Creating an image similarity function with TensorFlow and its application in e-commerce

Nina Pakhomova

Jun 29, 2017 • 6 min read

In our previous post, we showed you how to use image recognition to solve the issue of misattribution in e-commerce catalogs. Once you start to trust your models and have trained them to detect a valuable amount of attributes, it is easy to expand from attribute verification to auto tagging. However, our approach to misattribution is only very efficient when you already have a training set with a large set of images. The question remains, how can we take advantage of this technology when we only have a couple of examples in our library? In this post, we'll show you how to solve this issue by building an image similarity function, which can be used to build image search and fill attribution gaps.

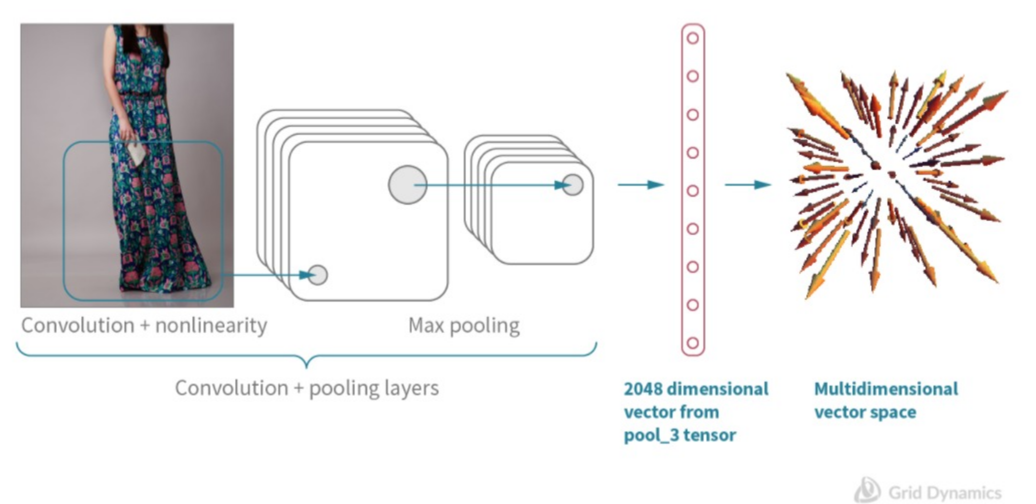
Image similarity: Filling attribution gaps with image recognition without a training set

As we know, our previous approach works well when you already have attributed data. The data set serves as a comparison for the new input from product images. Now, we need to figure out how to approach cases where the "Hawaiian" style attribute is needed, but there is no training set.

One approach would be to spend a couple of months creating an initial training set for your model. However, based on our experience, we posited that there has to be a way to get attributes from "similar images" using an untagged catalog. Taking this hypothesis as our starting point, we created an **image similarity** function. This turned out to be our first step to implementing image search within our e-commerce catalog.

With an image similarity function you can take a couple of examples that illustrate a new attribute. After which, you can search for images similar to the examples and assign the attribute to those images, as a baseline. As in the case of misattribution, we started with InceptionV3 Convolutional Neural Networks in TensorFlow. You may recall, Inception V3 is already trained and is able to recognize thousands of features. This time, instead of retraining the classification layer, we've taken vectors from the **pool_3** layer and started our research. The **pool_3** layer is the last pooling layer of this model, mentioned in the previous post.

We started by building a vector representation of every image in our catalog, which generated thousands of vectors. The result was a rich multi-dimensional vector space.

