Product Recommendation by Transfer Learning using CNN

Dickson Wanjau

EAI 6010 Applications of Artificial Intelligence

Prof. Chad Iverson

06/28/2020

Abstract

Product recommendations to customers are one of the most ubiquitous applications of machine learning and artificial intelligence in the e-commerce world. The most prevalent problem for many businesses is how to make effective product recommendation to its customers to increase sales volume, brand awareness and ensure customer loyalty. Recommendation systems apply knowledge discovery techniques such as the notion of similarity between objects, in an approach known as collaborative filtering. Its basic premise is that customers can be considered similar to each other if they share most of the products that they have purchased; equally, items can be considered similar to each other if they share a large number of customers who purchased them.

With the tremendous growth of data at their disposal, companies face daunting tasks of building scalable and effective recommender systems. The key challenges are; producing high quality recommendation systems performing many recommendations per second to thousands of users/ customers and items and achieving high coverage in the face of data sparsity. In traditional collaborative filtering systems, the amount of work increases with the number of participants (both customers and items) in the system. Other recommender systems are needed in such scenarios to produce high quality recommendations, even for large-scale problems.

These include item-based collaborative filtering and leveraging transfer learning machine learning techniques to scale these recommendation systems. Item-based techniques first analyze the user-item matrix to identify relationships between different items, and then use these relationships to indirectly compute recommendations for users. It is a model-based approach to collaborative filtering. The central idea underlying this method is that instead of looking at other users similar to the test user, we will directly recommend items that are similar to the items that have received a high rating by the test user. As we are directly comparing items instead of first comparing users in order to recommend items, we can build up a model to describe the similarity of items and then use the model rather than the entire database to make recommendations.

Transfer learning is a method of reusing a machine learning model. It is sometimes also considered as an extension of existing ML algorithms. Traditionally, if we want to make a machine learning model from scratch, we must start by collecting training dataset, do feature engineering, tuning a hyperparameter then test the model quality. This can consume a lot of time, even money if we train in the cloud service use GPU or TPU. More importantly, this model might not give satisfying results in the end.

In their book, Deep Learning, Goodfellow et al. refer to transfer learning in the context of generalization. Their definition is as follows: Situation where what has been learned in one setting is exploited to improve generalization in another setting. example. Let's assume our task is to identify objects in images within a restricted domain of a restaurant. Let's mark this task in its defined scope as T_1. Given the dataset for this task, we train a model and tune it to perform well (generalize) on unseen data points from the same domain (restaurant). Traditional supervised ML algorithms break down when we do not have sufficient training examples for the required tasks in given domains. Suppose we now must detect objects from images in a park or a café (say, task T_2). Ideally, we should be able to apply the model trained for T_1, but in reality we face performance degradation and models that do not generalize well. This happens for a variety of reasons, which we can liberally and collectively term as the model's bias toward training data and domain. Transfer learning thus enables us to utilize knowledge from previously

learned tasks and apply them to newer, related ones. If we have significantly more data for task T_1, we may utilize its learnings and generalize them for task T_2 (which has significantly less data).

In this paper we analyze on a very high-level the different recommendation engine frameworks. Finally we take a dive into transfer learning through a deep learning approach using Convolutional Neural Networks in the Keras library. We'll train our model on high resolution product image with multiple label attributes describing the product. The idea is given an image ID, can we use visual similarity to suggest other similar images. We'll leverage Neural Network embeddings to translate the high dimensional sparse vector into a low-dimensional space. Neural network embeddings have 2 primary purposes:

- Dimensionality Reduction — more efficient representation
- Finding nearest neighbors in the embedding space - These can be used to make recommendations based on user interests or cluster categories

Introduction

In this section we introduce the different recommender systems in a nutshell to give a little framework to our study. Below is the classification of recommender systems in general:

1. Content based recommendation systems:

Content based recommendation systems focus on properties of items. These recommendation systems use features between the given items and find similarities between them as a measure for giving recommendations. For example, a 20 min workout video on YouTube might get recommended because you have watched a 10 min workout video posted by the same creator. User reviews are not required in this type.

