# CAPSTONE PROJECT 2:

# A Simple Product Recommendation System with Transfer Learning

*Aditya Jakka*
*January 2019*

***Abstract***

If we hope to achieve a decent product recommendation system, it would have been trained using data from millions of products. For companies like Amazon, product ratings are also taken into account. Such companies not only have the data but also time and resources to implement a state of the art machine learning model to serve as their product recommendation system.

But that would be do if we're in a situation where we have a small/mid-sized company that doesn't have anywhere nearly as many products as Amazon? How would they obtain data? How would they train a model?

This project will propose an interesting and alternative approach to tackle this issue.

## TABLE OF CONTENTS

## Goal of the Project

Given an amazon product description, and its sentence embeddings computed using the sentence encoder, can we compute the nearest neighbor embeddings and have the encoder work like a recommendation system? For example, if we have a Banana Republic t-shirt, using this method, can we expect to see other similar t-shirts or other Banana Republic products? That would be very useful for an e-commerce/retail company, wouldn't you agree?

The project has been worked on keeping in mind both technical and non-technical audiences. Visuals have been used wherever possible to help us better understand basic concepts and interpret the results.

## Motivation for this project and practical application

Any decent recommender model would have been trained using data from millions of products. For companies like Amazon, such large amounts of data is easy to come by. But that's not all! Can the same recommender model be used for a year or months at a stretch without any modifications? Probably! Or probably not. Products evolve over time. Companies do away with older products and bring in newer ones every day. Companies like Amazon have the time and resources to ensure that their recommendation systems are up to date with the latest trends.

It's also important to consider data scarcity, which is something a lot of companies have to deal with it. Good quality data isn't easy to come by. A machine learning model is only as good as the data it is fed. Even the best data scientists cannot work together to put together a state of the art machine learning model without data. More data-->Better Machine Learning models. This is one of the major motivations for this project. To counter data scarcity.

Moreover, neural networks are being used for recommender systems. These models are designed using sophisticated state of the art architecture. The model needs to be designed first. Then they will be tested and modified again and again. Fine tuning deep learning models can be a monumental task. Now, what if we could come up with a decent recommender process that involves no training at all. Thanks to Google's universal sentence encoder, this is possible! Given a product description as text, this text input will be mapped to a high dimensional vector space (512 dimensions). These "embeddings" are then used to compute similarities based on which practical recommendations can be made.

## ***Concepts Explored***

### Word embeddings

Word embeddings are vector representations of words in a high dimensional space. These vectors are often numpy arrays. Now, one may wonder what's so special about these word vectors (aka embeddings). Well, that's how the magic happens. When these word vectors, which are basically numpy arrays, are plotted in the n-dimensional space, then words that are "similar" in the semantic sense are closer to each other in distance. For example - the words 'oranges' and 'apples' will be closer to each other than apples and vegetables or apples and shoes and so on. Word2Vec by Google was a state of the art NLP technique for finding words or sentences that are similar in a semantic sense.

While this is great, often times, the models have to be trained using a corpus of a large number of texts to get decent results. The quality of texts also play a huge role in determining the practical efficiency of the model.
Let's imagine a scenario where it's not possible for us to train a state of the art model using a corpus of large texts. Let's see we only have limited data to work with. This is a problem often encountered in the real world. Hence, its **TRANSFER LEARNING** to the rescue.

### Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on another task. This is, we can use the saved model (with its weights and architecture) and train this for a different problem.

Transfer learning has picked up pace and popularity in the world of Computer Vision. Some of the popular models used for transfer learning are VGG16, VGG19, MobileNet, Resnet50, Inception and so on. These models were trained on millions of images that can be grouped into thousands of categories. What's so impressive about this is that we probably don't ever have to train and image classification model from scratch. It's because this model has learned about objects, their shapes and other attributes.

Let us consider a few examples to understand what transfer learning really is. If a person knows how to boil an egg, he/she must surely know how to boil potatoes? Or, if we know how to program

using C, then we shouldn't have much trouble learning C++. The core basic programming concepts do not change. While we still have some learning to do, we definitely don't need to start from the very basics like we did with C. Hopefully, we can all agree on this one. OK, let's try another one. If one can drive a sedan, then surely he/she can drive a truck/bus too? Of course they aren't the same thing. But he/she can use his/her basic knowledge of driving, roads and traffic rules to learn to drive a bus or truck. That's where Transfer learning comes in. We don't have to train our models from scratch.
What is being implied here is that we don't necessarily have to reinvent the wheel each time for a different task. We'll be using **Google's Universal Sentence encoder** for this project.

## Universal Sentence Encoder

Through empirical tests, Google suggests that sentence level encodings as opposed to word embeddings gives more meaningful results and is better at generalizing and recognizing context. This sentence encoder encodes each input (the input can be a word, sentence, or paragraph) into a 512 dimensional vector space. This is represented by an array of numbers. These numbers in an array don't mean much on their own. We can think of these 512 numbers as coordinates in a 512 dimensional space. The higher the dimensional space, greater the accuracy but this is comes at the expense of computational power. The Universal Sentence encoder was trained using the **SNLI Corpus**. This module can be used for a variety of NLP tasks like clustering, classification and also for finding semantic relationships between texts.

When training models using word2Vec, we have to perform data preprocessing steps. These steps involve removing special characters, white spaces and stop words. We don't need to remove stop words when using the sentence encoder.

Google has published 2 state of the art universal sentence encoders -
1) This **module** is what we'll be using. It's based on a generater encoder. It has been trained to achieve very accurate results. The tradeoff here is more compute power.
2) This **module** is based on DAN (Deep Averaging Network). This encoder is a lot easier on the RAM and our tradeoff here is some compromised accuracy.
For more details on these encoders, please refer to this **article by VISHWESH SHRIMALI**.
Now, let's consider a visual explanation of how this module works. We'll also use Principal Component analysis for better insights as to how this model performs. Next, we'll discuss the

cosine similarity, which is a metric we'll be using to find the closest sentence embeddings which will help us retrieve the closest sentences in the semantic context.

