Building a Benchmarking Tool for E-Commerce

Final Project for Predictive Methods, Information Reduction Methods, and Machine Learning and Statistical Learning

Francesco Nardin, Josue Velasco, and Santiago Villamil

Aix - Marseille School of Economics
January 2024

Contents

1 Introduction

The main core of our project is to develop an image classification algorithm with the specific purpose of identifying best seller design topics by exploiting data coming from the online shopping platforms like Etsy[1].

The idea behind is that our approach to product classification and recognition can save a lot of time for online retailers who might usually have to go through a process of manually checking online trends and social media campaigns in order to have an idea of current product demand. Often, this process does not quantify exactly the presence of particular products on online markets. Our project aims to move ahead on the direction of recognizing the interests in products of customers and engage better the audiences set by retailers.

Our approach can provide more information to sellers, therefore reducing information asymmetry between market agents and improving market efficiency by signaling to suppliers the higher demand for a specific type of product and saving time and money to customers by increasing product supply and competitiveness, and therefore, profitability.

In our case we will be focusing on season trendy products for the case of Halloween, as it allows us to pin down our data set better as well as being one of the most profitable seasons to sale related products[20], especially in markets like the United States, where Etsy has a large presence but at the same time, it is not as popular or competitive as other marketplaces like Amazon. Our project offers a potential real world application case that could be interesting for online sellers to increase their margins by offering more appealing products, since, as a whole, we offer a benchmark tool to evaluate the current product offer and demand in the market allowing them to adapt their stocks accordingly to improve revenue. This tool would check the most popular designs, including their price ranges, number of comments (reflecting competitors' sales, as direct access is unavailable. The logic behind is that more comments typically indicate more sales, as reviews are generally left after purchasing the product), shipping costs, etc. By not just relying on product descriptions, the seller would have more information to make a better informed decision, specially since our tool offers a prediction on sales after considering all of these parameters.

The structure of our data extraction also allowed us to get more information about the products in an organized and consistent way. By being able to scrape the site, we ensure to have the latest available market information, which offers an opportunity to sellers to update their products in time for holidays sales.

2 Materials and Methods

In this section, we outline the materials used and the methodology employed in our project. The implementation of our Convolutional Neural Network (CNN) for image classification required careful consideration of data collection, categorization, and model training. Two distinct datasets were utilized, each serving a specific purpose in the development and evaluation of our predictive model.

2.1 Data Collection

The foundation of our project lies in the acquisition and preparation of two essential datasets. The first dataset, dedicated to CNN training, encompasses images distributed across six distinct categories. The second dataset, crucial for testing the CNN and creating the sales prediction model, is sourced directly from Etsy. The comprehensive nature of these datasets ensures a robust and informed approach to our CNN training and subsequent sales prediction modeling.

- **CNN training**: This dataset is composed of 13,650 images divided into six categories. These categories are: cats, clowns, ghosts, pumpkins, skeletons and witches.

- **Test and prediction**: This dataset comes from Etsy and it contains the images of the products as well as information of the products. The images are useful for the testing of the CNN, while the other features are important for the prediction of sales. This dataset contains the product description, its price, the type of product as sourced from the scraping query, the amount of reviews, and the file path location to facilitate image classification.

2.2 Web scraping images for the training

Web scraping is a pretty fast way to collect information from the web pages, since it allows an automated search for information. However, it is a borderline practice and many websites have some form of protection to avoid massive practice of web scraping. Given this delicate environment, we searched for websites that provided copyright-free images. Some websites also requested the registration to their platform and the creation of specific API keys. These keys act as a form of digital permission, allowing the script to make requests to the website's server. Some websites provide keys for enhanced security and control over access. As a matter of fact, the keys allowed to download for free a determined amount of images, as a rightful form of protection. Given all these limitations, we downloaded all the images we could from different websites: Pexels and Vecteezy. The API keys were needed only for the Pexel web scraping activity, while on Vecteezy, the data was freely web-scrapable.

The downloaded images were saved into separated folders in order to have the data already divided for training the CNN.

2.3 Data cleaning

The images downloaded thanks to the automated web scraping, given the selected keywords, were not always correct or useful for our purpose. As a matter of fact, the web scraping makes it faster to download but reduce the control over the actual results. For this reason, we then performed a manual data cleaning process to exclude images that were not coherent or good for our purpose.

2.4 Colors reduction

The images downloaded from these websites, especially from Pexels, are pictures with high definition. In our specific case, to train correctly the model, given the images of the train set and the form of the CNN, we decided to reduce the number of available colors in each image. This process is particularly important, because it simplifies the images by giving importance only to the most used colors. For this task we decided to use the K-means method. The K-means algorithm searches for

a pre-determined number of clusters within an unlabeled multidimensional dataset. The "cluster center" is the arithmetic mean of all the points belonging to the cluster and each point is closer to its own cluster center than to other cluster centers. The K-means is based on the Expectation–maximization (E–M) algorithm. In short, the expectation–maximization approach consists of the following procedure:

- Guess some cluster centers
- Repeat until converged:
  - E-Step: assign points to the nearest cluster center
  - M-Step: set the cluster centers to the mean

Here the "E-step" or "Expectation step" is so-named because it involves updating our expectation of which cluster each point belongs to. The "M-step" or "Maximization step" is so-named because it involves maximizing some fitness function that defines the location of the cluster centers which is accomplished by taking a simple mean of the data in each cluster.

We set the number of colors to 10, as the main idea is that each pixel is recolored to the centering color for its cluster. The results are really interesting. In fact, the images are still well recognizable even with little use of colors. What is more, the shapes are more recognizable and defined. As it is possible to see in **Figure 1**, on the left we have the original picture and, on the right, the one reduced in colors. The picture is still clear in its appearance, but the number of colors now present is just ten.

Figure 1: Color reduction with K-means

2.5 Data augmentation

In order to effectively train the model, the size of the dataset needs to be big enough. However, the limitation imposed by the API keys led to a dataset not enough extensive. For this reason, we decided to perform data augmentation. Data augmentation is a technique to artificially expand a dataset by applying various transformations to existing training examples. These transformations

include rotation, scaling, flipping, and cropping of images. The goal of this technique is to expose the model to a broader range of scenarios and making it capable of comprehending them.

As it is possible to see from the example in **Figure 2**, the simple image of the ghost on the left has been flipped and rotated several times. The empty spaces have been filled with the closest colors, in this case black.