2. Collaborative filtering recommendation systems:

On the other hand, Collaborative filtering systems focus on the relationship between users and items. These models require user reviews as they play a key role in scoring. Collaborative filtering models can recommend an item to user A based on the interests of a similar user B. User's previous history is also taken into consideration.

It consists of a rating matrix comprised of a set of users U = {u1, u2 , …, um} that have varying preferences on a set of items I = {i1 , i2 , …, in }. In addition, users can often express their preference by rating items on some scale. User similarity is then measured using distance measures such as Euclidean Distance, cosine distance (measures the cosine of the smallest angle between two vectors) or Jaccard Similarity. These are illustrated as follows:

$$d_{cosine}(a,b) = \cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|}$$

The cosine distance metric also known as the L2 Norm measures the cosine of the smallest angle between two vectors. If the vectors are parallel to each other, meaning that their angle is 0, then the cosine distance is 0 as well. If the two vectors are at a right angle to each other, then they have the largest distance according to this metric.

The numerator is the dot product between the two vectors and the denominator is the product of the magnitudes (computed via the L2 norm) of the two vectors.

Similarly, the Jaccard similarity is given by the cardinality of the logical intersection divided by the cardinality of the logical union.

$$d_{jaccard}(a,b) = 1 - \frac{|a \cap b|}{|a \cup b|}$$

In a nutshell, it is one minus the ratio of the number of positions in which the two vectors both have a positive rating over the total number of positions in which either of the two vectors have a positive rating.

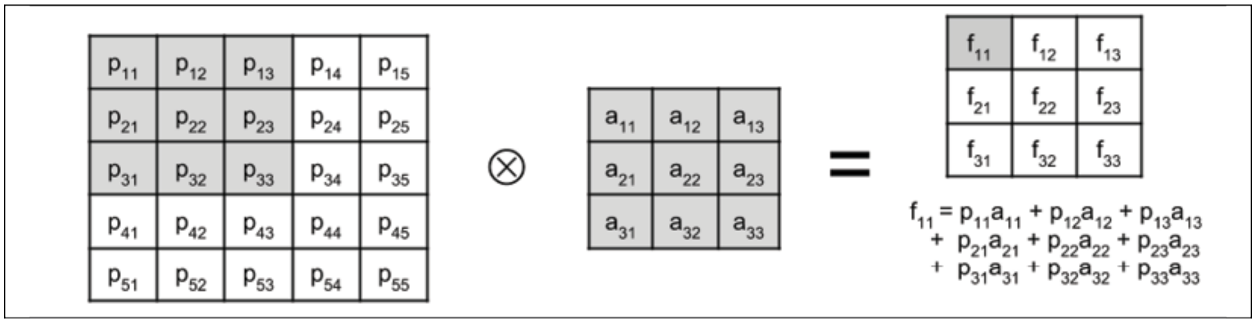
Analysis

Convolutional Neural Networks (CNN)

In our analysis we utilize the Convolutional Neural Networks which are a type of the so called Artificial Neural Networks (ANN).

CNN is a class of deep neural networks which is mostly used to do image recognition / classification, object detection, style transfer etc. In CNN, we take an image as an input, assign importance (learnable weights and biases) to its various aspects/objects in the image and be able to differentiate one from another. The kernel characterizes the CNN operations. The kernel can be visualized as a rectangular window that slides through the whole image from left to right, and from top to bottom. This operation is called convolution (Rowel A. (2020)).

