory behind siamese networks and contrastive loss functions [Hadsell et al. 2006], and how they may be used to train a similarity metric from real data.

Given a convolutional neural network (such as AlexNet or GoogLeNet), we can view the network as a function f that maps each image I into an embedding position x, given parameters θ: $x = f(I; \theta)$. The parameter vector θ contains all the weights and biases for the convolutional and inner product layers, and typically contains 1M to 100M values. The goal is to solve for the parameter vector θ such that the embedding produced through f has desirable properties and places similar items nearby.

Consider a pair of images (I_q, I_p) that are two views of the same object, and a pair of images (I_q, I_n) that show different objects. We can map these images into our embedding to get x_q, x_p, x_n. If we had a good embedding, we would find that x_q and x_p are nearby, and x_q and x_n are further apart. This can be formalized as the contrastive loss function L, which measures how well f is able to place similar images nearby and keep dissimilar images separated. As implemented in Caffe [Jia et al. 2014], L has the form:

$$L(\theta) = \underbrace{\sum_{(x_q, x_p)} L_p(x_q, x_p)}_{\text{Penalty for similar images that are far away}} + \underbrace{\sum_{(x_q, x_n)} L_n(x_q, x_n)}_{\text{Penalty for dissimilar images that are nearby}} \tag{1}$$

$$L_p(x_q, x_p) = ||x_q - x_p||_2^2 \tag{2}$$

$$L_n(x_q, x_n) = \max\left(0, m^2 - ||x_q - x_n||_2^2\right) \tag{3}$$

The loss consists of two penalties: L_p penalizes a positive pair (x_q, x_p) that is too far apart, and L_n penalizes a negative pair (x_q, x_n) that is closer than a margin m. If a negative pair is already separated by m, then there is no penalty for that pair and $L_n(x_q, x_n) = 0$.

To improve the embedding, we adjust θ to minimize the loss function L, which can be done with stochastic gradient descent with momentum [Krizhevsky et al. 2012] as follows:

$$v^{(t+1)} \leftarrow \mu \cdot v^{(t)} - \alpha \cdot \frac{\partial L}{\partial \theta}\left(\theta^{(t)}\right) \tag{4}$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + v^{(t+1)} \tag{5}$$

where $\mu \in [0, 1)$ is the momentum and $\alpha \in [0, \infty)$ is the learning rate. We use mini-batch learning, where L is approximated by only considering a handful of examples each iteration.

The remaining challenge is computing L and $\frac{\partial L}{\partial \theta}$. Hadsell et al. [2006] showed that an efficient method of computing and minimizing L is to construct a *siamese network* which is two copies of the CNN that share the same parameters θ. An indicator variable y selects whether each input pair I_1, I_2 is a positive ($y = 1$) or negative ($y = 0$) example. This entire structure can now be viewed as a new bigger network that consumes inputs I_1, I_2, y, θ and outputs L. With this view, it is now straightforward to apply the backpropagation algorithm [Rumelhart et al. 1986] and efficiently compute the gradient $\frac{\partial L}{\partial \theta}$.

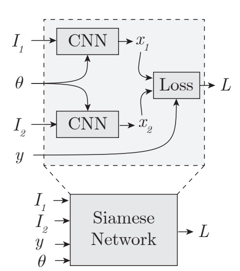


Figure 4: *Siamese network abstraction.*

For all of our experiments, we use Caffe [Jia et al. 2014], which contains efficient GPU implementations for training CNNs.

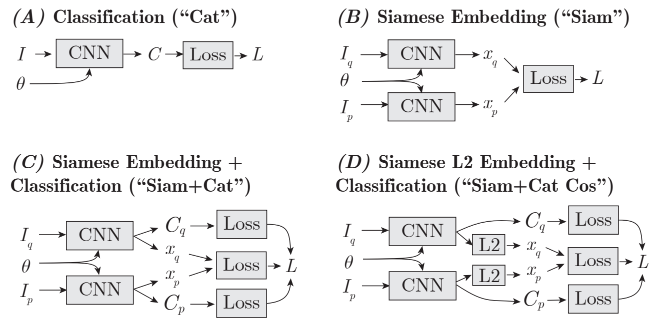


Figure 5: Training architectures. *We study the effect of several training architectures: (**A**) a CNN that is used for classification and then re-purposed as an embedding, (**B**) directly training an embedding, (**C**) also predicting the object categories C_q, C_p, and (**D**) also normalizing the embedding vectors to have unit L2 length (since Euclidean distance on normalized vectors is cosine distance). Loss for classification: softmax loss (softmax followed by cross-entropy); loss for embedding: contrastive loss.*

4 Our approach

For our work, we build a database of millions of products and scenes downloaded from Houzz.com. We focus on learning a single embedding that contains two types of images: in-situ examples of products I_q (cropped sub-images) and iconic product images I_p, as shown in Figure 3. Rather than hand-design the mapping $I_q \mapsto x_q$, we use a siamese network, described above, to automatically learn a mapping that is both high quality and fast to compute.

While siamese architectures [Hadsell et al. 2006] are quite effective, there are many ways to train the weights for a CNN. We explore different methods of training an embedding as shown in Figure 5. Some of these variations have been used before; for example, Razavian [2014b] used architecture **A** for instance retrieval, Chopra [2005] used **B** for identity verification, Wang [2014] used a triplet version of **B** with L2 normalization for fine-grained visual similarity, and Weston [2008] used **C** for MNIST digits and semantic role labeling. We evaluate these architectures on a real-world internet dataset.