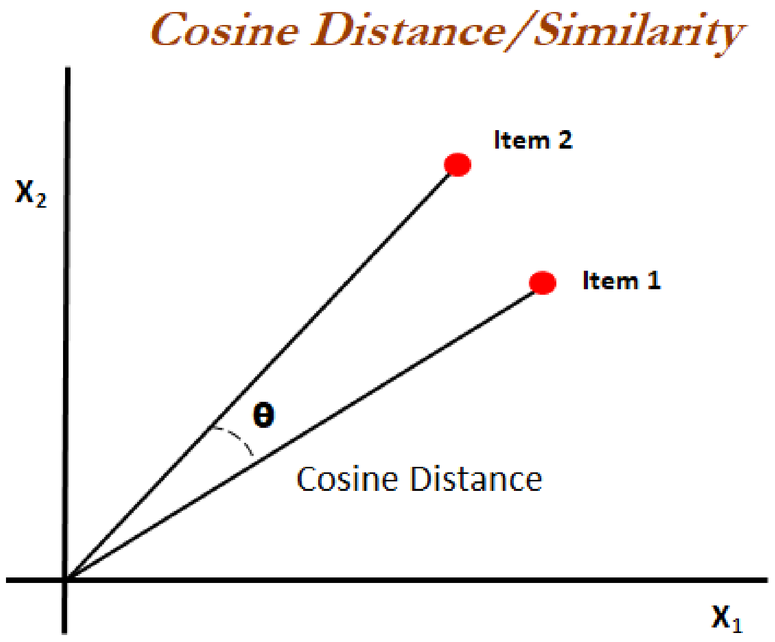
## Cosine similarity

It is a measure of similarity between two non-zero vectors. Given two arrays of the same shape, this metric calculates the angle between these arrays (sentence embeddings which are vectors/arrays). It is important to note that magnitude of the vectors has no effect on this metric. Rather, it is the orientation that matters (please refer to the second image below). It's important to note that the cos(0) is 1 and cos(90) = 0. Hence, vectors aligning in the exact same direction will have an angle of 0 degrees between them. Hence, computing the cosine (cosine similarity measure) gives us 1.

Let's take a look at the formula. The numerator represents the sum of the dot product between the vectors. The first term in the denominator represents the square root of the sum of the elements in array A. The second term represents the square root of the sum of elements in array B. The numerator is the sum of the dot product of the two vectors.

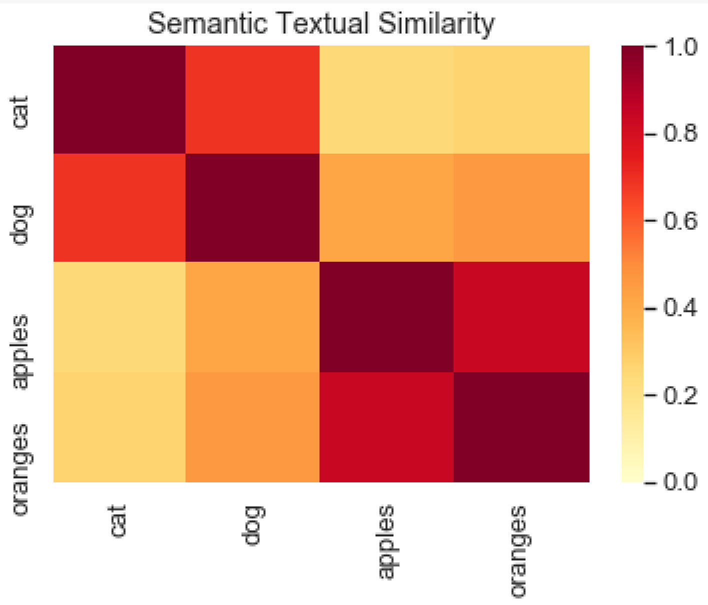
$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Now, observe how the cosine similarity is actually computed in the high dimensional space:

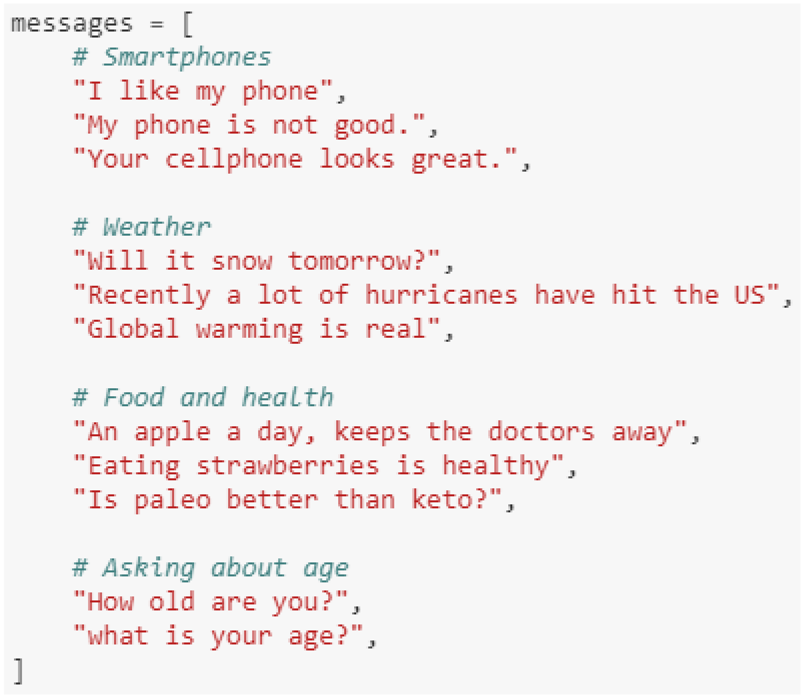


Now, let us plot a similarity matrix for the following words as per the sentence encoder.

Messages = ["cat", "dog", "apples", "oranges"]



The words "cat" and "dog" being animals are closely correlated to each other than "cat" and "apples" or cats and oranges. Similarly, apples and oranges being fruits seem to be highly correlated to each other. Something like this has already been achieved time and again using word2vec. So let's dig deeper and see how different the sentence encoder is. Let us consider the list of sentences below. We have sentences belong to 4 different contexts.