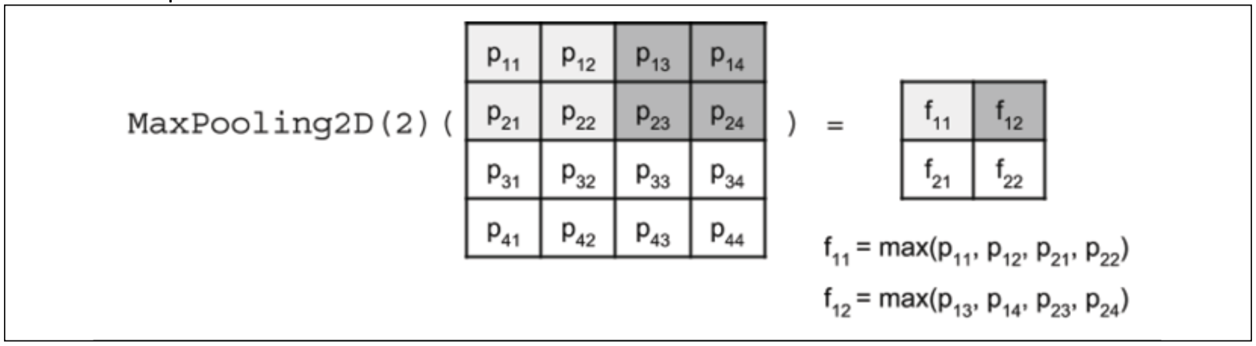
It transforms the input image into a feature map, which is a representation of what the kernel has learned from the input image. The feature map is then transformed into another feature map in the succeeding layer and so on.


p_{11} p_{12} p_{13} p_{14} p_{15}
p_{21} p_{22} p_{23} p_{24} p_{25}
p_{31} p_{32} p_{33} p_{34} p_{35}
p_{41} p_{42} p_{43} p_{44} p_{45}
p_{51} p_{52} p_{53} p_{54} p_{55}
$\otimes$
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
=
f_{11} f_{12} f_{13}
f_{21} f_{22} f_{23}
f_{31} f_{32} f_{33}
$f_{11} = p_{11}a_{11} + p_{12}a_{12} + p_{13}a_{13} + p_{21}a_{21} + p_{22}a_{22} + p_{23}a_{23} + p_{31}a_{31} + p_{32}a_{32} + p_{33}a_{33}$

For example, in the figure above, the 5 × 5 input image (or input feature map) where a 3 × 3 kernel is applied. The resulting feature map is shown after the convolution. We note that the resulting feature map is smaller than the original input image, this is because the convolution is only performed on valid elements. Further, the kernel cannot go beyond the borders of the image.

Lastly, the CNN employs pooling operation such as MaxPooling, GlobalMaxPooling and AveragePooling which compress each feature map. The significance of pooling is the reduction in feature map size, which translates to an increase in receptive field size.

Every patch of size pool_size * pool_size is reduced to 1 feature map point. The value is equal to the maximum feature point value within the patch. For example, after MaxPooling2D (2), the 2 × 2 kernel is now approximately convolving with a 4 × 4 patch. The CNN has learned a new set of feature maps for a different receptive field size as shown below:


MaxPooling2D(2) (
p_{11} p_{12} p_{13} p_{14}
p_{21} p_{22} p_{23} p_{24}
p_{31} p_{32} p_{33} p_{34}
p_{41} p_{42} p_{43} p_{44}
) =
f_{11} f_{12}
f_{21} f_{22}
$f_{11} = \max(p_{11}, p_{12}, p_{21}, p_{22})$
$f_{12} = \max(p_{13}, p_{14}, p_{23}, p_{24})$

AveragePooling takes the average of a patch instead of finding the maximum while Global pooling down samples the entire feature map to a single value.

The main reasons we use CNN's convolution in our project is to extract features such as edges, colors, corners from the input. As we go deeper inside the network (with max pooling), the network starts identifying more complex features such as shapes, digits, face parts as well.

Dataset

The dataset consists of high-resolution product images with label attributes describing the product which was manually entered while cataloging. Each product is identified by an ID like 42413. Due to ram limitations I only use 15000 images in dataset. There are some error rows in dataset, we use error_bad_lines=False to for automatic skip rows these rows.

