Similar Images Recommendations using FastAi and Annoy

Gautham Kumaran
Jul 15 · 5 min read

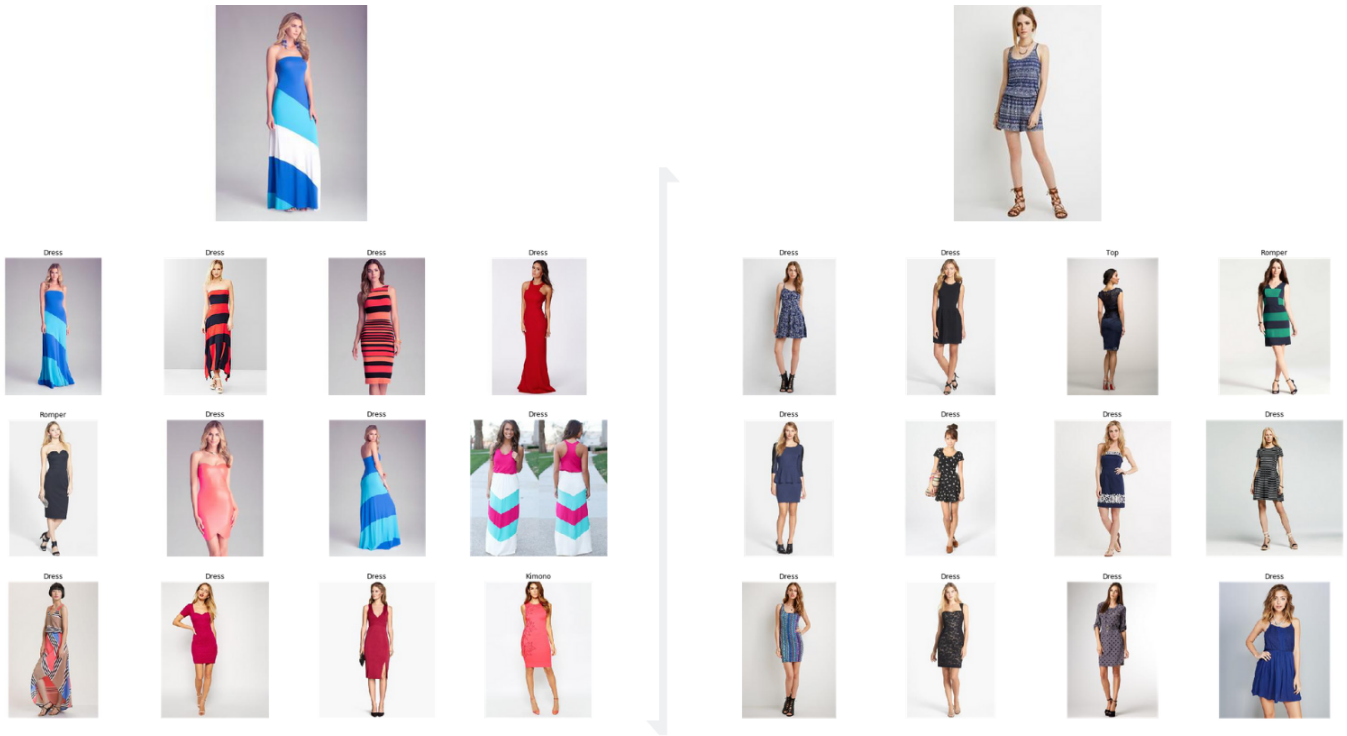


Image similarity-based recommendations, as shown above, is what we will be looking at, given a base image, recommend visually similar images. Image-based recommendations come in very handy in many scenarios especially in cases where visual tastes of the user are involved — Apparels, Jewellery, Accessories, etc. We will be using DeepFashion dataset which has **289,222 Apparel images** spread across **42 Categories**. Let's take a quick look at our toolbox for this project.

FastAi, a wrapper over PyTorch, which makes deep learning tasks astonishingly simple and comes with all best practices baked within it.

PyTorch, deep learning library by Facebook, we will be using a few features of PyTorch in our project.

Annoy, approximate nearest neighbor implementation open-sourced by Spotify, to index and search feature vectors efficiently.

The code for this project can be found in this jupyter notebook.

