**Project: Identifying a similar fashion apparel**

In this module, you are going to learn about a rich technique in deep learning named as transfer learning. For this, we will be using a fashion apparel data set fetched from Amazon.com using their Product Advertising API. When you visit any e-commerce website and open the webpage of a product, you must have seen some similar looking products somewhere on the page. This is done using the recommendation systems.

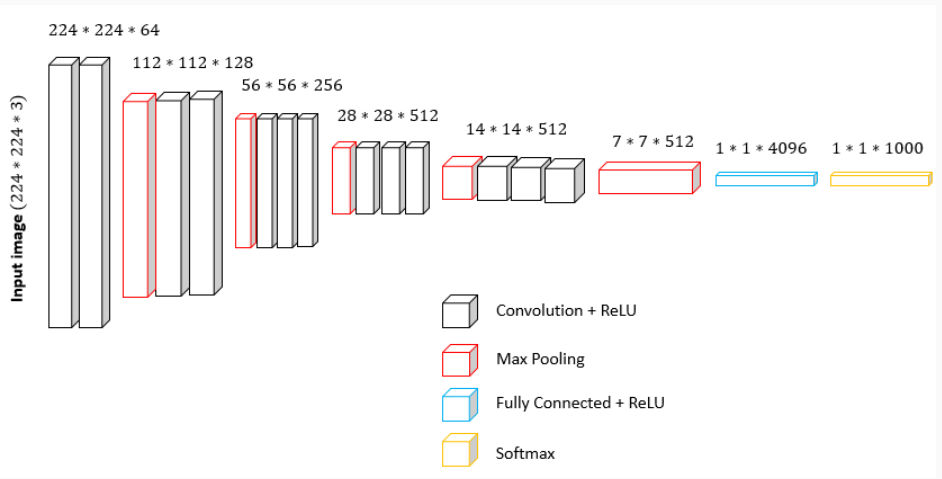
Here, we will learn how transfer learning can be helpful to answer this question.

Introducing transfer learning

Transfer learning is a procedure to use the parameters of a model in another model for a similar problem. This learning became important because it is not practical for everyone to train a model on huge data sets which requires ample hardware and software requirements. Therefore, if a model is trained on a high-end configuration system then the parameters of that model can be utilized in a system with a low-end configuration.

In this module, we are going to focus on a variant of CNN named VGG 16 to import its parameters and apply it in our case.

# VGG16 architecture is shown below:



The following points help to understand the architecture more clearly:

* The input image has a fixed size (224x224x3), three channels signify an RGB image.
* The stack of convolutional layers (black blocks) consists of a receptive field with size 3x3.
* The stride for the convolutional is 1 pixel.
* The padding for 3x3 convolutional layers is 1 pixel.
* It implements pooling with a stride of 2 and 2x2 pixel window.
* The blue block above indicates three full-connected layers with the first two having 4096 channels and the last having 1000 channels.
* The last yellow block signifies the usage of softmax layer.
* As an added note, all the hidden layers consist of ReLU activation function.

Finding similar fashion apparel – Implementation

So, before we start with the implementation, download the [images](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_012792125451714560326/web-hosted/assets/16kimages.zip), [their information](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_012792125451714560326/web-hosted/assets/apperaldata.csv), [features and Amazon Standard Identification Numbers (ASIN)](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_012792125451714560326/web-hosted/assets/features.zip) for each image to be used. Files are placed in google drive – Fashion Apparel- datasets

We first start by importing the libraries and dataset path:

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. from PIL import Image
5. from keras.preprocessing.image import ImageDataGenerator
6. from keras import applications
7. from sklearn.metrics import pairwise\_distances
8. *# Defining the image shape and path*
9. img\_width, img\_height = 224, 224
10. train\_data\_dir = '16k\_images/'

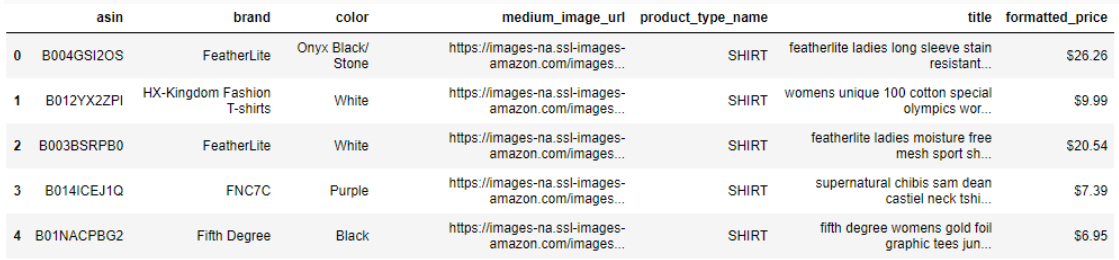
Next, we are going to build the VGG model on the available apparel dataset and also fetch the corresponding ASIN information. Since we have quite a lot of images and VGG are known to be way too slow, therefore, it is suggested to skip the execution of the code block presented below. Rather, we have provided you the corresponding results of this code block right at the starting of this page.