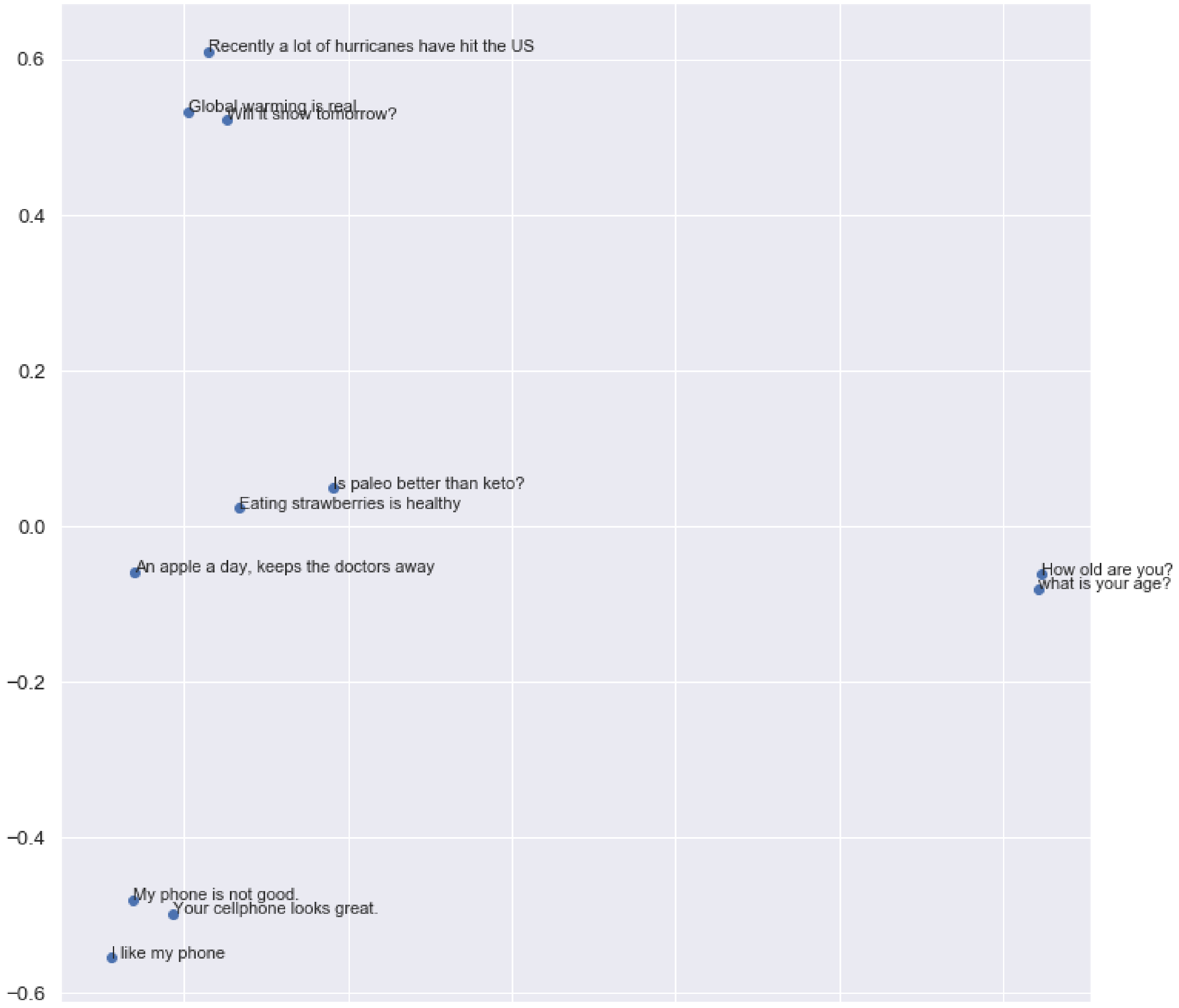
```
messages = [
    # Smartphones
    "I like my phone",
    "My phone is not good.",
    "Your cellphone looks great.",

    # Weather
    "Will it snow tomorrow?",
    "Recently a lot of hurricanes have hit the US",
    "Global warming is real",

    # Food and health
    "An apple a day, keeps the doctors away",
    "Eating strawberries is healthy",
    "Is paleo better than keto?",

    # Asking about age
    "How old are you?",
    "what is your age?",
]
```

The sentence embeddings for the above sentence were computed and then, using PCA, the dimensions were reduced to just 2. Let's take a look at the results below:



We observe that the sentences have been grouped into separate clusters based on context. What's even more incredible is that we achieved this result without even training the model. This is how sentence embeddings work. Sentences that are related in a semantic sense are closer to each other than those that aren't.

Next, we'll be applying these sentence transformation to ~200,000 amazon product descriptions and ~20000 movie synopses.

## Obtaining Data and Preprocessing

1. Obtain the data from this website. For this project, Amazon product files of 2 different categories: "Home and Kitchen" (Click here for direct link), and "Office Products" (direct link). Clothes and jewelry json file can be directly downloaded here. Lastly, the movies dataset can be downloaded here.

   Once downloaded, we may want to move the files into the raw data folder.

2. Clean the data for analysis. Once the raw data files have been downloaded, they need to be unzipped. This project was executed using Ubuntu 18.04 OS. Navigate to the location of your raw data files. Then run the following commands (for Ubuntu users) using terminal:

   - gunzip meta_Home_and_Kitchen.json.gz
   - gunzip meta_Office_Products.json.gz

   (Mac users may want to refer to this guide to unzip files)

   We can move the files to the interim data folder after the above step. Once the files have been unzipped, we'll have the json files in place. Json files aren't easy to work with and hence, it's up to us to convert the data in this json file into a nice Pandas dataframe that we can use. The cleanup files can be found in the interim data folder. But you may directly jump into the notebooks using the links below:

   - Clean up Home and kitchen json (for cleaning "Home and kitchen" json),
   - Clean up Office products json (for Office products json file).
   - Clean up clothes and jewelry json (for clothes and jewelry json file).
   - Clean up movies data file (for movies data file).

The function "ast.literal_eval" has been used to read json files. This function raises an exception if we have an invalid Python data type. Special characters were omitted from product descriptions.