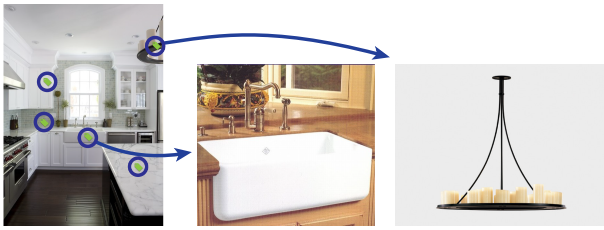
To train our CNN, we explore a new source of training data. We collect a hundred thousand examples of matching in-situ products and their iconic images. We use MTurk to collect the necessary in-situ bounding boxes for each product (Section 5). We then train multiple CNN architectures using this data and compare different multitask loss functions and different distance metrics. Finally, we demonstrate how the learned mapping can be used in visual search applications both across all object categories and within specific categories (Section 6). We perform searches in two directions, returning either product images or in-situ products in scenes.

5 Learning our visual similarity metric

In this section, we describe how we build a new dataset of positive and negative examples for training, how we use crowdsourcing to label the extent of each object, and how we train the network. Finally, we visualize the resulting embedding and demonstrate that we have learned a powerful visual similarity metric.

5.1 Collecting Training Data

Training the networks requires positive and negative examples of matching in-situ images and iconic product images. A great resource



(a) Full scene *(b)* Iconic product images

Figure 6: *Example product tags from in-situ objects to their products (Houzz.com), highlighted with blue circles. Two of the five tags contain iconic photos of the product. Photo credit: Fiorella Design.*

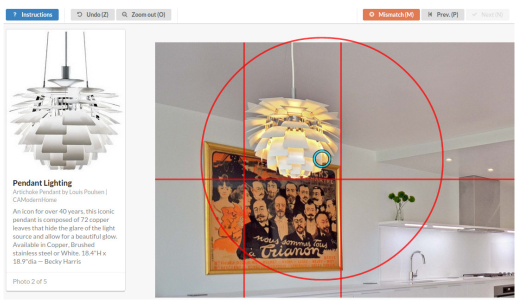


Figure 7: MTurk interface. *A video of the interface and instructions are included in the supplemental. Photo credit: Austin Rooke.*

for such images exist at websites like Houzz.com; the site contains millions of photos of both rooms and products. Many of the rooms contain product tags, where a "pro" user has annotated a product inside a room. The annotation can include a description, another photo, and/or a link to where the product may be purchased. For example, Figure 6 shows an example photo with several tags; two of which contain links to photos.

Images. To build our database, we recursively browsed pages on Houzz.com, downloading each photo and its metadata. In total, we downloaded 7,249,913 product photos and 6,515,869 room photos. Most products have an object category that is generally correct, which we later show can be used to improve the training process. Before we can use the data, we must detect duplicate and near-duplicate images; many images are uploaded thousands of times, sometimes with small variations. To detect both near- and exact-duplicates, we pass the images through AlexNet [Krizhevsky et al. 2012] and extract the output from the 7th layer (`fc7`), which has been shown to be a high quality image descriptor [Babenko et al. 2014]. We cluster any two images that have nearly identical descriptors, and then keep only one copy (the one with the most metadata). As a result, we retained 3,387,555 product and 6,093,452 room photos.

Product tags. Out of the 3,387,555 product photos, 178,712 have "product tags", where a "pro" user has marked an object in-situ with its corresponding iconic product photo. This gives us a single point, but we want to know the spatial extent of the product in the room. Further, many of these tags are incorrect or the result of spam. We use crowdsourcing to (a) provide a tight bounding box around each tagged in-situ object and (b) clean up invalid tags.

5.1.1 Crowdsourcing object extents

We chose to have workers draw bounding boxes instead of full polygon segments, as in Microsoft COCO [Lin et al. 2014] or OpenSurfaces [Bell et al. 2013], since bounding boxes are cheaper to acquire and we want training data of the same form as our final user input (bounding boxes).

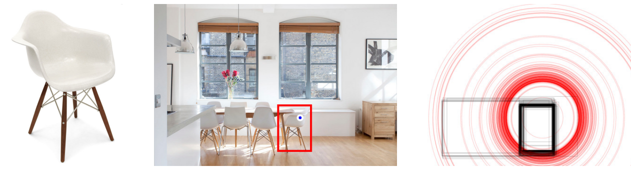


Figure 8: Example sentinel. *Left: product image shown to workers. Center: ground truth box (red) and product tag location (blue circle). Right: 172 worker responses with bounding box (black) and circle (red), rendered with 20% opacity. Photo credit: Increation Interiors.*

We designed a single MTurk task where workers are shown an iconic view of a product on the left, and the product in-situ on the right (see Figure 7). We show the location of the tag as an animated blue circle, to focus the worker's attention on the correct object (e.g., if there are multiple copies). See the supplemental for a sample video of the user interface. We then ask the worker to either draw a tight bounding box around the object, or flag the pair as a mismatched item (to eliminate spam). Note that workers are instructed to only flag very obvious mismatches and allow small variations on the products. Thus, the ground truth tags often have a different color or material but are otherwise nearly the same product.