Embeddings or Feature Vectors

Embeddings or feature vectors can be thought of as a concise, n-dimensional vector form on an object that aims to capture most of the information contained in the object. In its n-dimensional feature space, similar objects are closer to each other when compared dissimilar objects.

For an image, the embeddings can be obtained from **Convolutional Neural Networks**(CNN). CNN is able to understand/learn the information contained in the image, at varying levels of details in each of its layers. The initial convolutional layers understand the low-level shape such as corners, edges, etc. and the end layers understand the complex structures in the image, faces, buildings, etc. based on the task. The end layers of a CNN are typically fully connected linear layers. The output of these fully connected layers is used as image embeddings.

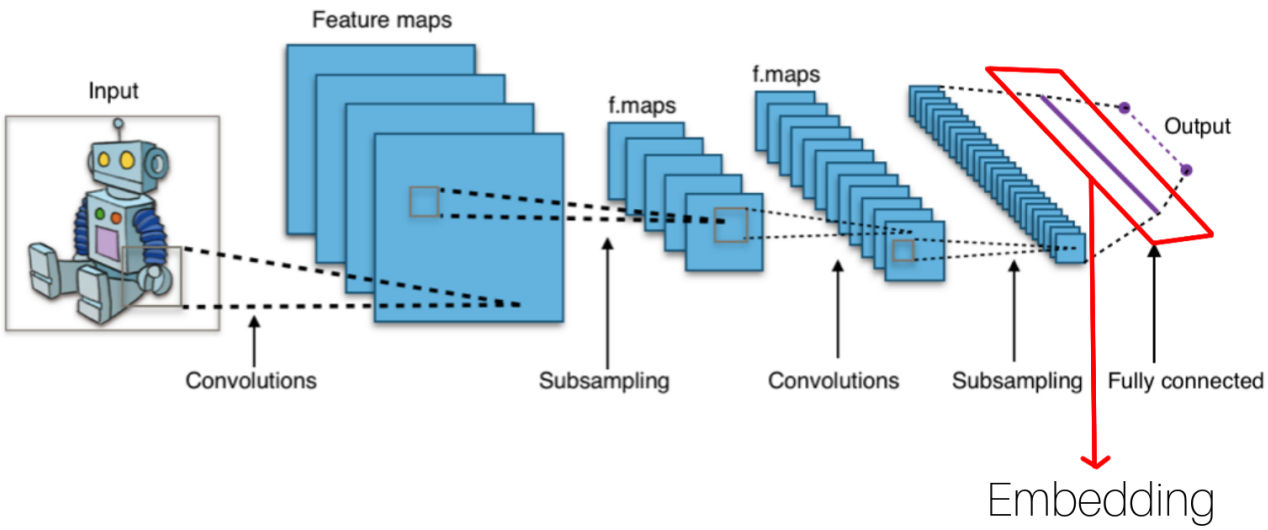


Image Embedding. CNN image source-en.wikipedia.org/wiki/Convolutional_neural_network

Fine-tune Pretrained Resnet152 model with FastAI

The CNN model that we are using is a pretrained **Resnet152** trained on Imagenet dataset for image classification. While this model can be used directly to obtain the embeddings, fine-tuning the model to the Deep Fashion dataset using FastAi will help the model understand the nuances of our dataset better. This process is called **Transfer learning**.

On training the model on DeepFashion category prediction task, it achieves a **top-3 accuracy of 88.4%** and **top-5 accuracy of 93.98%**, which is better than the benchmark scores published in the DeepFashion white paper[1].

epoch	train_loss	valid_loss	accuracy	top_k_accuracy	top_k_accuracy	time
0	1.538229	1.293813	0.628750	0.838200	0.907125	23:38
1	1.327644	1.198451	0.647325	0.853175	0.920200	23:35
2	1.305184	1.154403	0.660775	0.861350	0.925325	23:33
3	1.254822	1.128241	0.668025	0.866075	0.929050	23:32
4	1.218913	1.108451	0.676125	0.871200	0.931800	23:31
5	1.149093	1.072270	0.685550	0.876425	0.935100	23:30
6	1.153727	1.056559	0.690375	0.880525	0.937875	23:31
7	1.099121	1.046510	0.692975	0.881925	0.938575	23:30
8	1.095606	1.035576	0.697075	0.883850	0.939575	23:30
9	1.076328	1.035573	0.697000	0.884050	0.939825	23:31

Results of training resnet152 for 10 epochs

The feature vectors are extracted from this fine-tuned model.