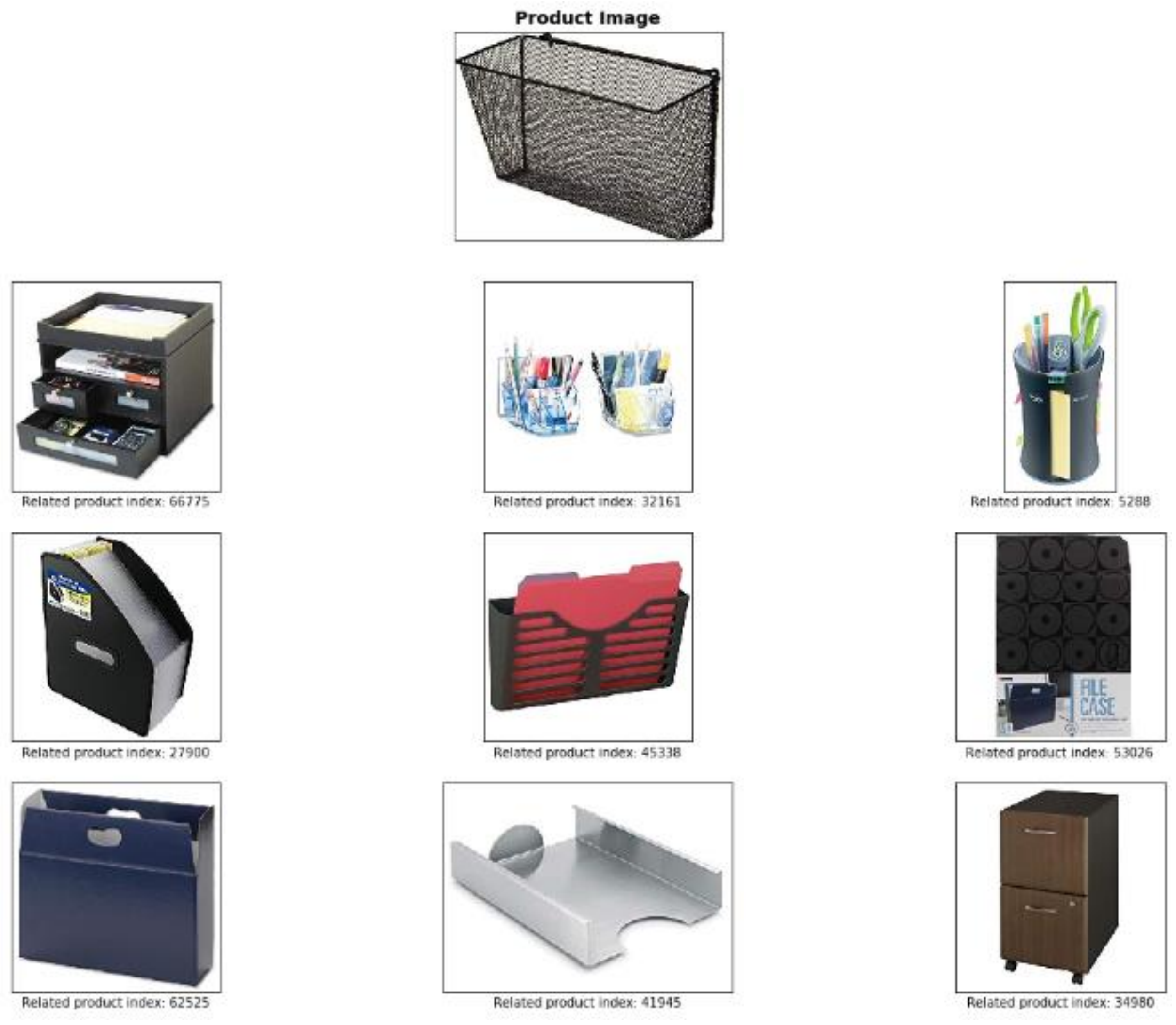
The cleaning steps are exactly the same line for all the four files. They only differ by file names. Once done, we move the files to the processed data folder.

## The Recommendation System at work

For this section of the project, only a handful of results will be presented and discussed. The complete jupyter notebooks for all of the products and movie recommendations can be found here.

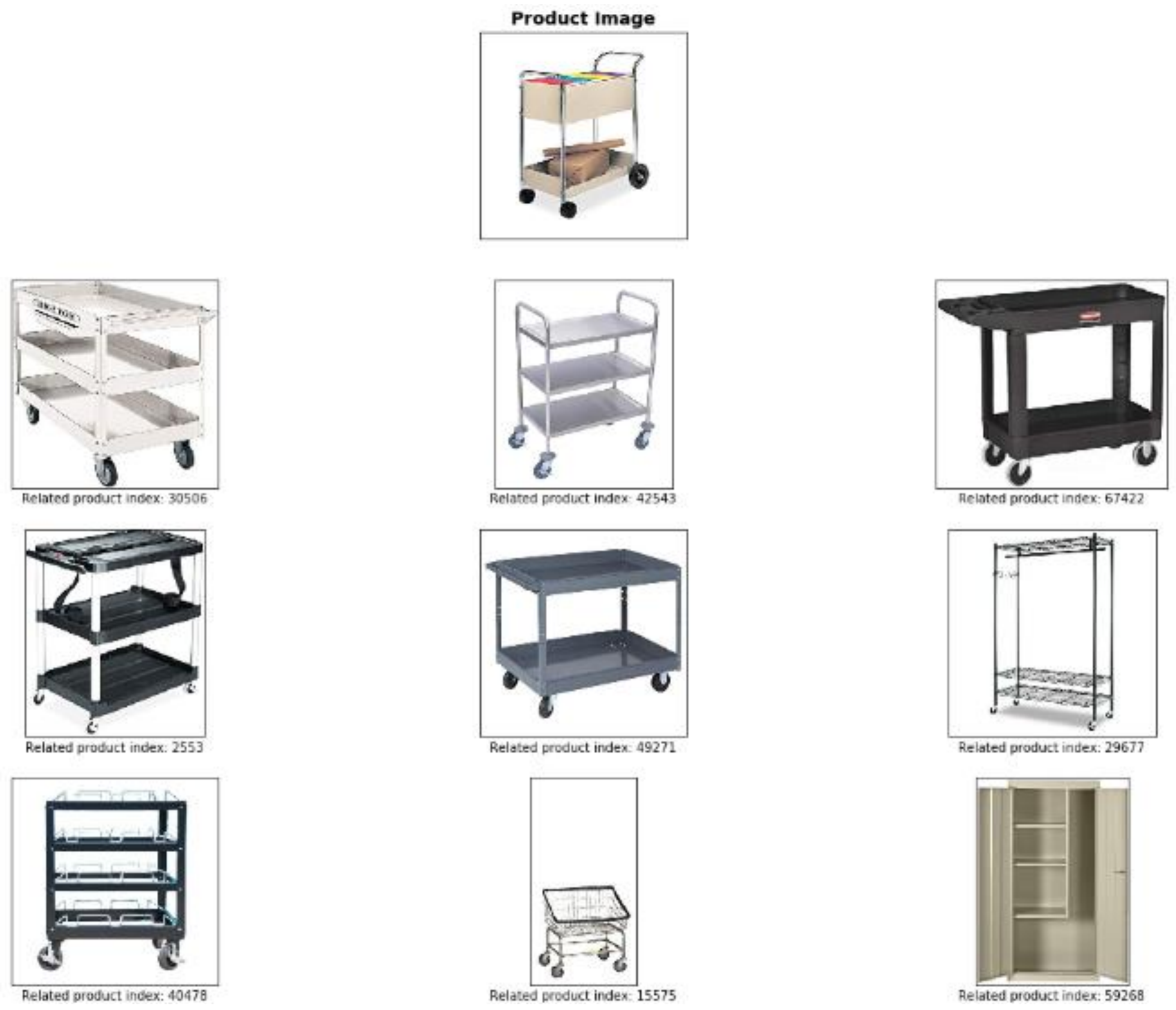
Let us observe a few results from the Office products category.