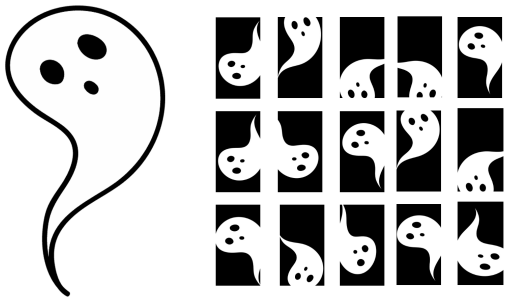


Figure 2: Augmented Ghosts

2.6 Web scraping images and data for the testing

Also the data for the test set of the CNN and for the predictive task were web scraped, this time from the Etsy website. The data scraped were of several type and needed to be properly stored and named to maintain the match between the different parts. An automatic link creation code was run in order to be able to extract the information automatically by just setting the type of product and desired price range. The layout of the website is a standard retail website with paginated structure. We constructed our dataset by taking advantage of the BeautifulSoup Python library. This library allows us to extract the whole HTML content of the websites and select the interesting sections for our purpose.

We identified the structure of the code for each displayed product and created an unique delimiter between each of them, therefore being sure that the scraped information was truly matching for each product and avoiding mixing. Data about prices, number of reviews, and product type was obtained following this method. To make it easier to eventually process and classify the images and matching them back again with their respective product information, it was necessary to name and store them in a way that facilitated classification and manual validation. All images coming from the same scraping query were stored on a dedicated folder, named with the product type and price range. The individual images were named with a truncated and clean version of the product description. Some additional processing was needed as the presence of some characters is not allowed for files, as they are used as location keys on file paths, (for example "/".)

One interesting insight about the online retailer web scraping is the sensibility of the website to the setting of the user-agent details. This parameter is used when requesting the html from the website and contains information about the type of device and browser from which the requests are being performed. Retail stores like Etsy can use this information to display content that they consider to be better given the profile of the customer. The manipulation of this parameter can also

allow to access the market of a particular country and therefore the language of the descriptions and used currency. Setting this parameter as desired can allow for further data comparisons and could also be very interesting for retailers online that operate in different countries. It is also useful for retailers who want to target a given niche based on customer profiling and on their device type.

3 Model's architecture

3.1 Basics of Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of Deep Learning neural network architecture commonly used in Computer Vision. They are designed to automatically and adaptively learn spatial hierarchies of features from tasks with grid-like topology, such as images[9].

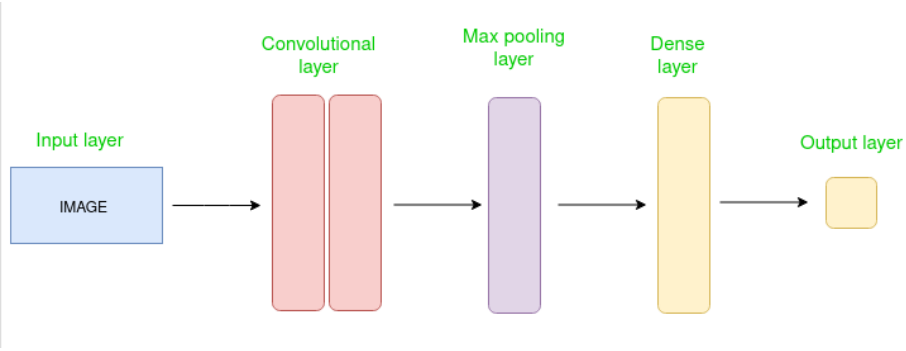


Figure 3: Basic architecture of a CNN

1. **Input Layer**: This is where we feed our data into the model. The number of neurons in this layer is equal to the total number of features in our data.
2. **Convolutional Layer**: This layer applies filters to the input image to extract features[8].
3. **Pooling Layer**: Downsamples the image to reduce computation.
4. **Fully Connected Layer**: Makes the final prediction.
5. **Output Layer**: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

The data is fed into the model and the output from each layer obtained from the above step is called feedforward. Then the error is calculated using a loss function. After that, there is the backpropagation through the model, which consists on calculating the derivatives, in order to adjust the weights and increase the precision of the model itself.

We drafted a first basic model composed of 6 layers that gave us a performance of 14.61% of accuracy on the validation set. Since this result was too low, we wanted to try with other common (and more efficient) architectures to see if our predictions improved.

3.2 Testing different architectures to compare performance

Our starting point was the traditional architecture of typical Convolutional Neural Networks, composed of 6 layers for training and one fully connected layer that classifies the images into 6 classes: witch, skeleton, ghost, clown, cat and pumpkin. The approximate maximum accuracy with this architecture was 15%. Since this result was really bad, being practically random classification, we decided to experiment with other architectures, since adding more layers didn't improve the accuracy.

3.2.1 ResNet-18

One of the most common used architectures is the ResNet-18 (short for Residual Networks), it is the most basic structure of the ResNet family (ResNet-34 and ResNet-50). The 18 indicates the number of layers used for training. The key idea of this architecture is the use of residual functions which help to solve the problem of vanishing/exploding gradients, which is a common issue in deep learning. It uses identity blocks which take advantage of a technique called skip connections, which connect the activations of a layer to further layers by skipping some layers in between[7]. This forms a residual block, and ResNets are made by stacking these residual blocks together. This encourages the model to focus on learning residuals or changes to the features, making optimization more efficient.

With this idea in mind, we constructed our model to see its performance on our data. We started with a convolutional layer followed by batch normalization and max pooling. Then, several identity blocks were stacked together. Each block consists of three convolutional layers, batch normalization, and skip connections with adjustments to ensure compatibility. The model ends with average pooling, flattening, and fully connected layers for classification. We also tried to introduce some modifications to ensure robustness of the model and potentially improving its performance. Some of the modifications were the introduction of more dropout layers applied after the convolutional layers within the identity blocks, in order to prevent overfitting by introducing some randomness. We also added dense layers after the average pooling, with batch normalization and dropout. These layers contribute to the model's final classification by providing more capacity for learning complex patterns, which resulted in a maximum accuracy on the validation set of 24.5%, leaving room for improvement.

3.2.2 DenseNet

We tried an architecture called Densely Connected Convolutional Networks (DenseNet)[19] which connects each layer to every other layer. We thought that this configuration could allow the model to recognize better the patterns in the data and generalize better on new, unseen data since it acts as a whole, giving it a general picture, instead on focusing on individual images, potentially improving the performance, since this dense connectivity pattern allows the network to learn more effectively by reusing features, hence reducing the number of parameters and enhancing the gradient flow during training. [5]

In DenseNet, each layer receives the feature maps of all preceding layers as input. The multiple inputs are concatenated into a single tensor. This architecture is designed to solve the vanishing gradient problem that arises in deep CNNs, where certain information can 'vanish' or get lost due to the long distance between input and output layers[11].