Extract Feature vectors from the Model using PyTorch Hooks

From the resnet152 architecture, we use the output of the second last fully connected layer as the embedding, they are vectors of dimension (512, 1).
In order to get the output of an intermediate layer from a model in PyTorch, we use a functionality called Hook. Hooks can be added to extract intermediate values of a model from either the forward pass or the backward pass.

The best method of using PyTorch hooks can be adapted from the FastAi library code itself.

```
1  class Hook():
2      def __init__(self, m:nn.Module, hook_func:HookFunc, is_forward:bool=True, detach:bo
```

```
        self.hook_func,self.detach,self.stored = hook_func,detach,None
        f = m.register_forward_hook if is_forward else m.register_backward_hook
        self.hook = f(self.hook_fn)
        self.removed = False

    def hook_fn(self, module:nn.Module, input:Tensors, output:Tensors):
        if self.detach:
            input  = (o.detach() for o in input ) if is_listy(input ) else input.detach
            output = (o.detach() for o in output) if is_listy(output) else output.detac
        self.stored = self.hook_func(module, input, output)

    def remove(self):
        if not self.removed:
            self.hook.remove()
            self.removed=True

    def __enter__(self, *args): return self
    def __exit__(self, *args): self.remove()

def get_output(module, input_value, output):
    return output.flatten(1)

def get_input(module, input_value, output):
    return list(input_value)[0]

def get_named_module_from_model(model, name):
    for n, m in model.named_modules():
        if n == name:
            return m
    return None

linear_output_layer = get_named_module_from_model(model, '1.4')

# getting all images in train
train_valid_images_df = data_df[data_df['dataset'] != 'test']
inference_data_source = (ImageList.from_df(df=train_valid_images_df, path=images_path,
                    .split_none()
                    .label_from_df(cols='category')
                )

inference_data = inference_data_source.transform(tmfs, size=224).databunch(bs=64).norma

# turning off shuffle
inference_dataloader = inference_data.train_dl.new(shuffle=False)

import time
img_repr_map = {}
```

```
50
51  with Hook(linear_output_layer, get_output, True, True) as hook:
52      start = time.time()
53      for i, (xb, yb) in enumerate(inference_dataloader):
54          bs = xb.shape[0]
55          img_ids = inference_dataloader.items[i*bs: (i+1)*bs]
56          result = model.eval()(xb)
57          img_reprs = hook.stored.cpu().numpy()
58          img_reprs = img_reprs.reshape(bs, -1)
59          for img_id, img_repr in zip(img_ids, img_reprs):
60              img_repr_map[img_id] = img_repr
61          if(len(img_repr_map) % 12800 == 0):
62              end = time.time()
63              print(f'{end-start} secs for 12800 images')
64              start = end
65
66  img_repr_df = pd.DataFrame(img_repr_map.items(), columns=['img_id', 'img_repr'])
67  img_repr_df['label'] = [inference_data.classes[x] for x in inference_data.train_ds.y.it
```

pytorch_hooks.py hosted with ❤ by GitHub view raw

Using Pytorch hooks we generated the feature vectors for all the images in train and valid dataset.

Recommend Similar Images using Feature vectors

Now that we have the feature vector of all images, we will have to get similar images given a base image. Initially, trying a naive approach, given a base image lets compute its similarity score to every other image in the dataset and get the top-n images.

```
1  def get_similar_images(img_index, n=10):
2      start = time.time()
3      base_img_id, base_vector, base_label  = img_repr_df.iloc[img_index, [0, 1, 2]]
4      cosine_similarity = 1 - img_repr_df['img_repr'].apply(lambda x: cosine(x, base_vecto
5      similar_img_ids = np.argsort(cosine_similarity)[-11:-1][::-1]
6      end = time.time()
7      print(f'{end - start} secs')
8      return base_img_id, base_label, img_repr_df.iloc[similar_img_ids]
```