**Note**: The code took almost 7 hours on AMD PRO A6-9500 R5, 8 COMPUTE CORES system with 8 GB RAM.

1. *# nb\_train\_samples = 16022*
2. *# epochs = 50*
3. *# batch\_size = 1 # Number of images to be processed at a time*
4. *# asins = []*
6. *# # Building the VGG16 network*
7. *# model = applications.VGG16(include\_top=False, weights='imagenet')*
8. *# generator = ImageDataGenerator(rescale=1. / 255).flow\_from\_directory(train\_data\_dir,*
9. *# target\_size=(img\_width, img\_height), batch\_size=batch\_size,*
10. *# class\_mode=None, shuffle=False)*
11. *# # Retrieving the ASIN information without any extension, hence i[2,-5]*
12. *# for i in generator.filenames:*
13. *# asins.append(i[2:-5])*
14. *# features = model.predict\_generator(generator, nb\_train\_samples // batch\_size)*
15. *# features = features.reshape((16022, 25088))*
16. *# # Saving the files in a NumPy format*
17. *# np.save(open('cnn\_features.npy', 'wb'), features)*
18. *# np.save(open('cnn\_feature\_asins.npy', 'wb'), np.array(asins))*

You can start by loading the given **.npy** files and the dataset:

1. *# Loading the features and corresponding ASINS info.*
2. features = np.load('cnn\_features.npy')
3. asins = np.load('cnn\_feature\_asins.npy')
4. asins = list(asins)
5. *# load the original 16K dataset information*
6. data = pd.read\_csv('apperal\_data.csv')
7. df\_asins = list(data['asin'])
8. *# Data overview*
9. data.head()



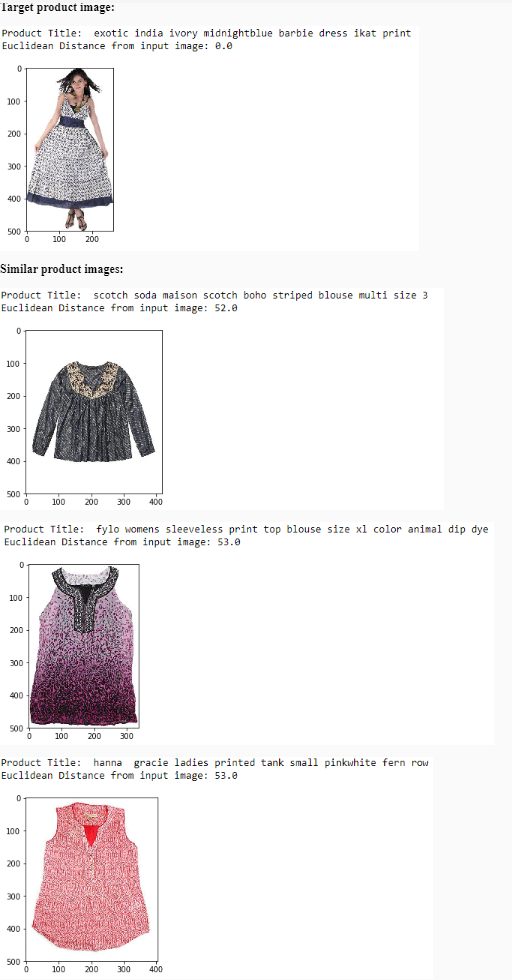
Now, we can call the features and use the Euclidean distance as a measure to compute the similarity between the images as shown:

1. *# Get similar products using CNN features (VGG-16)*
2. def get\_similar\_products\_cnn(doc\_id, num\_results):
3. doc\_id = asins.index(df\_asins[doc\_id])
4. pairwise\_dist = pairwise\_distances(features, features[doc\_id].reshape(1,-1))
5. *# Sorting the values in ascending order to pick up most similar images first*
6. indices = np.argsort(pairwise\_dist.flatten())[0:num\_results]
7. pdists = np.sort(pairwise\_dist.flatten())[0:num\_results]
8. *# Displaying the target and similar product images*
9. for i in range(len(indices)):
10. rows = data[['asin', 'medium\_image\_url', 'title']].loc[data['asin']==asins[indices[i]]]
11. for indx, row in rows.iterrows():
12. print('Product Title: ', row['title'])
13. print('Euclidean Distance from input image:', np.round(pdists[i]))
14. plt.imshow(Image.open('16k\_images/' + str(row['asin']) + '.jpeg'))
15. plt.show()

The above code computes the smallest Euclidean distance from a target image and displays the target image along with the number of similar images. We have used the NumPy's round function while displaying the Euclidean distance in the output. You can change it as per your convenience.

Here is an example:

1. *# First image is the target image and the following three images are similar images*
2. get\_similar\_products\_cnn(4761, 4)



Problem:

You are given with the cars dataset consisting of 8000+ images of various types of cars. You need to cluster all the images into 10 clusters using Transfer learning and KMeans clustering. The dataset can be downloaded from [here](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0128013557486960646568/web-hosted/assets/carstrain.zip) and you can use the VGG16 weights from [here](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0128013557486960646568/web-hosted/assets/vgg16weightstfdimorderingtfkernelsnotop.h5).

For your reference, the solution on a small dataset is presented in this [IPython notebook](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0128013557486960646568/web-hosted/assets/Transferlearningclustering.ipynb" \t "_blank).

This module gives you an idea about additional topics to explore and additional tools to help you code Deep Learning in Python. Here's the list of the same:

# Additional Learning

* Generative Adversarial Networks
* Restricted Boltzmann Machines
* Capsule Networks
* Deep Reinforcement Learning

# Additional Tools

* Tensorflow
* PyTorch
* Apache MXNet
* Theano

# Generative Adversarial Networks (GAN)

There are two modules in a GAN network, generator, and discriminator. If the scenario is to create handwritten numerals then the Generator takes some random weights, creates a numeral image which is fed into discriminator along with the real image. The discriminator tries to distinguish the real from the fake and thus provides feedback to the generator. The idea of generative learning is: given the labels how likely are the features. They have seen quite a lot of applications ranging from creating new clothes design to fake faces.

# Restricted Boltzmann Machines (RBM)

RBM belongs to unsupervised learning, generative model and is a subpart of Boltzmann Machines. They are also the building block of Deep Belief Networks which are composed of multiple layers of hidden units. They are composed of only input and hidden layers (no output layer) and are often stacked on top of each other where the output of one proceeds as the input to the next forming a feed-forward network. Their applications include classification, recommendation systems (collaborative filtering), dimensionality reduction, etc.

# Capsule Networks

Capsules networks are introduced to overcome the limitations of CNN. CNN is well known for its groundbreaking performance the time it became known to the world. But their limitation arises when they are not able to grasp the spatial information about an image. For example, if a CNN is trained to recognize a face, and you pass an image with shuffled face features, then still CNN can result in a positive answer. Capsule network overcomes this limitation by preserving the translational and rotational relationship between the simpler features.

# Deep Reinforcement Learning

Reinforcement learning, combined with the power of neural networks, has led to great products like AlphaGo. The basic intuition behind deep reinforcement learning is similar to that of reinforcement learning which is based on goal-oriented learning where their main objective is to maximize along a particular dimension, for instance, achieving more number of points in a game.

In this course, we have implemented all the topics using Keras. However, Keras is just one of the framework which is built upon the TensorFlow (a product by Google). Apart from Keras, you can also use the following deep learning software:

# TensorFlow

* An open-source library released under the Apache License 2.0.
* **Language compatibility:** Python and C.
* Developed by Google Brain.

# PyTorch

* An open-source library released under one of the BSD licenses and based on Torch library.
* Caffe2 (another popular DL framework) was merged with PyTorch in 2018.
* **Language compatibility:** Python
* Developed by Facebook.

# Apache MXNet

* An open-source library released under the Apache License 2.0.
* **Language compatibility:** C++, Python, R, Scala, Julia, Perl, MATLAB and JavaScript.
* Developed by Apache Software Foundation
* Amazon has chosen MXNet as the main DL framework of choice in AWS. It is also available in Microsoft Azure.

# Theano

* An open-source library released under the 3-Clause BSD License
* **Language compatibility:** Python
* Developed by Montreal Institute for Learning Algorithms (MILA), University of Montreal.