DenseNet is divided into DenseBlocks, where the dimensions of the feature maps remain constant within a block, but the number of filters between them is changed. The layers between the blocks are called Transition Layers which reduce the number of channels to half of that of the existing channels.

This architecture, returned a maximum validation accuracy of 23.6%, around the same as the previous one.

3.2.3 GoogLeNet

Proposed by researchers at Google (alongside various universities), this architecture is 22 layers deep and introduces several innovative features like 1x1 convolutions and inception modules[14]. After training, the model's performance on a validation set, we got an accuracy of up to 35%, even with the same small data sample used to test other architectures. Since this was the best performing model so far, we decided to explore it further and see if we could get even better results.

3.2.4 Ensemble models

Ensemble models in machine learning are a type of model that combines the predictions from several base estimators to improve generalizability and robustness over a single estimator[13]. The main idea behind ensemble learning is to reduce the variance and bias in the predictions by combining multiple models that are trained on different subsets of the data or using different algorithms. This way, the ensemble model can capture different aspects of the data and mitigate the weaknesses of individual models, resulting in more robust and accurate predictions[6]. Generally, ensembles have higher predictive accuracy.

One of the most popular ensemble learning algorithms is *Bagging* (short for Bootstrap Aggregating)[15]. It involves training multiple independent models on random subsets of the training data. The predictions of these individual models are then aggregated to produce a final prediction.

The main idea behind bagging is to decrease variance without increasing bias and prevent overfitting[18], potentially improving accuracy. This is achieved by combining the benefits of bootstrapping and aggregation to yield a stable model and improve the prediction performance. Bagging allows a more diverse set of models by creating larger subsets of the original data.

With this idea in mind, we decided to train different instances of our traditional model (since it will always served us as starting point to test performance before exploring other options), with different transformations applied on our dataset at the same time that we shuffled them, so the model could learn different patterns and potentially generalize better. It did improved compared to the traditional architecture, giving us an accuracy test of around 27% (almost as double as the original approach). However, this approach didn't return a higher accuracy as expected or as good the GoogLeNet architecture described above. Therefore, we decided to use this technique too on that model to see if it could improve, but it gave us an accuracy of 25%, even worse that the ensemble method using the traditional architecture. Before moving forward with different solutions, we considered applying different transformations on each batch of the images used during training to see if adding more diversity to the model could make the algorithm to learn more patterns and make it more robust to variations. Nevertheless, the maximum accuracy on the validation set this approach gave us was 26%, a bit better compared to the previous one, but not by much and still

not a good result. Therefore, we decided to try a different path and use more architectures but this time, using *transfer learning*.

3.3 Using pre-trained weights as auxiliaries

In the lasts years, several different architectures have been developed to train Convolutional Neural Networks for image classification tasks in order to increase accuracy and reduce error rate, as well as resources and energy consumption. With this idea in mind, we constructed simplified versions of different architectures using transfer learning with a small data set sample to find the best performing one.

The process begins with loading the pre-trained weights into our Python environment. These weights, typically trained on a large dataset like ImageNet, among others, that serve as an excellent starting point for the model. The next step involves modifying the output layer of the pre-trained model, which is designed for the original task, to match the number of classes in the new task, in our case, just 6 classes (witch, skeleton, clown, cat, ghost and pumpkin).

Once the model is set up, the fine-tuning process begins. During this phase, the pre-trained weights are updated to better fit the new data.

The use of pre-trained weights for fine-tuning offers several advantages. Firstly, it allows for the transfer of learned features from the pre-training task to the new task, which can lead to improved performance. This is particularly beneficial when the new task has limited training data like the one we used for testing the capacity of the model but also for the final model since it is sometimes difficult to get training images with the required characteristics to be high-quality data and in big volumes.

Secondly, it reduces the computational resources required for training. Since the model is not trained from scratch, the training process is faster and requires less computational power.

Lastly, the use of pre-trained weights can help mitigate common challenges in training deep neural networks, such as overfitting and the vanishing gradient problem. By starting with weights that already provide a good initialization, the model can converge faster during training.

This approach proved to be a powerful technique that offers several advantages. It simplifies the training process, improves model performance, and reduces computational requirements.

Some of the architectures we tested are:

- **VGG-16 (Visual Geometry Group)**: a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The "deep" refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers, respectively.[21] In our case, we used the model with 16 layers. The way it works is by replacing the large kernel-sized filters with several 3×3 kernel-sized filters one after the other. It was trained over more than 14 million images belonging to almost 1,000 classes; we took advatage of this and imported the pre-trained weights and adapt it to our specific use case by fine-tunning the model so it could properly assigned our images in the 6 classes, which gave us a maximum validation accuracy of around 24%

- **EfficientNet B7:** It is part of the EfficientNet family. Their key features are that they are able to scale up the baseline network in a balanced way, considering the depth, width, and resolution of the network. This scaling method is based on a compound coefficient that uniformly scales all dimensions[17]. B7 is the largest model in the family and has achieved higher performance while being 8.4 times smaller and 6.1 times faster than previous best CNN models. Once again we imported pre-trained weights and adapt it to our specific use case, resulting in an approximate maximum validation accuracy of 25%

- **MobileNet:** The idea behind this type of architecture is that they are based on a streamlined architecture that uses depth-wise separable convolutions to build lightweight deep neural networks. MobileNets introduce two simple global hyperparameters that efficiently trade off between latency and accuracy[2]. In our use case, building on top of this model gave us an approximate maximum accuracy on the validation set of 22.64%

- **ResNet50:** It stands for Residual Network and is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer).[4] The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a "bottleneck", which reduces the number of parameters and matrix multiplications, this enables a much faster training of each layer. With this approach, we ended up with an approximate maximum validation accuracy of 25.27%

- **GoogLeNet:** Due to the fact that, so far, the simple construction of this architecture gave us the best result, we decided to explore too the option of using transfer learning, hoping it could get better with the already pre-learned weights. After training it, the model's performance on the validation set gave us an accuracy of up to 68%! Even with the same small data sample used to test other architectures, being the best solution so far. However, we couldn't fully implement this approach due to technical difficulties, such as the model taking too long to train on the whole training set composed of over 13 thousand images making it way too heavy for the remote GPU usage hosted by Google, arising difficulties in reproducing the results; therefore, pushing us to use the model without learning transfer, ending up with a maximum validation accuracy of around 80% making it reliable for predictions. However, since the nature of our test images (Etsy products), contain other elements such as models or mock-ups, it was a little challenging for our model to replicate this accuracy, nevertheless, it makes it a good first approach to build on top of for future versions of the tool.