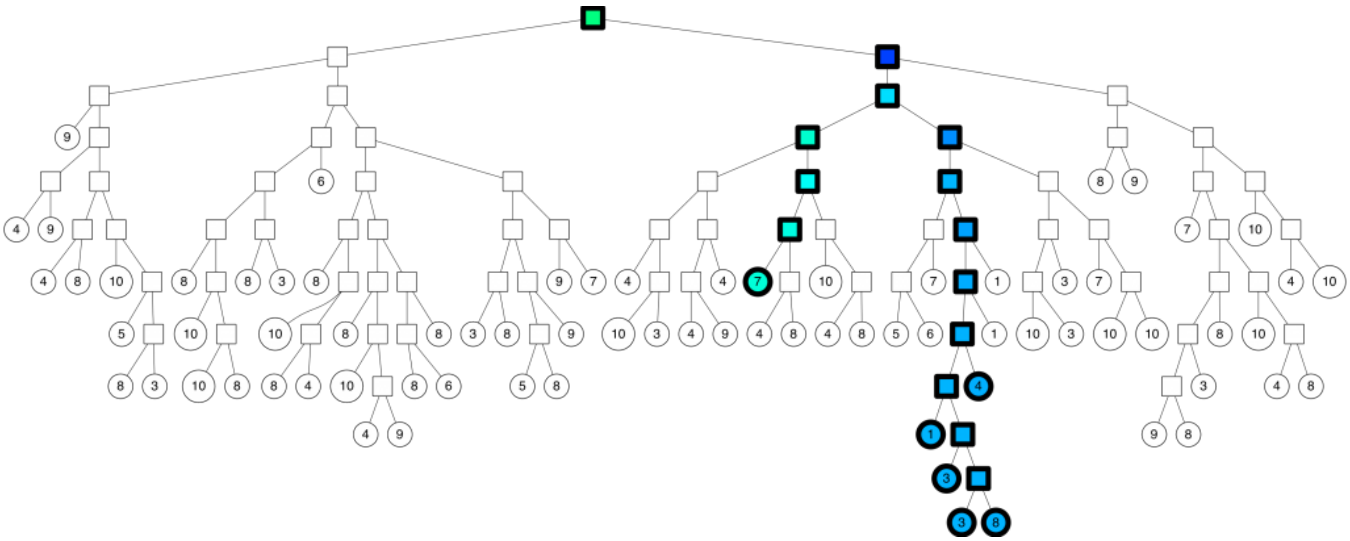
similar_images_naive.py hosted with ❤ by GitHub view raw

This approach does not scale, each query has an O(N) complexity and takes an approximate time of **10 seconds for 249,222 images**. To reduce this complexity to sub-linear time we will be using **Approximate Nearest Neighbor algorithms.** Approximate nearest neighbor algorithms can be seen as a trade-off between accuracy

and performance. The benchmark comparisons on various ANN algorithms is available in a blog post by Eric Bernhardsson, the author of Annoy.

Using Annoy to get similar Images

Annoy (**A**pproximate **N**earest **N**eighbor **O**h **Y**eah) is a binary tree-based implementation of ANN. The best explanation for it's working can found in this writeup by Eric.



Annoy — binary tree traversal for a query, source-erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html

The accuracy-performance tradeoff is controlled by the number of binary trees we build, more trees mean better accuracy. We will be using the same number of trees as the number of classes in our dataset. The below code is used to build the index.

```
from annoy import AnnoyIndex

feature_dim = len(img_repr_df['img_repr'][0])
t = AnnoyIndex(feature_dim, metric='euclidean')

for i, vector in enumerate(img_repr_df['img_repr']):
    t.add_item(i, vector)

_  = t.build(inference_data.c)
```

annoy_build_index.py hosted with ❤ by GitHub view raw

Now let's try querying the Annoy index.

```
def get_similar_images_annoy(img_index):
```