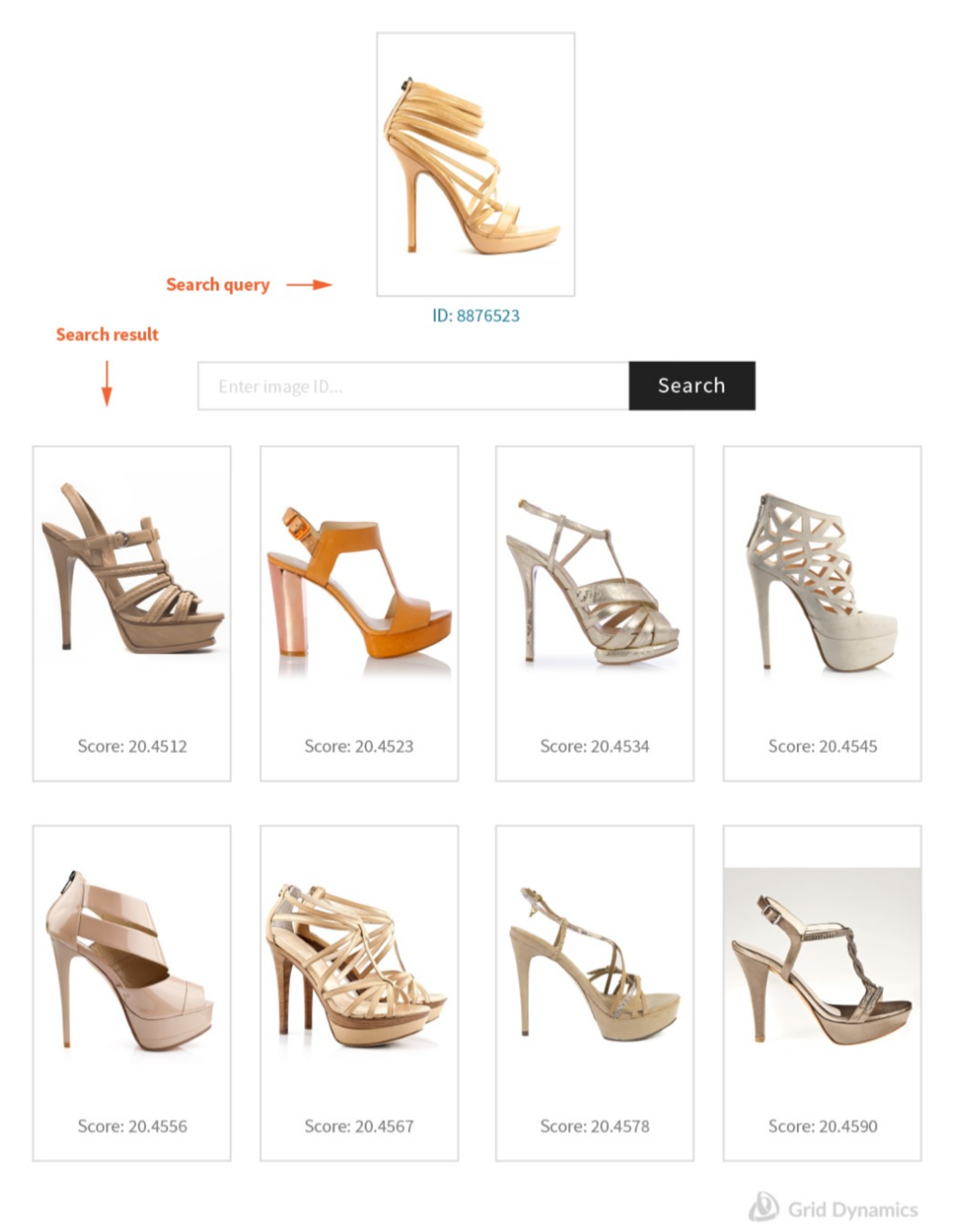


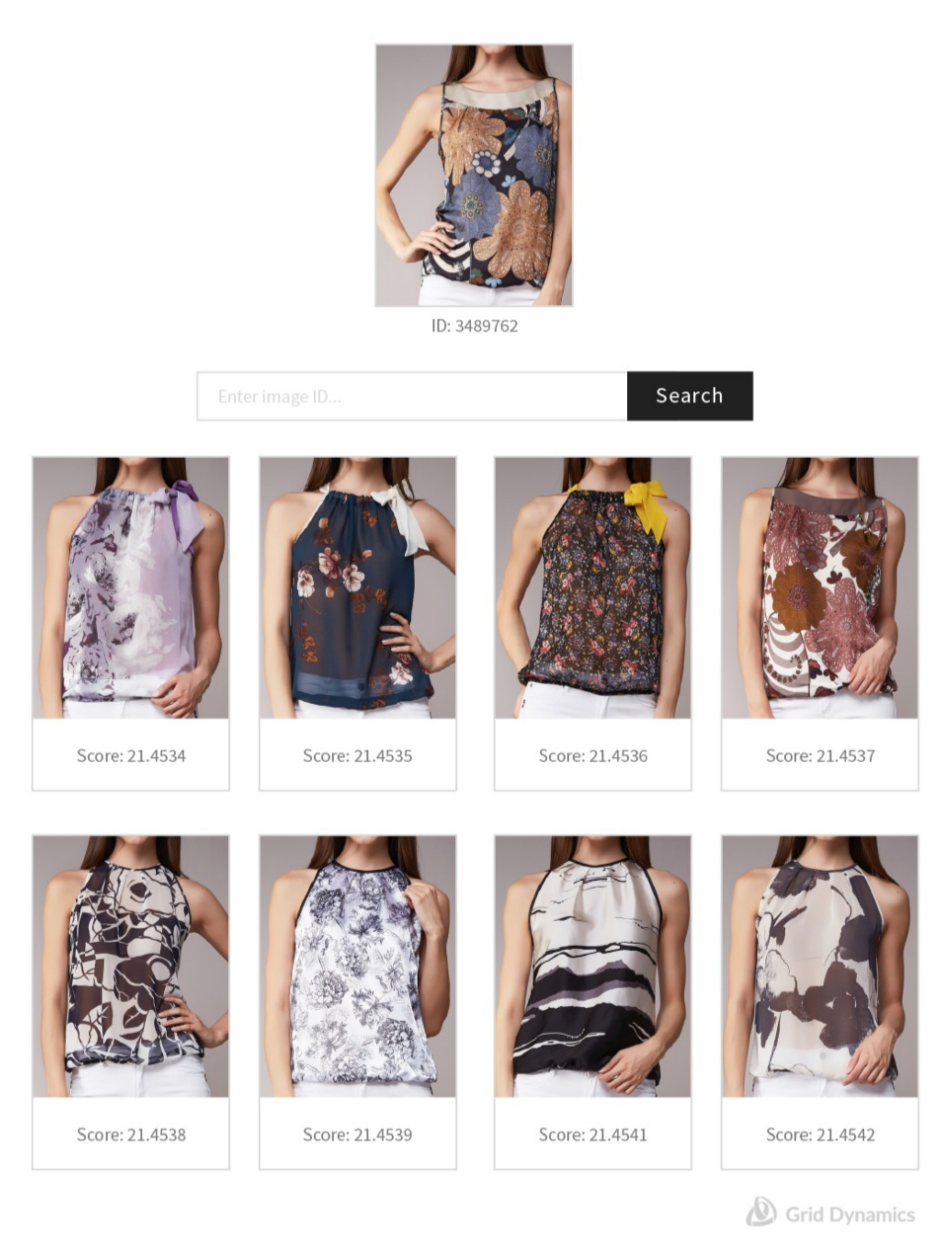
Upon analysis, we discovered that Euclidian squared distance works as similarity function for images within the multi-dimensional space. This means, the more similar two images are, the closer their corresponding coordinates manifest within the space. As a result, when you apply a distance function that spans across all items in a set, you can build a lot of different features, including:

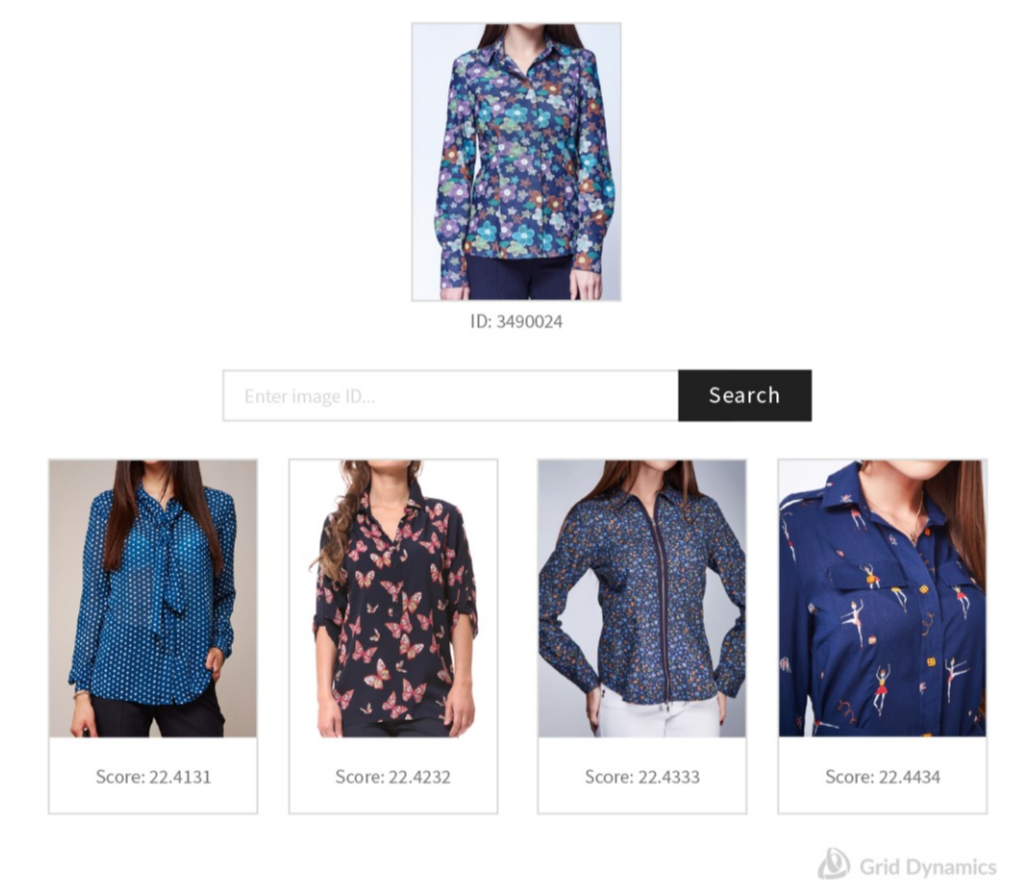
- **Clusterization algorithms**: Identify clusters of products/images that represent new attributes. A merchandiser could, for example, select the centroid of the cluster (the most representative point within the group) and assign a new attribute for the represented image. The attribute would then be automatically assigned to all images in the given cluster.
- **Image navigation:** Identify the starting point for your image gallery through which a customer can gradually navigate to the desired image/product.
- **Image search:** Retrieve the top number of corresponding images to a given image/product.
- **Visual filtering:** Build a custom image filter, based on a user's preferences.