After running and testing these architectures, we found out that they performed similar since they all gave a validation accuracy of around 25%.

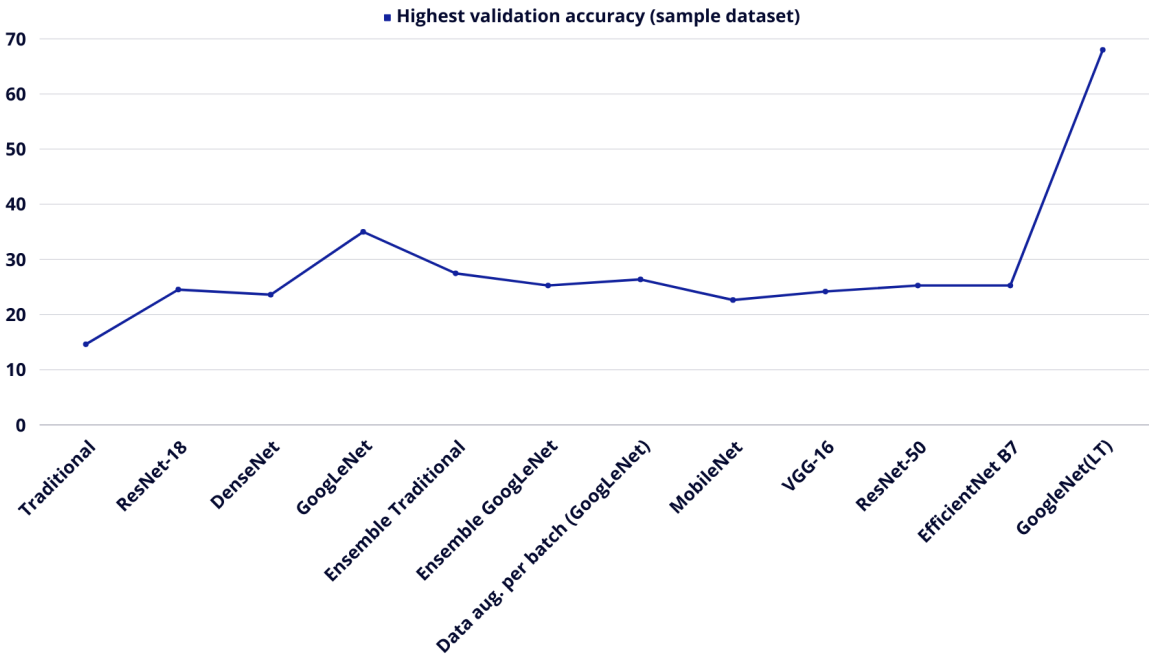


Figure 4: Performance of the architectures

We even tried to apply different techniques to avoid over-fitting in case that this was the cause of the low performance (because some models did good on the training set but not so much in the validation one). Some of the approaches we took to avoid it were:

- Data augmentation: as mentioned before, we artificially increased the size of our dataset by performing different transformations on the original data set like rotating the images, zooming in, shearing, etc., in order for the model to have more training data to look for patterns in the images so it could generalize better and learn how to categorize better on new, unseen data (the validation set)

- Dropout rate: we used this parameter to randomly omit (set their values to zero) some neurons during training.

- Early stopping: this was one of the most important methods for us because it stopped the training of our model after 5 epochs (since we established the '*patience*' at 5 which is the one responsible for checking if the validation loss decreases. Otherwise, it stops the model and restores the best weights (even if they were at a previous epoch before the last one). This way, we don't overfit our model and ensure a good generalization on new, out of sample data.

- Bigger data set: besides increasing artificially our data, we also wanted to have an originally big enough and high-quality data so the model could learn the general patterns and forms of

each type of category for our classification task.

3.4 GoogLeNet: The best performing architecture

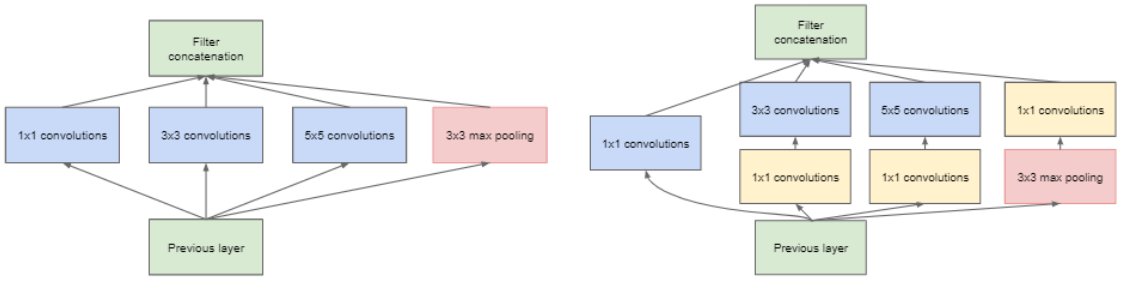
As mentioned before, this was our best performing architecture so far (even with a small data set sample), and the reason behind this, is that this architecture introduces new innovations such as a 1x1 convolutions in the middle of the architecture in order to reduce the number of parameters (weights and biases) and it allows at the same time to increase the depth of the architecture.

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Figure 5: GoogLeNet performance vs other architectures (ILSVRC 2014 image classification challenge)

Another key feature is the use of *Global Average Pooling* which is a fully connected layer used at the end of the network that takes a feature map of 7x7 and averages it to 1x1 which decreases the number of trainable parameters to 0 and improves the top-1 accuracy by 0.6%

One of the breaking-through innovations of this architecture is the introduction of ***inception modules*** of 1x1, 3x3, 5x5 convolution and 3x3 max pooling that perform in a parallel way at the input to stack them together and generate a final output. By doing this, the convolution filters can handle better objects at multiple scales and different sizes better than other architectures.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 6: Classical inception modules vs GoogLeNet's inception modules

Finally, GoogLeNet uses some intermediate classifier branches in the middle of the architecture during training that serve as auxiliary classifiers. These branches consist of a *5x5 average pooling layer* with a stride of 3, a *1x1 convolutions* with *128 filters*, *a fully connected layer of 1,025 outputs*, a *ReLU activation function*, a *dropout regularization* with a ratio of *0.7* and a *softmax classification layer* with *1,000 classes output*. The generated loss of these layers added to *total loss* with a weight of *0.3*[10]. These layers help in combating gradient vanishing problem and also provide regularization.