First up, we have a file holder accessory which is represented by the first image. The 9 products below are its related products.


Product Image
Related product index: 66775
Related product index: 32161
Related product index: 5288
Related product index: 27900
Related product index: 45338
Related product index: 53026
Related product index: 62525
Related product index: 41945
Related product index: 34980

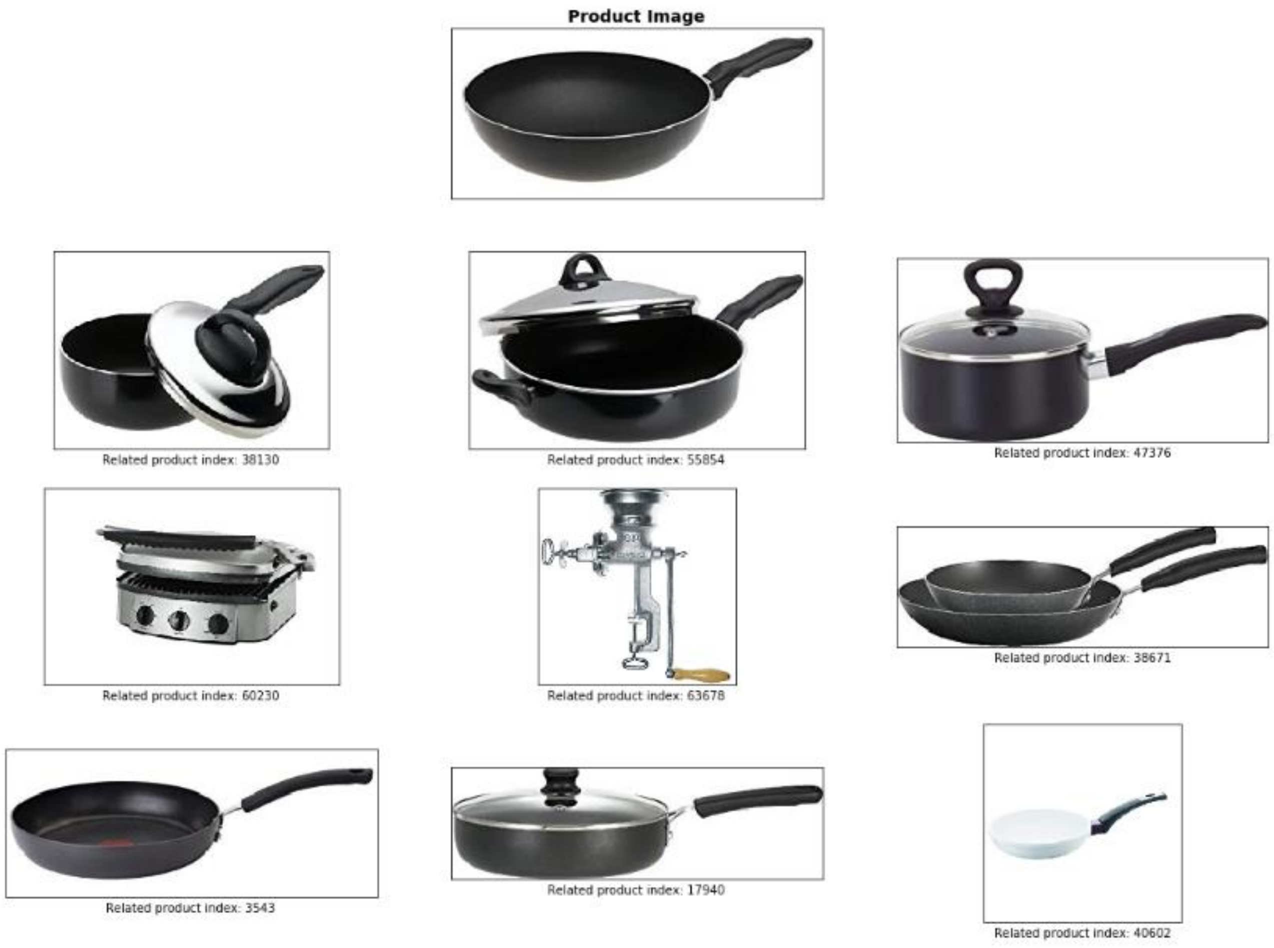
For the related products, we have other file holders, desktop organizers, stationary holders and even an office drawer. In the semantic sense, we can think of how all of these products are similar – they are used to hold or contain objects like documents or stationary.

Let’s take a look at another product from the office products category. What we have below is a mail cart made of steel. It is also capable of holding folders


Product Image
Related product index: 30506
Related product index: 42543
Related product index: 67422
Related product index: 2553
Related product index: 49271
Related product index: 29677
Related product index: 40478
Related product index: 15575
Related product index: 59268