We load the data into Python as follows specifying the dataset path in order to access the images in jpg. Format.

```python
#Importing requisite libraries
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2 # OpenCV library for Image processing and Computer Vision.
import os
```

```python
DATASET_PATH = "C:\\Users\\dicks\\Desktop\\Pred Analytics\\Spring 2020\\Applications of AI\\175990_396802_bundle_archive"
print(os.listdir(DATASET_PATH))
```

```
['images', 'myntradataset', 'styles.csv']
```

```python
df = pd.read_csv("C:\\Users\\dicks\\Desktop\\Pred Analytics\\Spring 2020\\Applications of AI\\175990_396802_bundle_archive\\m
df.head(5)
```

	id	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName
0	15970	Men	Apparel	Topwear	Shirts	Navy Blue	Fall	2011	Casual	Turtle Check Men Navy Blue Shirt
1	39386	Men	Apparel	Bottomwear	Jeans	Blue	Summer	2012	Casual	Peter England Men Party Blue Jeans
2	59263	Women	Accessories	Watches	Watches	Silver	Winter	2016	Casual	Titan Women Silver Watch
3	21379	Men	Apparel	Bottomwear	Track Pants	Black	Fall	2011	Casual	Manchester United Men Solid Black Track Pants
4	53759	Men	Apparel	Topwear	Tshirts	Grey	Summer	2012	Casual	Puma Men Grey T-shirt

```
df['image_path'] = df['id'].apply(lambda row: 'C://Users//dicks//Desktop//Pred Analytics//Spring 2020//Applications of AI//1
df = df.sample(frac=1).reset_index(drop=True)
df['image'] = df.apply(lambda row: str(row['id']) + ".jpg", axis=1)
df.head(10)
```

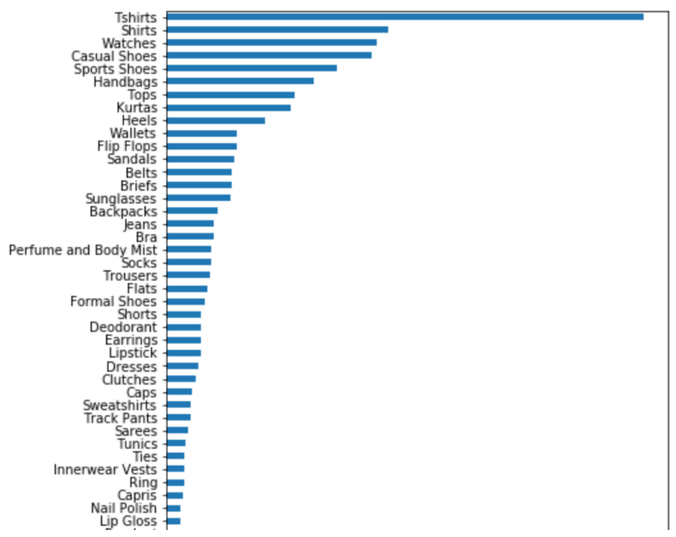
id	gender	masterCategory	subCategory	articleType	baseColour	season	year	usage	productDisplayName	image_path	image
34650	Women	Footwear	Shoes	Sports Shoes	Black	Summer	2012	Sports	Fila Women Top Fuel Black Sports Shoes	C://Users//dicks//Desktop//Pred Analytics//Spr...	34650.jpg
37646	Women	Accessories	Wallets	Wallets	Black	Summer	2012	Casual	Wills Lifestyle Women Black Wallet	C://Users//dicks//Desktop//Pred Analytics//Spr...	37646.jpg
21145	Women	Apparel	Topwear	Shirts	Black	Fall	2011	Casual	s.Oliver Women Printed Black Shirt	C://Users//dicks//Desktop//Pred Analytics//Spr...	21145.jpg

Exploratory Data Analysis.

We plot a bar chart to reveal the most frequent items in the data set as follows:

```
plt.figure(figsize=(7,20))
df.articleType.value_counts().sort_values().plot(kind='barh')
```

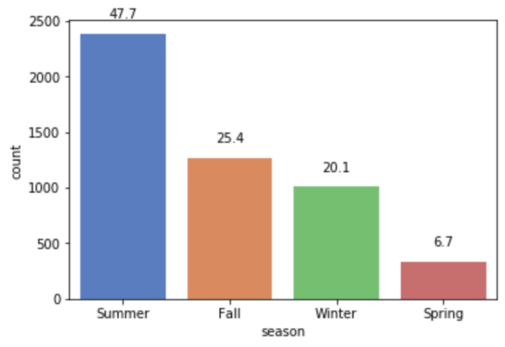
```
<matplotlib.axes._subplots.AxesSubplot at 0x2958c30a2e8>
```



Clothing items such as shirts, shoes and bags form the bulk of the items in our dataset. This is no surprise since our dataset comprises of fashion product items.