The resulting architecture is a model of 22 layers deep. The architecture was designed to keep computational efficiency in mind. The idea behind is that the architecture can be run on individual devices even with low computational resources.

3.5 Inception V1 vs Inception V3

As previously explained, we used an imported library in Python called Inception, which is part of the GoogLeNet architecture. The Inception models, come in different versions denoted as V1, V2, and V3. The first version (V1), was the original design of the architecture which proved to be a big jump in performance and efficiency over traditional or previous architectures, but over the years, new changes and improvements have been done on the first proposal, being V3 the latest and best version[12], which is the one that we used for our use-case. Some key differences between InceptionV1 and InceptionV3 are:

1. **Architectural Changes:**

InceptionV1 introduced the concept of inception modules, which are blocks containing parallel convolutional layers of different filter sizes while *InceptionV3* is a later version that refines the original architecture. It includes factorized convolutions and additional optimizations to improve training efficiency.

2. **Factorized Convolutions:**

InceptionV3 uses factorized convolutions, which break down a convolutional layer into smaller convolutions. This helps to reduce the number of parameters and computation while preserving

representational power.

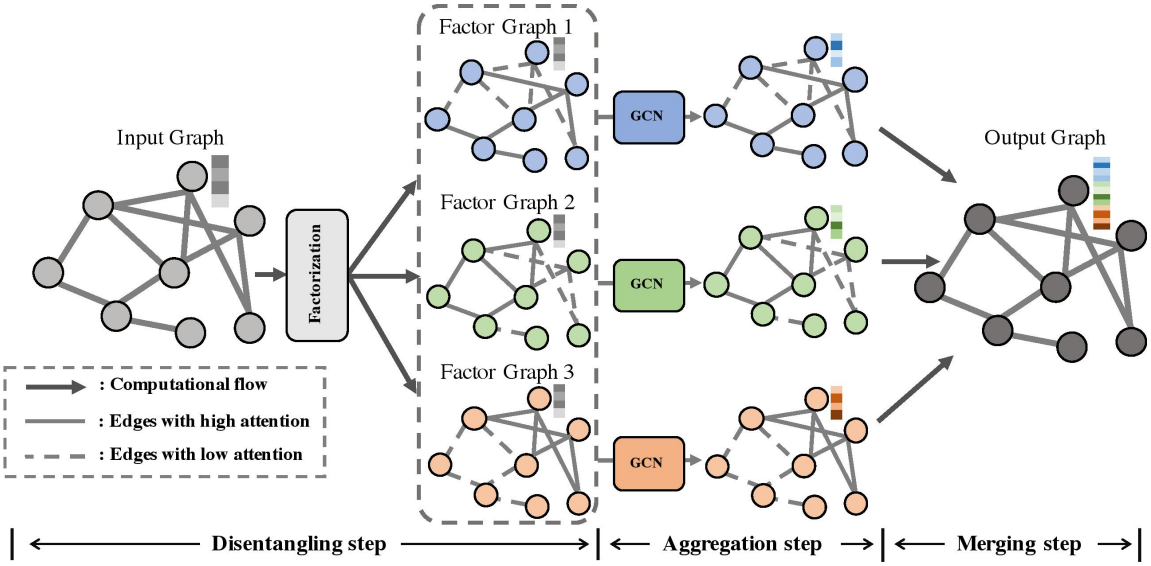


Figure 7: Factorized convolutions

3. **Batch Normalization:**

InceptionV3 incorporates batch normalization, which helps stabilize and accelerate training by normalizing the inputs of each layer.

4. **Reduction Blocks:**

InceptionV3 includes reduction blocks that incorporate strides in the convolutional layers to reduce spatial dimensions more efficiently.

5. **Auxiliary Classifiers:**

Both architectures use auxiliary classifiers during training to mitigate the vanishing gradient problem. However, the design and placement of these classifiers differ.

6. **Overall Performance:**

InceptionV3 generally achieves better performance and accuracy compared to InceptionV1, especially in terms of training efficiency and generalization to diverse datasets.

In summary, InceptionV3 is an evolution of InceptionV1, incorporating improvements in architectural design, optimization techniques, and the use of factorized convolutions.

3.6 Understanding what the model sees: using PCA

Applying PCA (Principal Component Analysis) to the intermediate features of the neural network can provide insights into the patterns or features the model is using for training. PCA is a dimensionality reduction technique that transforms the original features into a set of linearly uncorrelated variables called principal components. These components capture the most significant variations in the data.

We used it as follows:

1. **Loading the pre-trained model**: Which we saved in an h5 file that contains all the weights learned by the model on the training dataset that we divided into our needed categories.

2. **Preparing the data:** We extracted the paths to the images from each folder and saved them in a data frame alongside the category to which image belongs to, so we could split it more easily into training and validation sets.

3. **Creating a feature extraction model:** In order to extract the features that the model uses for predicting, we needed to create a new model that outputs these features just before the final dense layer. This is done by creating a new model instance that takes the same input as the pre-trained model but outputs the layer before the final dense layer which is the one in charge of the actual categorization, so we could get the features that the algorithm uses for categorizing.

4. **Extracting features from the images:** we used the feature extraction model previously created in order to extract features from each batch of images.

5. **Applying PCA to the features:** Finally, we used PCA to the extracted features to reduce their dimensionality so we can plot them and understand a bit better how the model is classifying the images. The number of principal components is set to 3, which allows the transformed features to be visualized in a 3D scatter plot. Each point in the plot corresponds to an image, and the points are colored according to their category.