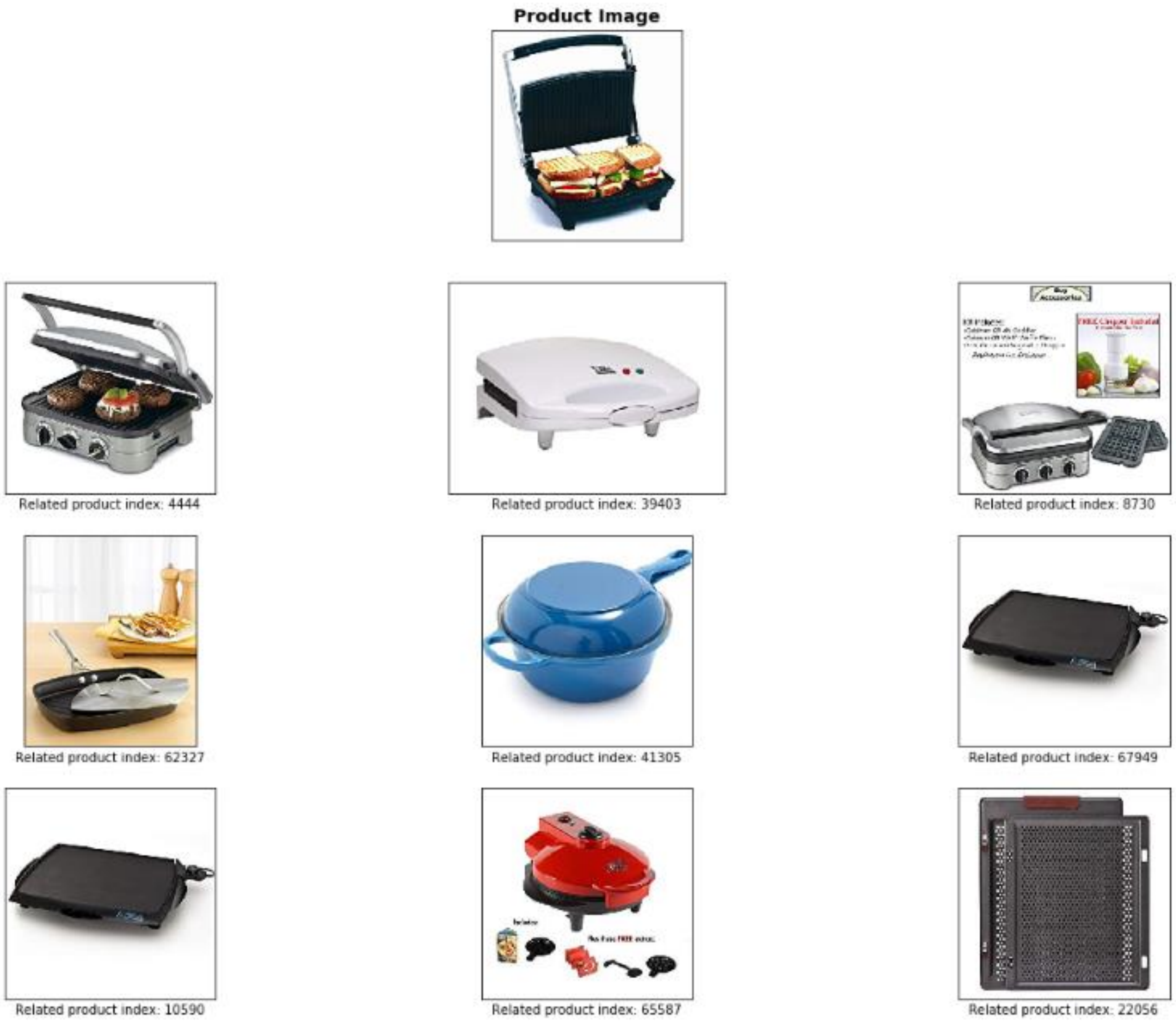
We have a shelf cart, a utility cart, a garment rack (29677) and even a laundry cart (Product index: 15575). We also have a cabinet (Product index: 59268). We can consider the cabinet and laundry cart as “outliers” here.

Next up in the kitchen products category, we have a frying pan.


Product Image
Related product index: 38130
Related product index: 55854
Related product index: 47376
Related product index: 60230
Related product index: 63678
Related product index: 38671
Related product index: 3543
Related product index: 17940
Related product index: 40602

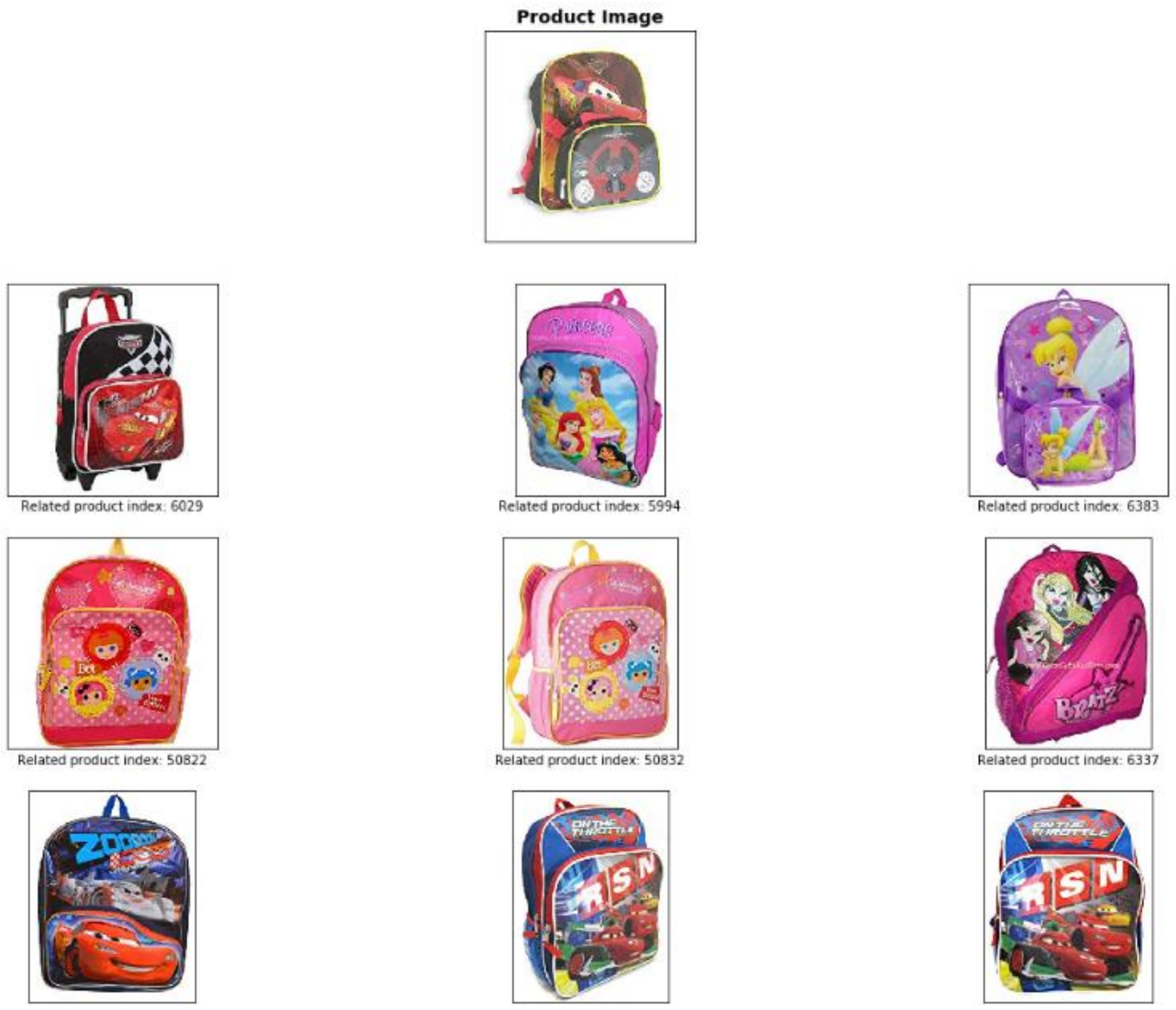
Let’s observe the related products. We have other frying and stir pans.