The size of the object in-situ in the image can vary considerably depending on the object, the viewpoint, etc. We collect the extent of each object in-situ by asking the worker to click *five* times: (1) workers first click to set a bounding circle around the object, (2) then workers click to place a vertical line on the left/right edge of the object, (3-5) workers place the remaining bounding lines on the right/left, top, bottom. The initial bounding circle lets us quickly adjust the zoom level to increase accuracy. This step is critical to let us handle the very large variation in size of the objects while giving the workers the visual detail they need to see to provide good input. We also found in testing that using infinite lines, instead of boxes, makes it easier to draw a box around oddly shaped items.

In addition, we added "undo" and "go back" buttons to allow workers to fix mistakes. At the end of the task, workers are shown all of their bounding boxes on a new page, and can click to redo any item. These features were used by 60.4% of the workers. Worker feedback for our interface was unanimously positive.

5.1.2 Quality control

When crowdsourcing on Mechanical Turk, it is crucial to implement rigorous quality control measures, to avoid sloppy work. Some workers intentionally submit random results, others do not read the instructions. We ensure quality results with two methods: sentinels and duplication [Gingold et al. 2012]. Sentinels ensure that bad workers are quickly blocked, and duplication ensures that small mistakes by the remaining good workers are caught.

Sentinels. Sentinels are secret test items randomly mixed into each task. Users must agree with the ground truth by having an intersection-over-union (IOU) score of at least 0.7, $\mathrm{IOU}(A, B) = \frac{|A \cap B|}{|A \cup B|}$. If users make at least n mistakes and have an accuracy less than $n \cdot 10\%$, $3 \leq n \leq 8$, we prevent the user from submitting. Thus, a worker who submits 3 incorrect answers in a row will be blocked immediately, but we will wait longer before blocking a borderline worker. We order the sentinels so that the most difficult ones are presented first, so bad workers will be blocked after submitting just a few tasks. In total, we have 248 sentinels, and 6 of them are listed in the instructions with the correct answer. Despite giving away some of the answers, 11.9% of workers were blocked by our sentinels.

Figure 8 shows the variety of responses obtained for a single sentinel. Note that this example appears slightly ambiguous, since there are multiple copies of the chair that the worker could annotate. However, we have asked the worker to label only the one with the blue dot (which is animated in the task to make it more salient). It is important that we get all workers to follow a single convention so that we can use worker agreement as a measure of success.

Duplication. Even if a worker is reliable, they may be given a difficult example, or they may make occasional small mistakes. Therefore, we collect two copies of every bounding box and check whether the workers agree. If the intersection-over-union (IOU) of the two boxes is above 0.7, then we choose the box from the worker with the higher average sentinel score. If the workers disagree, we collect more boxes (up to 5) until we find a pair of responses that agrees (IOU ≥ 0.7).

Results With 1,742 workers, we collected 449,107 total responses which were aggregated to 101,945 final bounding boxes for an average cost of $0.0251 per final box. An additional 2,429 tags were "mismatched", i.e., at least half of the workers labeled *mismatch*. Workers were paid $0.05 to label 7 boxes (1 of which is a sentinel) and spent an average of 17.1 seconds per response.

5.2 Learning a distance metric

From crowdsourcing, we have collected a dataset of positive examples—the same product in-situ and in an iconic image. We now describe how we convert this data into a full dataset, how we train the network, and finally visualize the resulting embedding.

5.2.1 Generating positive and negative training data

The contrastive loss function consists of two types of examples: positive examples of similar pairs and negative examples of dissimilar pairs. Figure 9 shows how the contrastive loss works for positive and negative examples respectively for our domain. The gradient of the loss function acts like a force (shown as a red arrow) that pulls together x_p and x_q and pushes apart x_q and x_n.

We have 101,945 pairs of the form: (in-situ bounding box, iconic product image). This forms all of our positive examples (I_q, I_p). To build negative examples, we take each in-situ image I_q and pick 80 random product images of the same object category, and 20 random product images of a different category. We repeat the positive pair 5 times, to give a 1:20 positive to negative ratio. We also augment the dataset by re-cropping the in-situ images with different amounts of padding: $\{0, 8, 16, 32, 48, 64, 80\}$ pixels, measured with respect to a fixed 256x256 square input shape (for example, 16 pixels of padding means that 1/8$^{\text{th}}$ of each dimension is scene context).

We split all of these examples into training, validation, and test sets, making sure to separate bounding boxes by photo to avoid any contamination. This results in 63,820,250 training pairs and 3,667,769 validation pairs. We hold out 6,391 photos for testing which gives 10,000 unseen test bounding boxes. After splitting the examples, we randomly shuffle the pairs within each set.

5.2.2 Training the network

Various parameter choices have to be made when training the CNN.