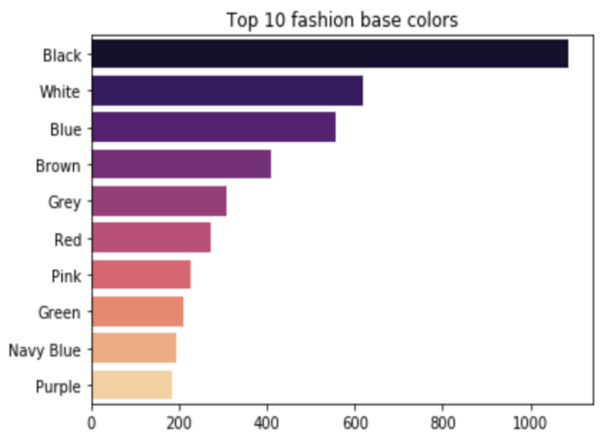
We note that majority of these items were purchased during the summer season as follows accounting for 48% of all items purchased:

```
splot1 = sns.countplot(df['season'], palette='muted')

for p in splot1.patches:
    splot1.annotate(format(p.get_height() / df.shape[0] * 100, '.1f'),
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    rotation=0, ha='center', va='bottom', xytext=(0, 10), textcoords='offset points')

plt.show()
```



For the color of the items, black color was the most prevalent.

```
color_count = df['baseColour'].value_counts()[:10]

sns.barplot(x=color_count.values,
            y=color_count.index,
            palette='magma')

plt.title('Top 10 fashion base colors')
plt.tight_layout()
plt.show()
```



The bulk of the product items were shared between the binary sexes, males and females, with non-binary sex making up for only 4.62%

```
import plotly.express as px
fig = px.pie(df, names='gender')
fig.show()
```

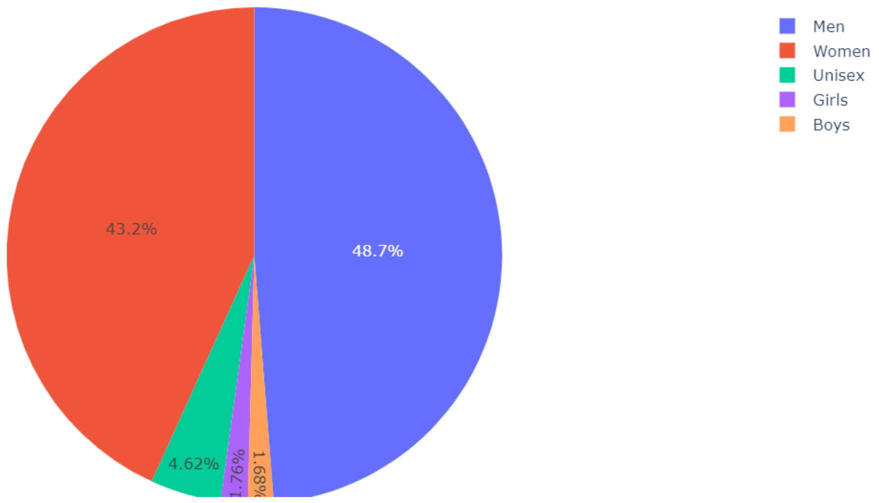


Image Processing and Product Recommendations

We then import our deep learning libraries and use transfer learning by building up on a pretrained model (ResNet 50) to train our CNN model by adjusting and improving upon the weights and biases.

```
import tensorflow as tf
import keras
from keras import Model
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
from keras.layers import GlobalMaxPooling2D
tf.__version__
```

```
# Input Shape
img_width, img_height, _ = 224, 224, 3 #load_image(df.iloc[0].image).shape

# Pre-Trained Model
base_model = ResNet50(weights='imagenet',
                      include_top=False,
                      input_shape = (img_width, img_height, 3))
base_model.trainable = False

# Add Layer Embedding
model = keras.Sequential([
    base_model,
    GlobalMaxPooling2D()
])

model.summary()
```