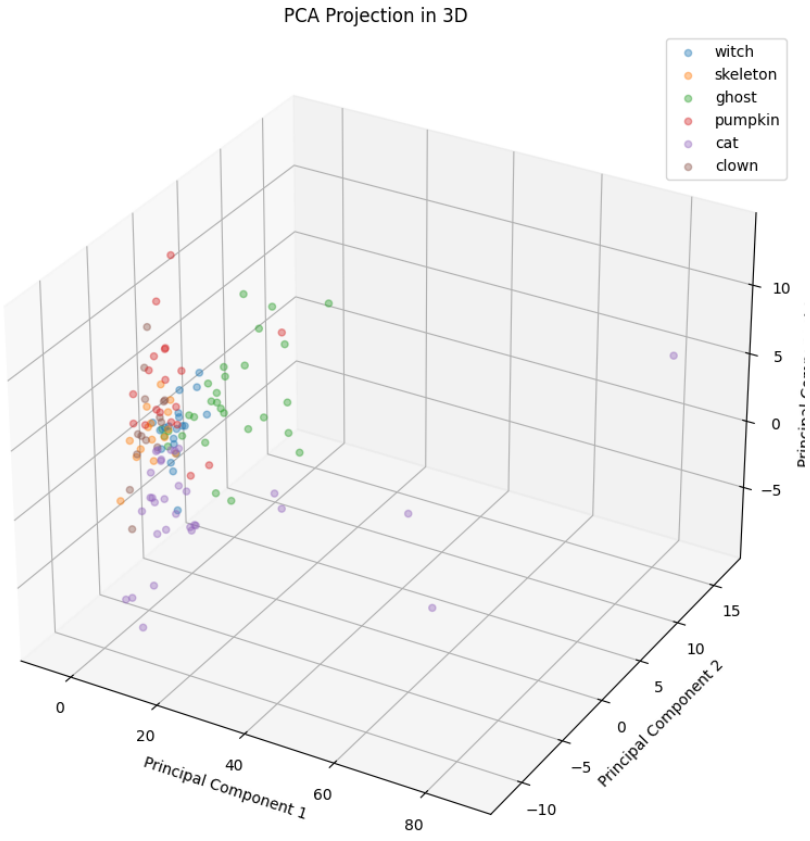


Figure 8: Feature Exctraction with GoogLeNet

After applying PCA, we can visualize the principal components to gain insights into the most important features that contribute to the variance in the data. This visualization can help us interpret the learned representations in a more understandable way. For example, we can see that the model makes better distinctions between ghosts and cats, which would result in a better categorization between these two classes, however, it struggles a little bit to differentiate between skeletons, clowns and witches (which is indeed reflected in our test set where some images that are clearly witches are categorized as skeletons and vice versa), maybe due to human-like features.

This could be a useful technique to understand what is going on with our model so in future iterations, we can fine-tune them like adding more training images that make clearer distinctions between skeletons and witches in order to get more accurate predictions.

4 Prediction Task

After training and testing the model for classification of images, we wanted to apply directly the results of the CNN to try to forecast the sales of a product given some characteristics. For this reason, the pieces of information from the Etsy website where not just related to the image, but also the price and the type of product were scraped.

This dataset, regarding the product from Etsy, was merged with the prediction of the CNN in order to obtain a new feature useful for the prediction of sales.

4.1 Exploratory Data Analysis

Before diving into the creation of the prediction models, we started with a preliminary data analysis. The scope of this analysis is to get to know the distribution of the data before creating the model. Some important information can also be used in the business evaluation of this project, especially for the identification of the most trending images in the season.

Starting from the results of the CNN we displayed the most frequent categories of images that appeared in our data set. It is interesting to note that in general the most used design is skeleton with 28% of the total images, followed by witches and ghosts. However, if we consider only the category of shirts and sweaters, some of the most interesting for the business application of our product, the top three changes a little. As a matter of fact, for shirts it is composed of ghosts, witch and pumpkin; while for the sweater it is made of skeletons, witch and ghosts. As it is possible to see in **Figure 9**.

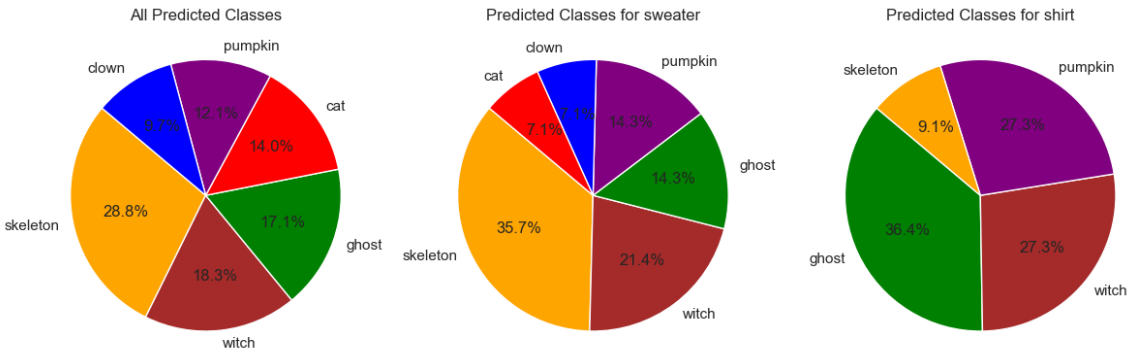


Figure 9: Predicted graphics for the whole data, for sweater and shirts

Let's now have a deeper look into our variable of interest. In fact, we are going to use the variable Review-Number as a proxy of sales. For this reason, it is important to see how it is distributed.

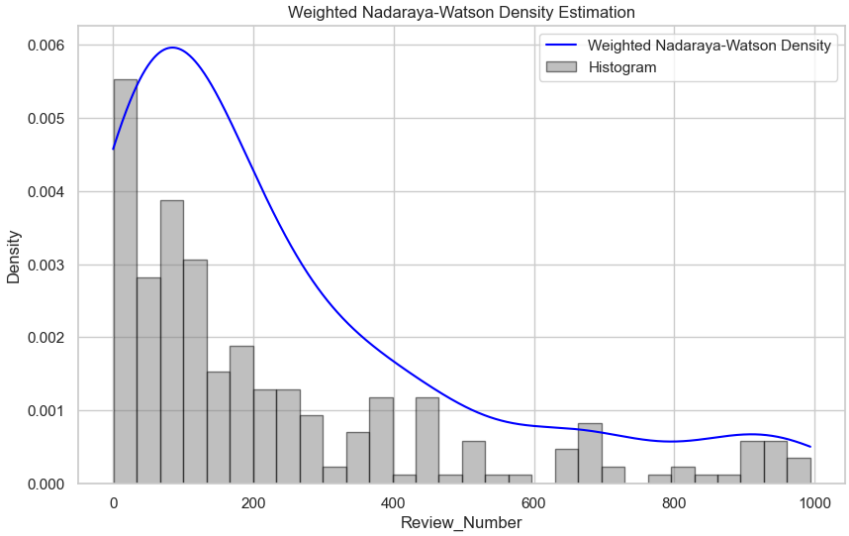


Figure 10: Number of Reviews and their estimated distribution

From this graph, we can easily see that most of the products have less than 200 reviews (first quartil of the dataset), while only some products surpass the median (500 comments), making our distribution skewed to the right. This is a clear sign that there are many products available in the e-commerce platform Etsy but only some of them are considered best sellers, others have a little number of sold items, and this is may be caused by many different reasons.