Image search: Retrieve images similar to given image

After performing a market analysis, we decided to focus on image search as our preferred functionality. Image search can serve as the basis for a better search and navigation experience for users. Before getting into the technical details, let's start with a couple of examples. Below, you can see the images that were used as the query with the image search results presented in a table underneath.





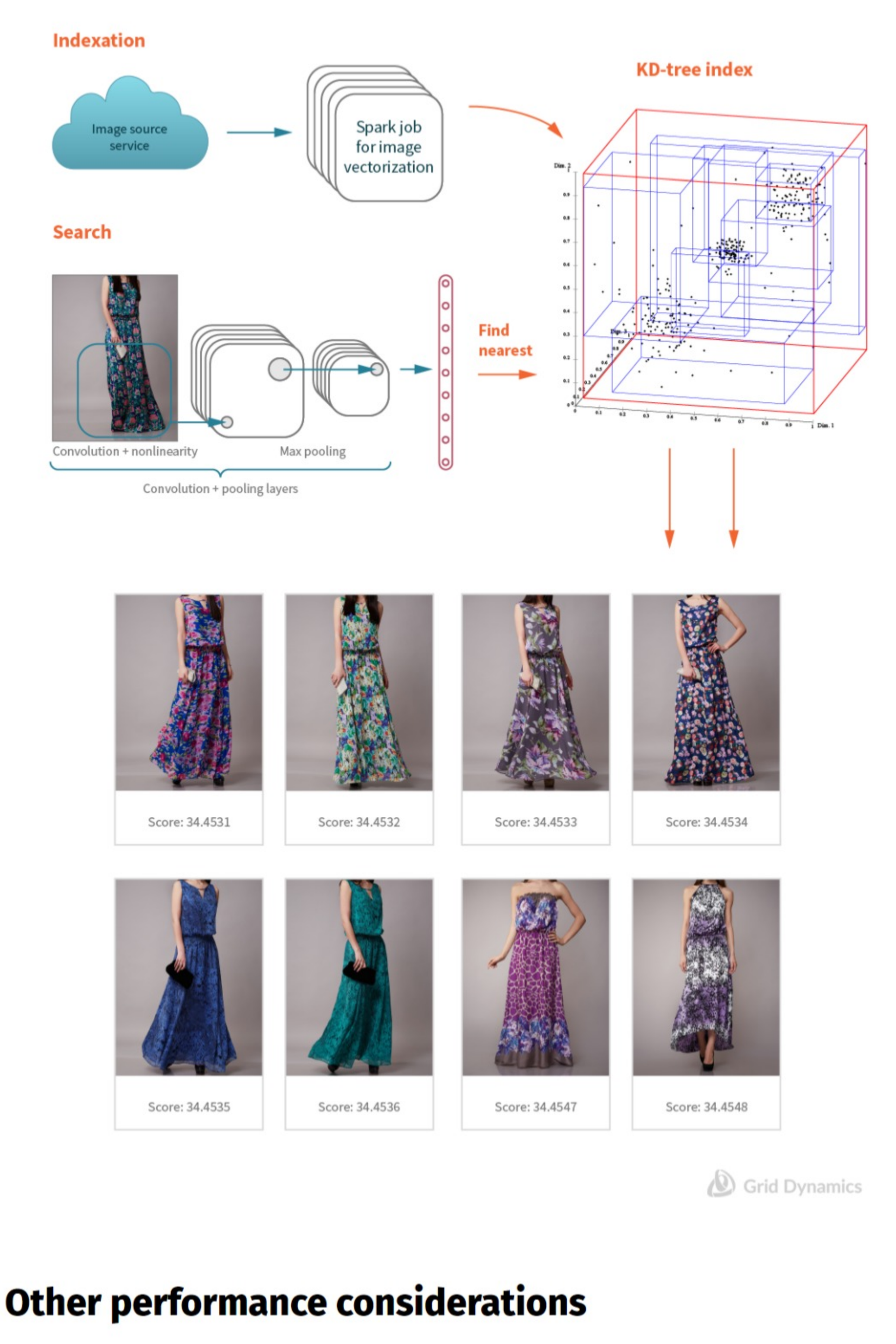


Our first version of image search returned good quality results; however, it had a number of performance issues. As such, we shifted our focus to performance improvement.

Performance improvements: Find the nearest vectors in multidimensional space

Considering our previous learning about distance functions in multi-dimensional space, the image search solution can now be rephrased as "find the nearest vectors in multidimensional space". Again, this is because the more similar two images are, the closer they appear. We built a K-dimensional TREE data structure to perform this operation. A K-dimensional TREE is a data structure for organizing a number of points in a space with K dimensions within a Euclidian plane. It is a binary search tree with other constraints imposed on it. K-d trees are very useful for range and nearest neighbour searches.

By implementing the K-d Tree, we saw a significant performance boost, with a 15x boost within a 500k image index. The K-d Tree data structure is very efficient, but it's complexity grows tremendously in proportion to vectors' dimensionality. In order to combat this issue, we decided to use Principal component analysis (PCA), to simplify vectors into their core components. PCA is a methodology for emphasizing variation and highlighting patterns in datasets. It is often used to simplify data for mining and visualization purposes. After testing the results post PCA, we found that decreasing a vector's dimensionality from 2048 to 1024 did not affect image search quality significantly.



Other performance considerations