We have a Panini Grill as the main product. Let’s check out some of the related products.


Product Image
Related product index: 4444
Related product index: 39403
Related product index: 8730
Related product index: 62327
Related product index: 41305
Related product index: 67949
Related product index: 10590
Related product index: 65587
Related product index: 22056

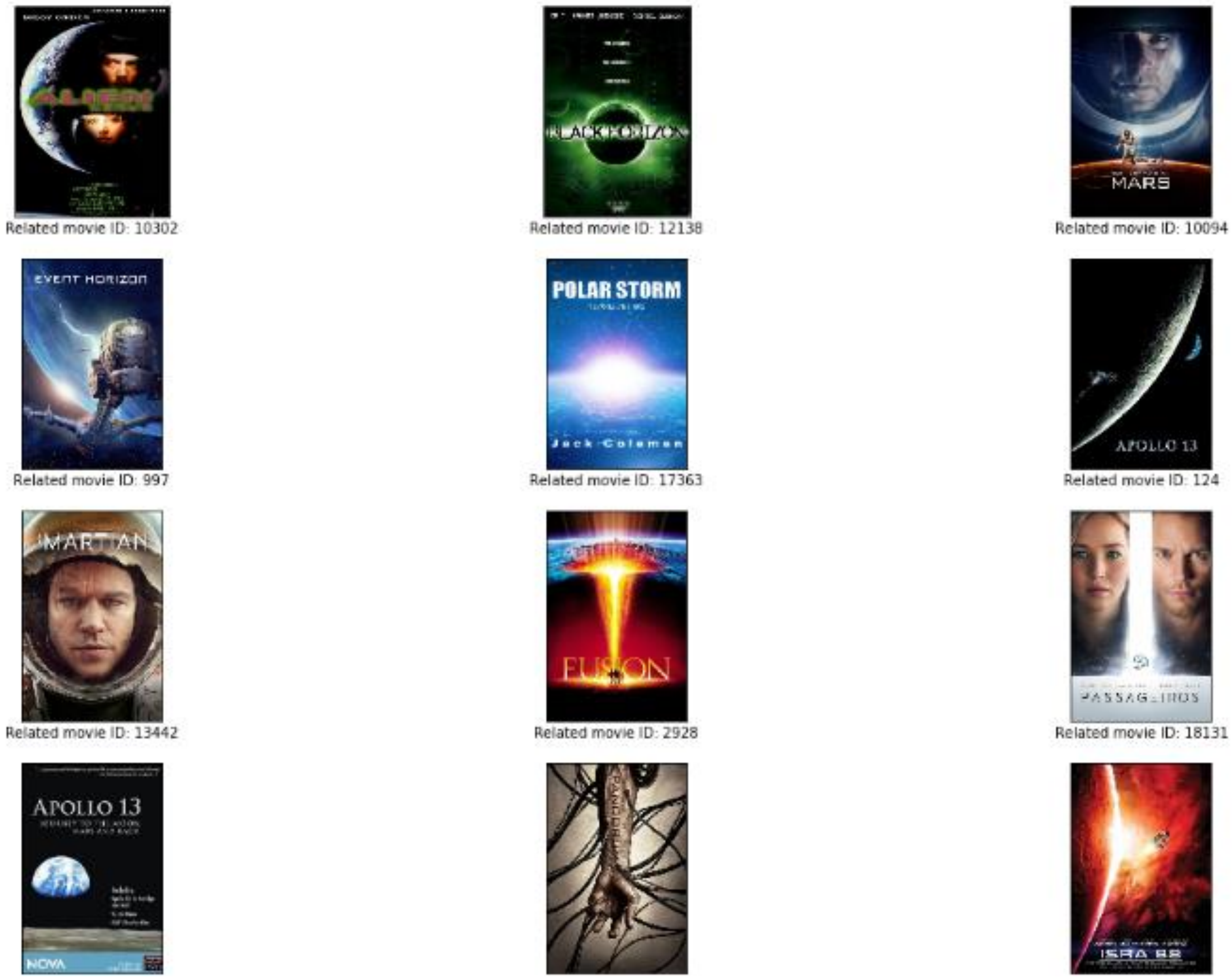
We get other grill machines and even frying saucepans  (Product indices: 41305 & 22056.

Next, we have a kids backpack. The theme of this backpack has been inspired from the “Cars” movie franchise.


Product Image
Related product index: 6029
Related product index: 5994
Related product index: 6383
Related product index: 50822
Related product index: 50832
Related product index: 6337

We observe that the rest of the related products are also kids’ backpack which are either inspired from the “Cars” franchise or other children’s movie franchises.