Selecting the training architecture. As shown earlier in Figure 5, we explore multiple methods of training the CNN: (**A**) training on only category labels (no product tags), (**B**) training on only product tags (no category labels), (**C**) training on both product tags and category labels, and (**D**) additionally normalizing the embedding

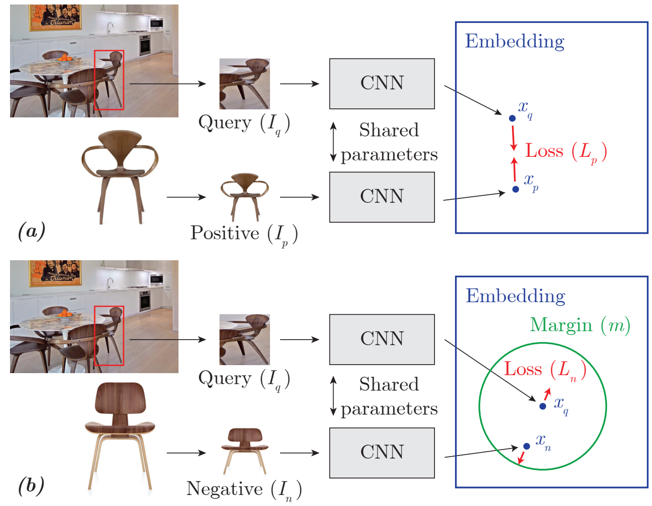


Figure 9: Training procedure. *In each mini-batch, we include a mix of **(a)** positive and **(b)** negative examples. In each iteration, we take a step to decrease the loss function; this can be viewed as "forces" on the points (red arrows). Photo credit: Austin Rooke.*

to unit L2 length. In the supplemental we detail the different training parameters for each architecture (learning rate, momentum, etc.).

Selecting the margin. The margin m, in the contrastive loss function, can be chosen arbitrarily, since the CNN can learn to globally scale the embedding proportional to m. The only important aspect is the relative scale of the margin and the embedding, at the time of initialization. Making the margin too large can make the problem unstable and diverge; making it too small can make learning too slow. Therefore, we try a few different margins $m \in \{1, \sqrt{10}, \sqrt{100}, \sqrt{1000}\}$ and select the one that performs best. When using L2 normalization (Figure 5(d)), we use $m = \sqrt{0.2}$.

Initializing the network parameters. We are training with stochastic gradient descent (SGD); the choice of initialization can have a large impact on both the time taken to minimize the objective, and on the quality of the final optimum. It has been shown that for many problem domains [Razavian et al. 2014a], transfer learning (training the network to a different task prior to the final task) can greatly improve performance. Thus, we use networks that were trained on a large-scale object recognition benchmark (ILSVRC2012), and use the learned weights hosted on the BVLC Caffe website [Jia et al. 2014] to initialize our networks.

To convert the network used for object classification to one for embedding, we remove the "softmax" operation at the end of the network and replace the last layer with an inner product layer with a D-dimensional output. Since the network was pre-trained on warped square images, we similarly warp our queries. We try multiple dimensions $D \in \{256, 1024, 4096\}$. Later we describe how we quantitatively evaluate performance and select the best network.

5.2.3 Visualizing the embedding

After the network has converged, we visualize the result by projecting our D-dimensional embedding down to two dimensions using the t-SNE algorithm [Van Der Maaten and Hinton 2008]. As shown in Figure 10, we visualize our embedding (trained with architecture **D**) by pasting each photo in its assigned 2D location.

When visualizing all products on the same 2D plane, we see that

Figure 10: 2D embedding visualization using t-SNE [Van Der Maaten and Hinton 2008]. *This embedding was trained using architecture* ***D*** *and is 256D before being non-linearly projected to 2D. To reduce visual clutter, each photo is snapped to a grid (overlapping photos are selected arbitrarily). Full embeddings are in the supplemental, including those for a single object category. Best viewed on a monitor.*

they are generally organized by object category. Further, when considering the portion of the plane occupied by a specific category, the products appear to be generally organized by some notion of visual style, even though the network was not trained to model the concept of "style". The supplementary material includes more visualizations of embeddings including embeddings computed for individual object classes. These are best viewed on a monitor.

6 Results and applications

We now qualitatively and quantitatively evaluate the results. We demonstrate visual search in three applications: finding similar products, in-situ usages of a product, and visually similar products across different object categories.

6.1 Visual search

Product search. The main use of our projection $I_q \mapsto x_q$ is to look up visually similar images. Figure 11 shows several example queries randomly sampled from the test set. The results are not curated and are truly representative of a random selection of our output results. The query object may be significantly occluded, rotated, scaled, or deformed; or, the product image may be a schematic representation rather than an actual photo; or, the in-situ image is of a glass or transparent object, and therefore visually very different from the iconic product image. Nonetheless, we find that our descriptor generally performs very well. In fact, in some cases our results are closer than the tagged iconic product image because the ground truth often shows related items of different colors. Usually when the search fails, it tends to still return items that are similar in some way. In the supplemental we show 500 more random uncurated examples of searches returned from our test set.

In-situ search. Since we have a descriptor that can model both iconic and in-situ cropped images, we can search in the "reverse" direction, where the query is a product and the returned results are cropped boxes in scenes. We use the bounding boxes we already collected from crowdsourcing as the database to search over. Figure 15 shows random examples sampled from the test set. Notice that since the scenes being searched over are designer photos, this becomes a powerful technique for exploring design ideas. For example, we could discover which tables go well with a chair by finding scenes that contain the chair and then looking for tables in the scenes.