Looking at the amount of comments for each product category, we can observe that the most interesting items in the platform are ties, outfits and dresses (**Figure 11**). This can be related to the fact that in Etsy, products are usually more fun than in some other platforms. This is an interesting thing to keep in mind, especially when a company is looking for a new digital platform where to sell its own products. In fact, if the products are too serious for the platform, they may not perform as well as others.

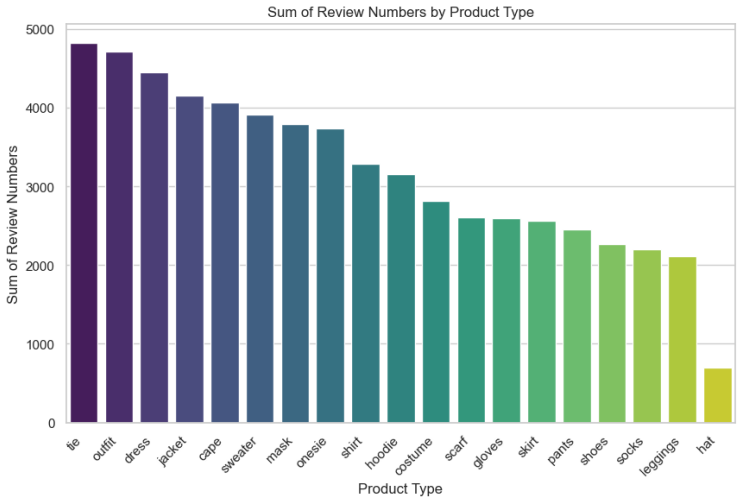


Figure 11: Number of Reviews by Product Type

It is also possible to observe that prices are evenly distributed between the categories of products available on the platform. This is quite interesting given that the median of the prices is for all products below 100$ but there are some really expensive ones in categories like jacket and dresses that may underline that some of the products have a much higher value than the vast majority of the others. See **Figure 12**.

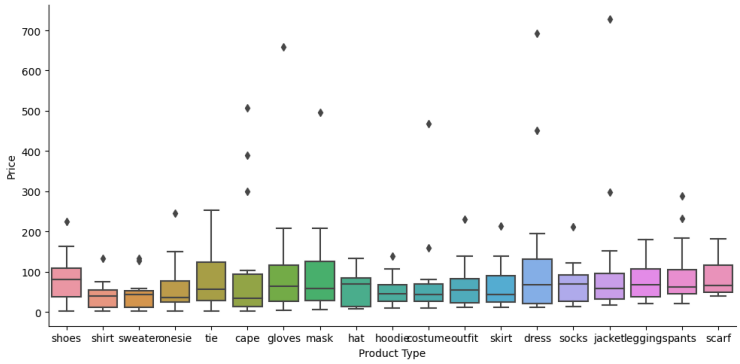


Figure 12: Box Plot of the Product Price by Type

4.2 Predictive Analysis of Sales using Random Forest

For the sake of evaluating how relevant our product topic predictions were to predict sales as proxied by the number of reviews, we performed a *Random Forest Regression (RFR)* over a training set with 80% of our scraped product dataset. The dataset processing to perform our random forest regression required transforming are categorical variables into dummies. This was done because our categorical features are not ordinal in any way, therefore not allowing as to use the *label encoder* command. For our model, the number of reviews is our variable of interest. the resulting explanatory variables, after the one hot encoding, were divided as follows: 20 variables describing the clothing category, 6 variables describing the printing on the product and the price.

The reasons to have chosen a random forest regression as method are multiple, namely the following advantages:

- High Predictive Accuracy: Random Forest tends to provide high accuracy in prediction tasks. It can capture complex relationships between features and the target variable, making it effective for sales prediction where multiple factors can influence outcomes.
- Non-Linearity Handling: Random Forest can capture non-linear relationships between input features and sales. This is crucial as sales data often exhibits non-linear patterns that may not be well-modeled by linear regression.
- Robust to Overfitting: The ensemble nature of Random Forest, combining multiple decision trees, helps in reducing overfitting. By aggregating the predictions of multiple trees, it minimizes the risk of fitting the model too closely to the training data.
- Variable Importance: Random Forest provides a feature importance score, allowing to identify the most influential features for sales prediction. This can provide valuable insights into which factors are driving sales.
- No Need for Feature Scaling: Random Forest is not sensitive to the scale of input features, eliminating the need for feature scaling. This simplifies the preprocessing steps in comparison to some other regression algorithms.
- Interpretability: While individual decision trees can be complex, the ensemble nature of Random Forest allows for a degree of interpretability. Feature importance scores can be used to understand the relative contribution of each feature to the predictions.

To implement the random forest regression, we first splitted the data into train and test sets, leaving 20% of the observations for the test. Next, we tuned the model hyperparameters using RandomizedSearchCV. This methodology allows us to test different hyperparameter combinations to find the best setting. We considered the following hyperparameter combinations:

- $n\_estimators$: The number of trees in the forest.
- $max\_features$: The number of features to consider at each split.
- max_depth: The maximum depth of the trees.
- $min\_samples\_split$: The minimum number of samples required to split a node.

- *min_samples_leaf*: The minimum number of samples required at each leaf node.
- *bootstrap*: A binary parameter indicating whether bootstrap samples are used when building trees.

The model that was found to be the best using this methodology was the following:

Best Model Hyperparameters:

- `n_estimators:` 400
- `min_samples_split:` 2
- `min_samples_leaf:` 4
- `max_features:` 'sqrt'
- `max_depth:` 10
- `bootstrap:` True
- `Best Negative RMSE:` -65628.17

The results are interesting and the choice of some of the hyperparameters makes sense for the type of data that we are dealing with. For example, having a *min_samples_split* value of 2 means that our regression can be really sensitive to noise. Splits would potentially change with the presence of one more or one less observation.

Although this is the case of the splitting, the obtained *min_samples_leaf* value of 4 can be thought as balancing the sensitivity to changes in the data, by counteracting over-fitting. Having a maximum tree depth of 10 also makes sense considering the relatively small amount of data, by also avoiding over-fitting. Further robustness of the RFR was achieved by setting the bootsrapping hyperparameter to *True*.