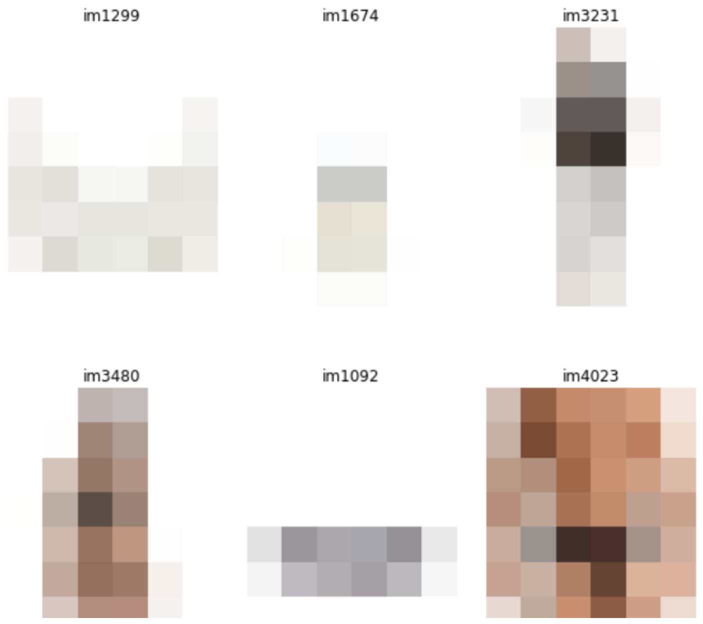
```
WARNING:tensorflow:From C:\Users\dicks\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: coloca
te_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
C:\Users\dicks\Anaconda3\lib\site-packages\keras_applications\resnet50.py:265: UserWarning:

The output shape of `ResNet50(include_top=False)` has been changed since Keras 2.2.0.
```

We define a function to read the images into Python. Due to GPU limitations on my laptop, the images were rendered as blurry.

```
#Leverage OpenCV to process our images
def plot_figures(figures, nrows = 1, ncols=1,figsize=(8, 8)):
    """Plot a dictionary of figures.

    Parameters
    ----------
    figures : <title, figure> dictionary
    ncols : number of columns of subplots wanted in the display
    nrows : number of rows of subplots wanted in the figure
    """

    fig, axeslist = plt.subplots(ncols=ncols, nrows=nrows,figsize=figsize)
    for ind,title in enumerate(figures):
        axeslist.ravel()[ind].imshow(cv2.cvtColor(figures[title], cv2.COLOR_BGR2RGB))
        axeslist.ravel()[ind].set_title(title)
        axeslist.ravel()[ind].set_axis_off()
    plt.tight_layout()

def img_path(img):
    return DATASET_PATH+"/images/"+img

def load_image(img, resized_fac = 0.1):
    img     = cv2.imread(img_path(img))
    w, h, _ = img.shape
    resized = cv2.resize(img, (int(h*resized_fac), int(w*resized_fac)), interpolation = cv2.INTER_AREA)
    return resized
```

```
# generation of a dictionary of (title, images)
figures = {'im'+str(i): load_image(row.image) for i, row in df.sample(6).iterrows()}
# plot of the images in a figure, with 2 rows and 3 columns
plot_figures(figures, 2, 3)
```



We then define a function for the Neural Network embeddings to translate the high dimensional sparse vector into a low-dimensional space. Neural network embeddings have 2 primary purposes:

- Dimensionality Reduction
- Finding nearest neighbors in the embedding space - These can be used to make recommendations based on user interests or cluster categories

```
def get_embedding(model, img_name):
    # Reshape
    img = image.load_img(img_path(img_name), target_size=(img_width, img_height))
    # img to Array
    x   = image.img_to_array(img)
    # Expand Dim (1, w, h)
    x   = np.expand_dims(x, axis=0)
    # Pre process Input
    x   = preprocess_input(x)
    return model.predict(x).reshape(-1)
```

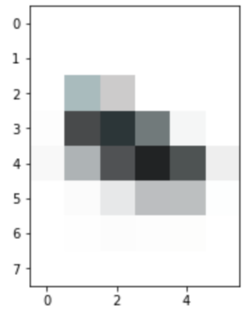
```
emb = get_embedding(model, df.iloc[0].image)
emb.shape
```

```
(2048,)
```

We can obtain the different embeddings for different images by slicing the dataframe.

```
img_array = load_image(df.iloc[0].image)
plt.imshow(cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB))
print(img_array.shape)
print(emb)
```

```
(8, 6, 3)
[-0.27788344 -0.9236274   3.096362   ...  2.0123973   3.7927566
  0.18798757]
```



We convert our images into embeddings using the following code albeit it takes time.

```
%%time
#import swifter

# Parallel apply
df_sample      = df#.sample(10)
map_embeddings = df_sample['image'].apply(lambda img: get_embedding(model, img))
df_embs        = map_embeddings.apply(pd.Series)

print(df_embs.shape)
df_embs.head()
```

```
(5000, 2048)
Wall time: 22min 30s
```

We then compute the pairwise cosine distances between the neural network embeddings to identify similarities among items in our dataset as follows:

```
from sklearn.metrics.pairwise import pairwise_distances

# Calculating
cosine_sim = 1-pairwise_distances(df_embs, metric='cosine')
cosine_sim[:4, :4]
```

```
array([[1.        , 0.5170084 , 0.48579508, 0.49812913],
       [0.5170084 , 0.99999905, 0.3978365 , 0.46903765],
       [0.48579508, 0.3978365 , 0.9999988 , 0.6855088 ],
       [0.49812913, 0.46903765, 0.6855088 , 1.        ]], dtype=float32)
```

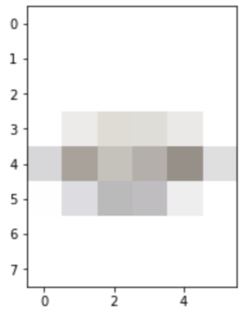
We then compute the recommendations for our top 5 items in the dataset based on the Cosine distance as follows:

```
indices = pd.Series(range(len(df)), index=df.index)
indices

# Function that get movie recommendations based on the cosine similarity score of movie genres
def get_recommender(idx, df, top_n = 5):
    sim_idx    = indices[idx]
    sim_scores = list(enumerate(cosine_sim[sim_idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:top_n+1]
    idx_rec    = [i[0] for i in sim_scores]
    idx_sim    = [i[1] for i in sim_scores]

    return indices.iloc[idx_rec].index, idx_sim

get_recommender(2993, df, top_n = 5)
```

```
(Int64Index([2483, 1574, 354, 2166, 1092], dtype='int64'),
 [0.8868654, 0.88389874, 0.8783474, 0.8746303, 0.8501394])
```

For a particular image, we can observe the top 6 recommended image items using the following Python function.

```
idx_ref = 2993

# Recommendations
idx_rec, idx_sim = get_recommender(idx_ref, df, top_n = 6)

# Plot
#===================
plt.imshow(cv2.cvtColor(load_image(df.iloc[idx_ref].image), cv2.COLOR_BGR2RGB))

# generation of a dictionary of (title, images)
figures = {'im'+str(i): load_image(row.image) for i, row in df.loc[idx_rec].iterrows()}
# plot of the images in a figure, with 2 rows and 3 columns
plot_figures(figures, 2, 3)
```



References.

1. Rowel A. (2020). *Advanced Deep Learning with TensorFlow 2 and Keras*. Athens, Greece. Packt Publishers
2. Dowel O. (2018). Deep Learning Cookbook. Boston, MA. O'Reilly Media, Inc.