Cross-category search. When retrieving very large sets of items for an input query, we found that when items show up from a different object category, these items tend to be visually or stylistically similar to the input query in some way. Despite not training the descriptor to reason about visual style, the descriptor is powerful enough to place stylistically similar items nearby in the space. We can explore this behavior further by explicitly searching only products of a *different* object category than the query (e.g. finding a table that is visually similar to a chair). Note that before we can do these sorts of queries, it is necessary to clean up the category labels on the dataset, since miscategorized items will show up in every search. Therefore, we use the "GN Cat" CNN (architecture **A**) [Szegedy et al. 2015] to predict a category for every product, and we remove all labels that either aren't in the top-20 or have confidence $\leq 1\%$. Figure 14 shows example queries across a variety of object categories. These results are curated, since most queries do not have a

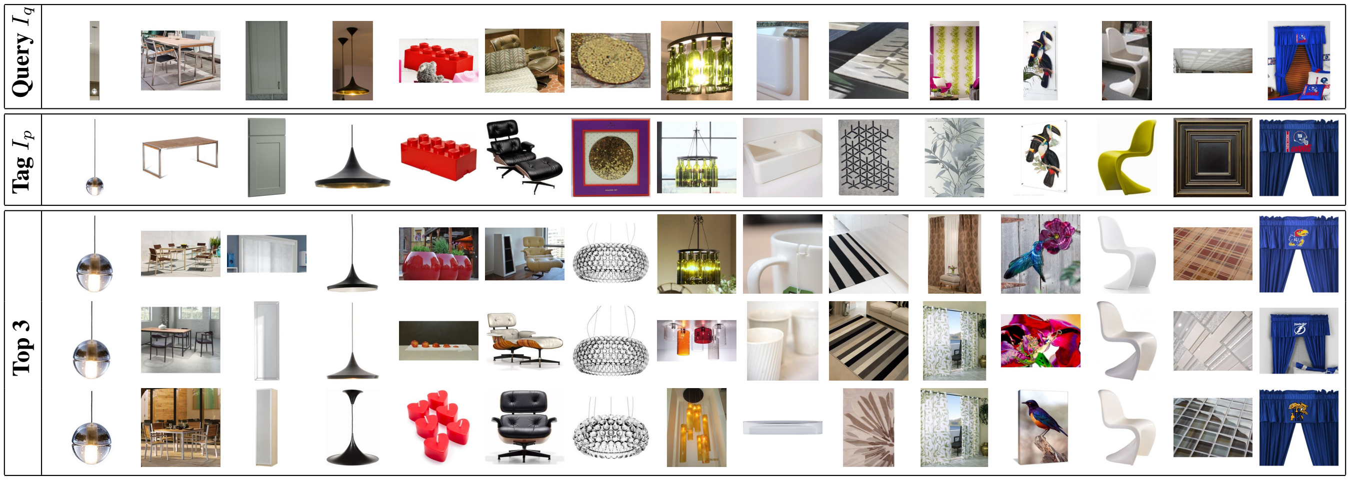


Figure 11: Product search: uncurated random queries from the test set. *For each query I_q, we show the top 3 retrievals using our method as well as the tagged canonical image I_p from Houzz.com. Object categories are not known at test time. Note that sometimes the retrieved results are closer to the query than I_p.*

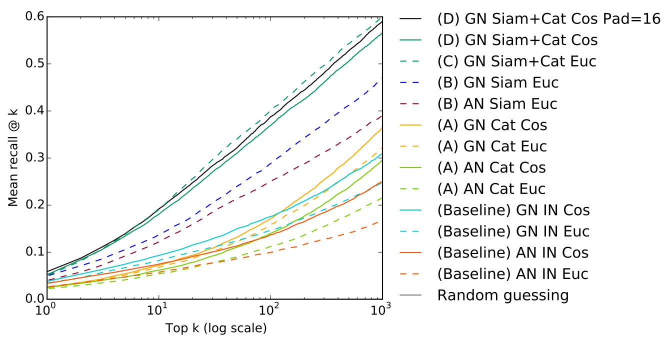


Figure 12: Quantitative evaluation (log scale). *Recall (whether or not the single tagged item was returned) as a function of the number of items returned (k). Recall for each query is either 0 or 1, and is averaged across 10,000 items. "GN": GoogLeNet, "AN": AlexNet, "Euc": Euclidean distance, "Cos" cosine distance, "Siam": trained with a Siamese network, "Cat": trained with object categories, "IN": ImageNet weights (not trained at all), "P=16": the box is expanded so that after warping to a square, there are 16 pixels of padding. Random guessing is flat along the bottom.*

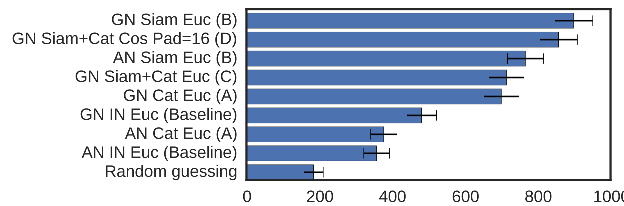


Figure 13: User study for cross-category searches. *Number of times that a user clicked on the prediction from a given algorithm. Error bars: 95% confidence interval (bootstrap sampling). Naming conventions are explained in Figure 12.*

distinctive style to them, and thus it is not apparent that anything has been matched. In the next section, we show a user study that evaluates to what extent our CNNs return stylistically similar items.

The product search and "in-situ" search are both trained with architecture **D**, and "cross-category" search uses architecture **B** (Figure 5). In the next section we detail how we quantitatively evaluated each architecture and selected the best.