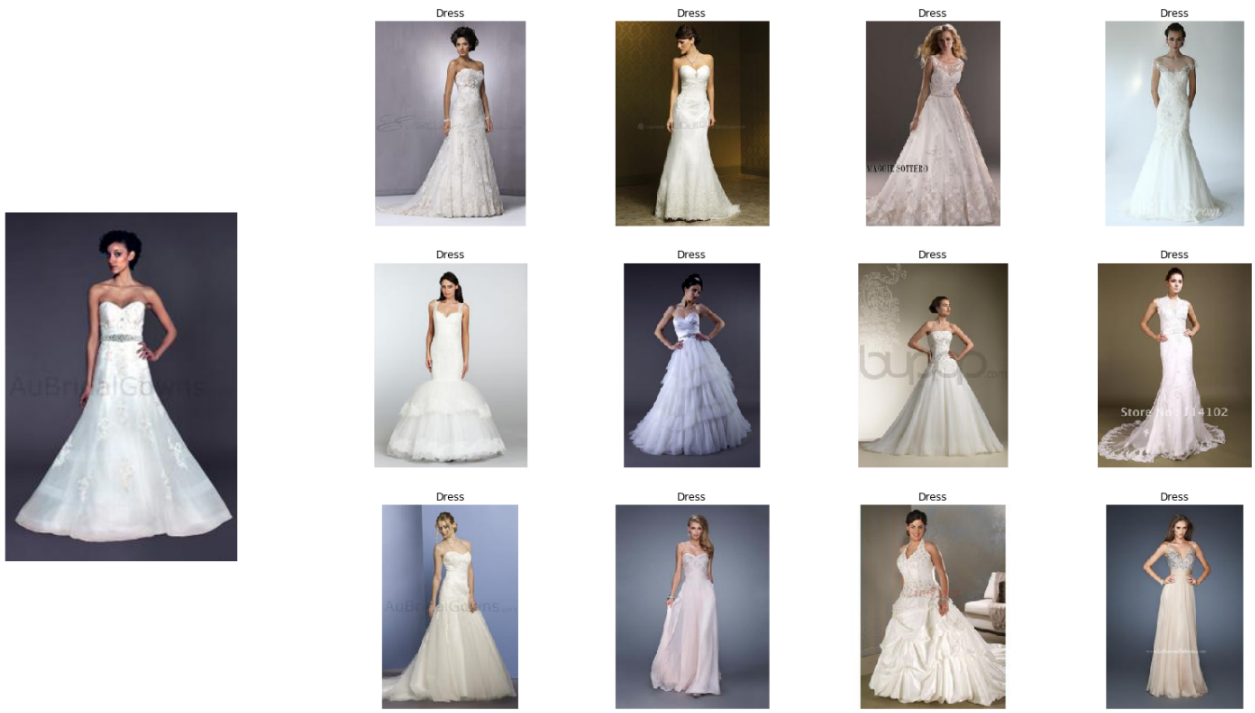
```
    start = time.time()
    base_img_id, base_vector, base_label  = img_repr_df.iloc[img_index, [0, 1, 2]]
    similar_img_ids = t.get_nns_by_item(img_index, 13)
    end = time.time()
    print(f'{(end - start) * 1000} ms')
    return base_img_id, base_label, img_repr_df.iloc[similar_img_ids[1:]]

base_image, base_label, similar_images_df = get_similar_images_annoy(212693)
```

annoy_query.py hosted with ❤️ by GitHub view raw

The approximate query time using Annoy has reduced to **2ms**, which is several orders of magnitude lesser than the naive approach.

Similar Images Recommendations

Here are a few more recommendations obtained by querying the Annoy index.



Bridal Dresses

Tee

Hoodie

Tee

Sweater

Hoodies

Tee

Tee

Tank

Tank

Tank

Tank

Tank

Tee

Tee

Tee

Tank

Blazer

Girls tees

Leggings

Jeans

Joggers

Jeans

Sweatpants

Sweatpants

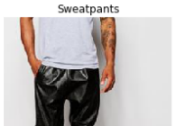
Sweatpants

Sweatpants

Joggers

Jeans

Sweatpants



Leggings

Men's pants

. . .

Resources

[1] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In CVPR, 2016.

Annoy Explanation — Nearest neighbors and vector models — part 2 — algorithms and data structures

ANN algorithms Benchmark

Hope you found this post informative, please leave your comments and thoughts below. 😊

Machine Learning Artificial Intelligence Data Science Convolution Neural Net Fastai

About Help Legal