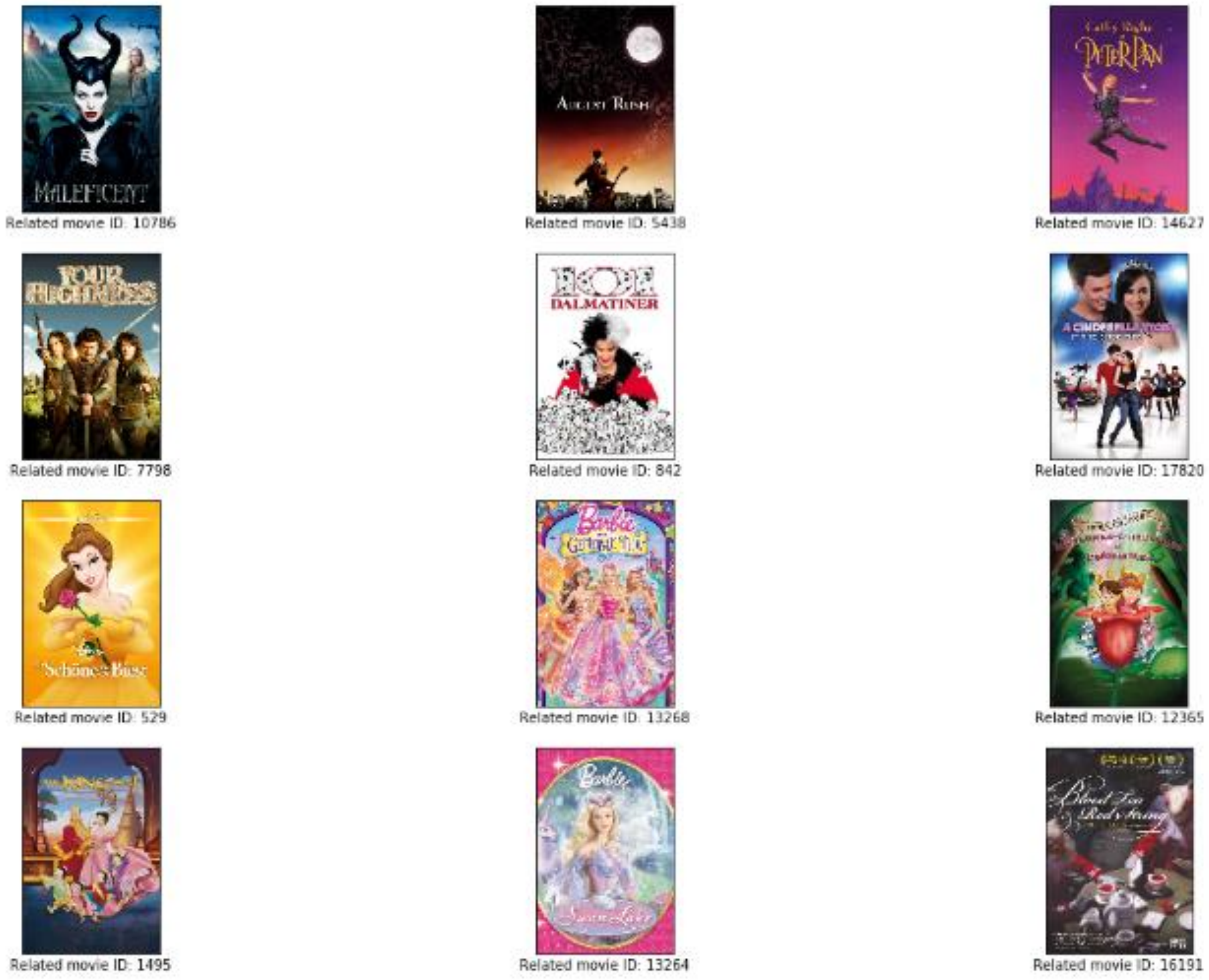
Moving on to the movies section of this project now. We have the very popular 2013 movie "Gravity". This is a "space" themed movie. Let's check out movies that are most similar to the plot of this movie.



The sentence encoder mapped space themed movies closer together in the high dimensional space and hence the visually appealing result.

Next, we have a popular Disney movie “Beaty and the Beast”.

Product Image



For related movies, we’ve got more Disney movies and other children’s movies.

## MODULE AND DATA LIMITATIONS

This module was trained using high quality data (the SNLI Corpus). If the quality of our input data is poor, then the module may not function in a desirable manner. Sentences without proper structure or poor grammatical constructs could play a part in the module performing poorly.

The sentence encoder has two versions –

i) The transformer encoder version responsible for very high accuracy but with higher computational cost.

ii) The Deep Averaging Network (DAN) model which requires significantly lesser compute power but comes at the price of some compromised accuracy.

Organizations will have to consider the pros and cons of both these models.

While the main goal of this project was to try and propose a decent recommendation system with no training from our side, we would want “enough” data to see desirable results. As is the case for any machine learning model, more data (quality data) will yield better results.

## EVALUATING THE MODULE

For an everyday machine learning model, metrics like accuracy, squared loss, ROC curves and Area under the curve (AUC) plots can be leveraged to determine the efficiency of a model.

For our model, one of the most practical ways to evaluate performance would be A/B testing. Let us try and understand with an example. We have a retail company, “XYZ”. They have a website that customers could log into and shop online. For some users, they could display recommendations using the existing model (the control group). For another group of users, they could have recommendations displayed using the new model. Selection of users must be done in a random manner.

Now, they would have to capture the click through rate for both cases. They could carry out this “experiment” for a month or a few weeks. Next, they measure the average click through rates for both groups of users. Let us assume that the mean was indeed better using the new module. Then, they would have to conduct a hypothesis between the two groups of users. If the hypothesis tests suggest statistical significance, they could consider using the new model. Statistical significance in this case would suggest that the new module is indeed performing better.

If no statistical significance was observed, it may still be worthwhile considering productionizing this module. The current module was trained using limited data and if their range of products evolve over time, they would to consider training another model with fresh data.

## PRODUCTIONIZING THE MODULE

If the module performs well, then an organization would be interested in productionizing this model. We need to consider how this will work in real time using currently available data. Given a product description, our code will first need to look-up the index of that product. Next, from the cosine similarity matrix, we need to find “n” most products that are relevant to the product in query (or product that is currently being viewed by the customer).

Once we have these relevant products, we can have these products displayed below the main product where these could attract a customer’s eye.

# CONCLUSIONS AND FUTURE SCOPE

Deep learning/machine learning models perform better when they are trained using high quality data. Often, for some organizations, such data isn't easily available. Labeled data also isn't easily available sometimes.

Transfer learning could be a power tool that organizations can leverage to counter data scarcity. Even if we need to train a model using pre-trained transfer learning models, we would obtain significantly better results than training our model from scratch. (Using the limited resources that we may have)

For this project, we considered Amazon products around 200,000 products (from three separate categories) and almost 20,000 movies. The next question is: How practical is this tool for other practical purposes? Perhaps this can be used for something like Airbnb. Given the descriptions of a listing, we could recommend similar listings in the area. Or this could be used to recommend similar books and websites to users.