6.2 Evaluating the metric

For our dataset, we cannot measure precision, since we only have a list of image positive pairs (in-situ image I_q and iconic product image I_p). However, we can measure recall of the product image—that is, we look at the closest k products to the query I_q and measure whether or not the tagged product I_p appears in the top k results. For each query, recall will be either 0 or 1; we average this over our test set of 10,000 pairs. We plot mean recall for each k in Figure 12.

It is important to note that image I_p is usually *not* the best match for I_q, as there is significant redundancy in the product images, even after accounting for near-duplicates. Often I_q and I_p differ in materials or in color. Nonetheless, a visual similarity metric should be able to deal with these issues, and place x_q and x_p reasonably close together in the embedding (so that its rank is small). So while a lower rank is better, a rank of 1 is not a reasonable expectation.

Baseline. To evaluate our embedding, we compare how well it can rank products compared to two high quality image descriptors for two different CNNs when trained on ImageNet ("IN"). For each CNN, we use the output from the last hidden layer and call them "AN IN" (i.e., AlexNet layer `fc7`) and "GN IN" (i.e., GoogLeNet layer `pool5/7x7_s1`). These two architectures are popular and have become the basis of a wide variety of state-of-the-art algorithms for image retrieval [Babenko et al. 2014; Razavian et al. 2014b]. We tried applying PCA since it has been shown that it can improve performance [Razavian et al. 2014b], but we found that it performed the same as the original descriptors, and thus is not shown.

Distance metrics. We compare two versions of each descriptor: a Euclidean version ("Euc") and a L2 normalized version ("Cos", since cosine distance is equal to Euclidean distance on normalized vectors). We evaluated other metrics including L1, Canberra, and Bray-Curtis dissimilarity on the baseline networks. We found that other metrics performed either comparably or worse than Cosine, and thus we only evaluate Cosine for the full dataset. For example, in Figure 12, "GN Cat Cos" means that we train GoogLeNet with object categories, and measure cosine distance using the last layer.

Padding. At test time, we experimented with adding different amounts of padding to the input query. We find that a modest amount of padding, 16 pixels, is optimal. Since all algorithms benefit from padding, we only show the effect on the best algorithm.

Figure 14: Style search: example cross-category queries. *For each object category ("armchairs", "rugs", etc.), we show the closest product of that category to the input I_q, even though the input (usually) does not belong to that category. We show this for 12 different object categories. Note that object category is not used to compute the descriptor x_q; it is only used at retrieval time to filter the set of products.*

The supplemental includes our full padding evaluation.

Training architectures. As shown in Figure 12 the best architecture for image retrieval is **D**, which is a siamese GoogLeNet CNN trained on both image pairs and object category labels, with cosine as the distance metric. For all experiments, we find that GoogLeNet (GN) consistently outperforms AlexNet (AN), and thus we only consider variations of **C** and **D** using GoogLeNet. Architecture **A** is the commonly used technique of fine-tuning a CNN to the target dataset. It performs better than the ImageNet baselines, but not as well as any of the siamese architectures **B**, **C**, **D**. It might appear that **C** is the best curve (which is missing L2 normalization), but we emphasize that we show up to $k = 1000$ only for completeness and the top few results $k < 10$ matter the most (where **D** is the best).

Embedding dimension. We studied the effect of dimension on architecture **C**. Using more dimensions makes it easier to satisfy constraints, but also significantly increases the amount of space and time required to search. We find that 256, 1024, and 4096 all perform about the same, and thus use 256 dimensions for all experiments. See the supplemental for the full curves.

Runtime. One of the key benefits of using CNNs is that computing $I_q \mapsto x_q$ is very efficient. Using a Grid K520 GPU (on an Amazon EC2 g2.2xlarge instance), we can compute x_q in about 100 ms, with most of the time spent loading the image. Even with brute force lookup (implemented as dense matrix operations), we can rank x_q against all 3,387,555 products in about one second on a single CPU. This can be accelerated using approximate nearest neighbor techniques [Muja and Lowe 2014], and is left for future work.

User study. As described in Section 6.1, Figure 14 shows examples of cross-category searches. Since our siamese CNNs were trained on image pairs and/or object categories, it is unclear which (if any) should work as a style descriptor for this type of search. Therefore, we set up a user study where a user is shown an item of one category and 9 items of a second category (e.g. one chair in context, and 9 table product images) and instructed to choose the best match, stylistically. We run this experiment on MTurk, where each of the 9 items are generated by a different model (8 models, plus one random). Each grid of 9 items is shown to 5 users, and only grids where a majority agree on an answer are kept. The supplemental contains screenshots of our user study. The results are shown in Figure 13, and we can see that all methods outperform random guessing, siamese architectures perform the best, and not training at all (baseline) performs the worst.

6.3 Discussion, limitations, and future work

There are many avenues to further improve the quality of our embedding, and to generalize our results.

Multitask predictions. We found that traning for object category along with the embedding performs the best for product search. Since the embedding and the object prediction are related by fully connected layers, the two spaces are similar. Thus, we effectively expand our training set of 86,945 pairs with additional 3,387,555 product category labels. At test time, we currently don't use the predicted object category—it is only a regularizer to improve the embedding. However, we could use the object category and use it to prune the set of retrieved results, potentially improving precision.