Performance is closely tied to the success of implementing this kind of approach. The speed of search within e-commerce is critical to the consumer's user experience and subsequent conversion. For this reason, image search needs to be built in such a way as to optimize performance. Luckily, many sophisticated performance optimizations are possible within this approach, such as ball tree. A ball tree is a data structure that can be used for fast, high-dimensional nearest-neighbour searches. The scorer can be programmed to match every image, or alternatively it can be programmed to skip some images with low similarity based on a threshold or more advanced criteria.

The critical point is that the scoring mechanism is able to work on the same query in tandem with all other Lucene scoring mechanisms. This also makes it possible to combine visual search with filtration. Lucene has a number of tools to help with performance as well, such as its two-phase iterator API. It consists of an approximation phase that quickly iterates a superset of matching documents, and a verification phase that can be used to check if a document in this superset actually matches the query.

Imagine that you are searching for "knee-length dress" and applying a filter as well. The fact that we can dissociate the approximation from the verification phase allows us to intersect the approximation with the filter first, so that we will verify positions on a smaller set of attributes. This is very useful for our search use-case in general. However, a two-phase iteration pattern also applies to geo-distance queries, where it can use a distance computation as a verification, and only apply filters that can return all images in the index as an approximation and run image similarity as a verification.

For more information on building search with Solr Lucene, check out our Search blog series.

Conclusion

Using Machine Learning for image recognition is nascent technology with huge potential for business applications. In this series of blog posts, we've covered how to use ML to build an image search functionality within an e-commerce catalog and how it can be used to resolve misattribution and attribution gaps, leveraging Google's TensorFlow framework. In subsequent posts, we will cover how to integrate ML technology with other best-of-breed tools like Solr-based search.

Don't forget to subscribe to our blog to get the latest Open Source blueprints for Search, QA, Real-time Analytics, In-Stream Processing and more. If you liked this post, comment below.

Machine Learning and Artificial Intelligence | Image search | Tensorflow image similarity | Image similarity | Image similarity algorithm | Search