Style similarity vs. image similiarity. While we have not trained the CNN to model visual style, it seems to have learned a visual similarity metric powerful enough to place stylistically similar items nearby in the space. But style similarity and visual similarity are not the same. Two objects being visually similar often implies that they are also stylistically similar, but the converse is not true. Thus, if our embedding were to be used by designers interested in searching across compatible styles, we would need to explicitly train for this behavior. In future work, we hope to collect new datasets of style relationships to explore this behavior.

Active learning. We have a total of 3,387,555 product photos and 6,093,452 scene photos, but only 101,945 known pairs (I_q, I_p). As a result, dissimilar pairs of images that are not part of our known relationships may be placed nearby in the space, but there is no way to train or test for this. A human-in-the-loop approach may be useful here, where workers filter out false positives while training progresses. We propose exploring this in the future.

7 Conclusion

We have presented a visual search algorithm to match in-situ images with iconic product images. We achieved this by using a crowdsourcing pipeline for generating training data, and training a multitask siamese CNN to compute a high quality embedding of product images across multiple image domains. We demonstrated the utility of this embedding on several visual search tasks: searching for products within a category, searching across categories, and searching for usages of a product in scenes. Many future avenues of research remain, including: training to understand visual style in products, and improved faceted query interfaces, among others.

Acknowledgements

This work was supported in part by Amazon AWS for Education, a NSERC PGS-D scholarship, the National Science Foundation (grants IIS-1149393, IIS-1011919, IIS-1161645), and the Intel Science and Technology Center for Visual Computing. We thank the Houzz users who gave us permission to reproduce their photos: Crisp Architects, Rob Karosis, Fiorella Design, Austin Rooke, Increation Interiors. Credits for thumbnails are in the supplemental.

References

BABENKO, A., SLESAREV, A., CHIGORIN, A., AND LEMPITSKY, V. S. 2014. Neural codes for image retrieval. In *ECCV*.

BELL, S., UPCHURCH, P., SNAVELY, N., AND BALA, K. 2013. OpenSurfaces: A richly annotated catalog of surface appearance. *ACM Trans. on Graphics (SIGGRAPH) 32*, 4.

CHATFIELD, K., SIMONYAN, K., VEDALDI, A., AND ZISSERMAN, A. 2014. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*.

CHECHIK, G., SHARMA, V., SHALIT, U., AND BENGIO, S. 2010. Large scale online learning of image similarity through ranking. *JMLR*.

CHOPRA, S., HADSELL, R., AND LECUN, Y. 2005. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, IEEE Press.

GARCES, E., AGARWALA, A., GUTIERREZ, D., AND HERTZMANN, A. 2014. A similarity measure for illustration style. *ACM Trans. Graph. 33*, 4 (July).

GINGOLD, Y., SHAMIR, A., AND COHEN-OR, D. 2012. Micro perceptual human computation. *TOG 31*, 5.

GIROD, B., CHANDRASEKHAR, V., CHEN, D. M., CHEUNG, N.-M., GRZESZCZUK, R., REZNIK, Y., TAKACS, G., TSAI, S. S., AND VEDANTHAM, R., 2011. Mobile visual search.

HADSELL, R., CHOPRA, S., AND LECUN, Y. 2006. Dimensionality reduction by learning an invariant mapping. In *CVPR*, IEEE Press.

JEGOU, H., PERRONNIN, F., DOUZE, M., SANCHEZ, J., PEREZ, P., AND SCHMID, C. 2012. Aggregating local image descriptors into compact codes. *PAMI 34*, 9.

JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*.

KARAYEV, S., TRENTACOSTE, M., HAN, H., AGARWALA, A., DARRELL, T., HERTZMANN, A., AND WINNEMOELLER, H. 2014. Recognizing image style. In *BMVC*.

KOVASHKA, A., PARIKH, D., AND GRAUMAN, K. 2012. Whittlesearch: Image search with relative attribute feedback. In *CVPR*.

KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

KULIS, B. 2012. Metric learning: A survey. *Foundations and Trends in Machine Learning 5*, 4.

LECUN, Y., BOSER, B., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W., AND JACKEL, L. D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4.

LIN, T., MAIRE, M., BELONGIE, S., HAYS, J., PERONA, P., RAMANAN, D., DOLLÁR, P., AND ZITNICK, C. L. 2014. Microsoft COCO: common objects in context. *ECCV*.

MUJA, M., AND LOWE, D. G. 2014. Scalable nearest neighbor algorithms for high dimensional data. *PAMI*.

O'DONOVAN, P., LĪBEKS, J., AGARWALA, A., AND HERTZMANN, A. 2014. Exploratory font selection using crowdsourced attributes. *ACM Trans. Graph. 33*, 4.

ORDONEZ, V., JAGADEESH, V., DI, W., BHARDWAJ, A., AND PIRAMUTHU, R. 2014. Furniture-geek: Understanding fine-grained furniture attributes from freely associated text and tags. In *WACV*, 317–324.

PARIKH, D., AND GRAUMAN, K. 2011. Relative attributes. In *ICCV*, 503–510.

PERRONNIN, F., AND DANCE, C. 2007. Fisher kernels on visual vocabularies for image categorization. In *CVPR*.

RAZAVIAN, A. S., AZIZPOUR, H., SULLIVAN, J., AND CARLSSON, S. 2014. CNN features off-the-shelf: an astounding baseline for recognition. *Deep Vision (CVPR Workshop)*.

RAZAVIAN, A. S., SULLIVAN, J., MAKI, A., AND CARLSSON, S. 2014. Visual instance retrieval with deep convolutional networks. *arXiv:1412.6574*.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. 1986. Learning internal representations by error-propagation. *Parallel Distributed Processing 1*.

SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. 2015. Going deeper with convolutions. *CVPR*.

TAIGMAN, Y., YANG, M., RANZATO, M. A., AND WOLF, L. 2014. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*.

VAN DER MAATEN, L., AND HINTON, G. 2008. Visualizing data using t-SNE. In *Journal of Machine Learning*.

WANG, J., SONG, Y., LEUNG, T., ROSENBERG, C., WANG, J., PHILBIN, J., CHEN, B., AND WU, Y. 2014. Learning fine-grained image similarity with deep ranking. In *CVPR*.

WESTON, J., RATLE, F., AND COLLOBERT, R. 2008. Deep learning via semi-supervised embedding. In *ICML*.

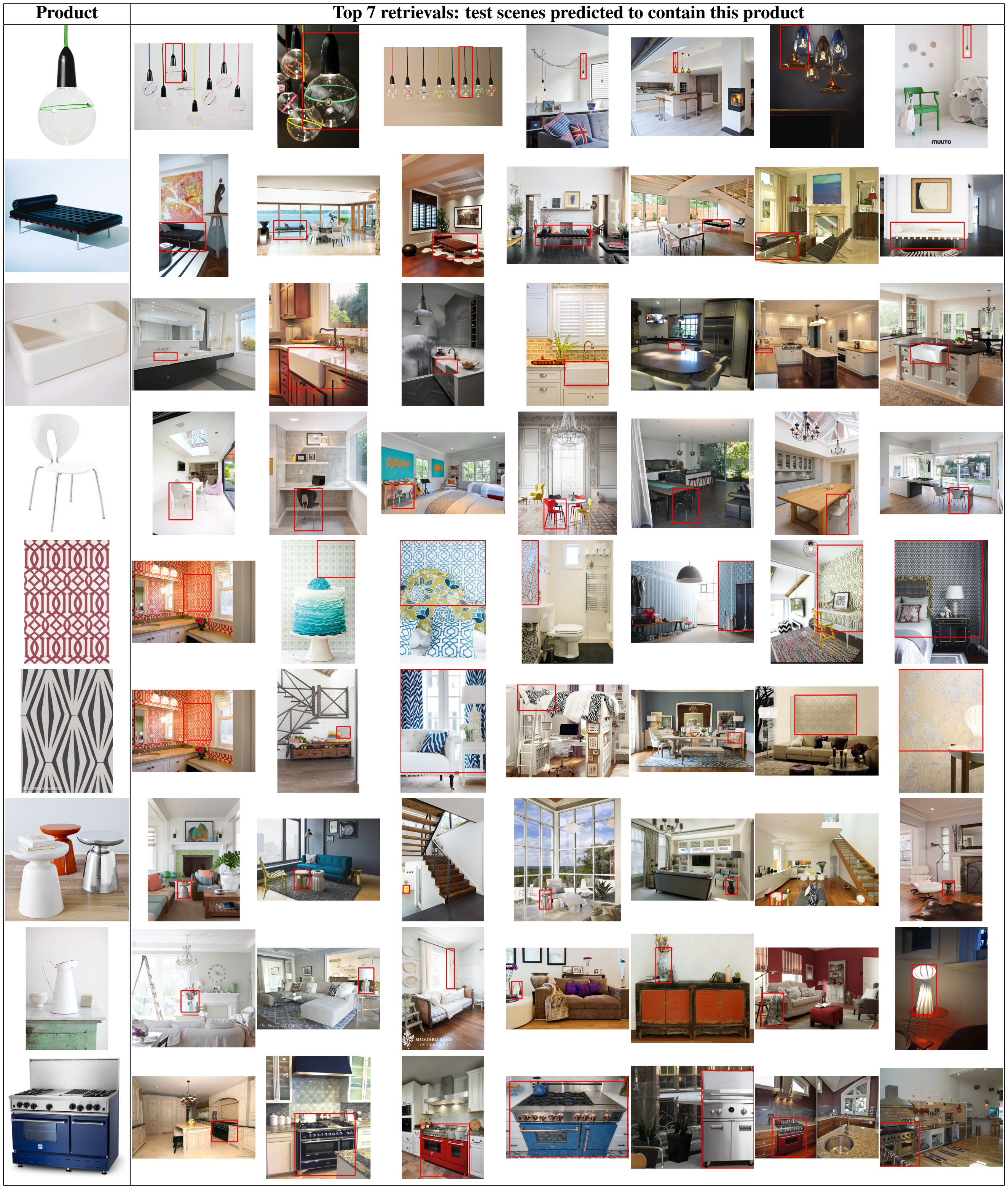


Figure 15: In-situ product search: random uncurated queries searching the test set. *Here, the query is a product, and the retrievals are examples of where that product was used in designer images on the web. The boxes were drawn by mturk workers, for use in testing product search; we are assuming that the localization problem is solved. Also note that the product images (queries) were seen during training, but the scenes being searched over were not (since they are from the test set). When sampling random rows to display, we only consider items that are tagged at least 7 times (since we are showing the top 7 retrievals). We show many more in the supplemental.*