We can visualize the prediction performance on our testing sample with a visualization of the predicted vs the actual values. The result was the following:

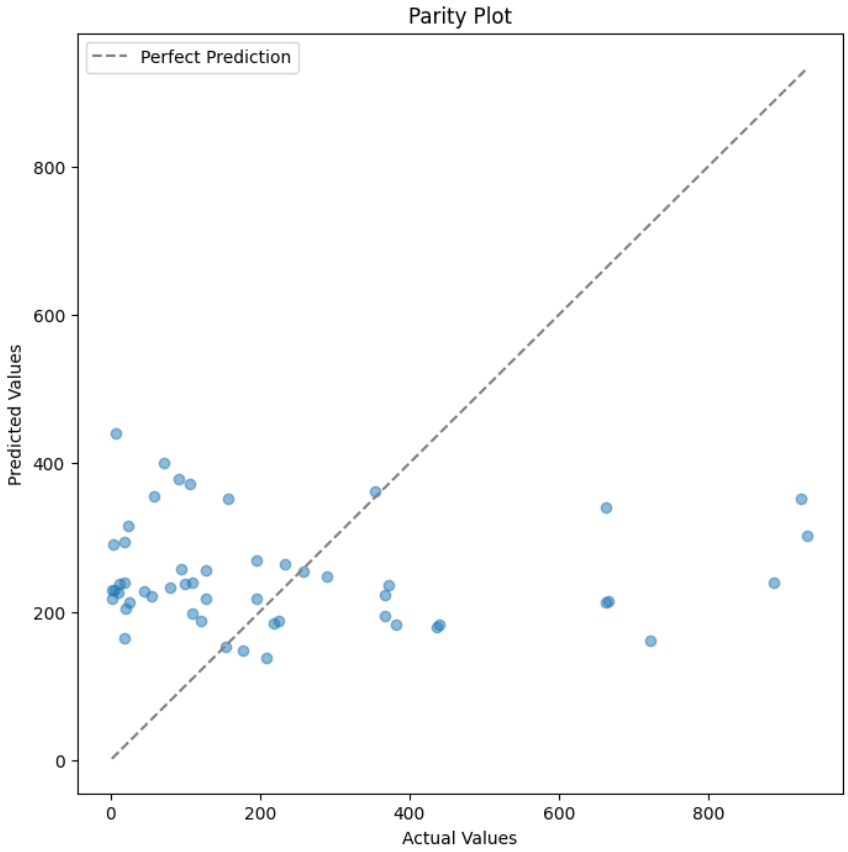


Figure 13: Parity Plot of the Random Forest Regression

The plot makes it clear that the random forest has not been able to provide a good prediction based on our features, and that is very understandable. The amount of data used for the training of the model was may insufficient in size and also other features could have had played and important role in the prediction of sales, however, this feature may be missing from our data. This is even easier to grasp when we take a look at the feature importance plotted for our model:

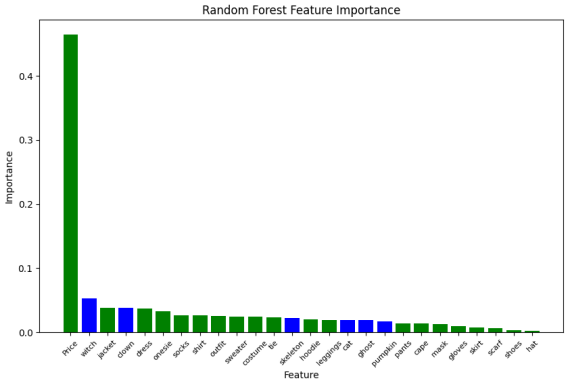


Figure 14: Disaggregated Feature Importance of the Model

For simplicity purposes, we can aggregate back again the dummy features and obtain the following feature importance:

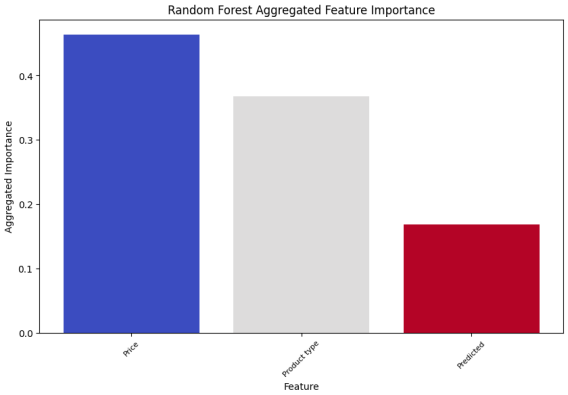


Figure 15: Aggregated Feature Importance of the Model

We can confirm that indeed the predicted motif is the least important of the features to predict sales as proxied by reviews. Two main explanations can be given to these results: The first is that our CNN is trained on a set of data that can have potential classification options that are not taken into consideration on our training. Therefore, many motifs are not being correctly captured. The second explanation is that even if the classification were correct and the motif were accurately predicted, we are not capable of extracting more abstract information about the images such as quality design. These remarks help us defining future considerations to projects similar to this one that we might develop.

5 Conclusion

The models we created for the image classification and for the prediction of sales have many positive aspects but also many downfalls. As a matter of fact, the end prediction of sales given the information that we had is not at all sufficient. We can trace back this difficulty on many different levels starting from the data collection, maybe not always precise; from the data cleaning where some images could create confusion since they are borderline between categories; from the not sufficient width of the classes considered, there could have been also classes on which the model was not trained, such as monsters, bats, vampires... Also the structure of the image recognition model focuses more on the image as a whole and does not take into consideration that in the end the test set would have the class of interest as a small part of the whole picture, to the data used for the training and testing of the Random Forest Regression where maybe some important features were not present and the size of the dataset was not big enough. Given all this input there is space for future opportunities of development of the idea.

5.1 Further Opportunities

We have identified many areas of improvement and refinement of our model and approach by encountering some challenges in our way. We could extend our model to also recognize the product type without relying on simply being a result for a search query, as some products are misclassified or purposely targeted incorrectly. In fact, data for training might be more available if the purpose is to identify a dress, shirt, pants, etc. Another potential improvement to our model could be to go deeper than obtaining a single class for each image.

Having a multiclass classification model would be even better as we would obtain a probability output for each of our target classes, and therefore be able to have more confidence on our results if there is a very strong probability for a given object or design.

What is more, it would be possible to extend the model to a multilabel classifier. In this case, the output classification for each image would be a probability for each of the potential products or designs on the image. This version would be the most powerful of all, as it would be able to classify multiple designs on multiple images for single images, therefore narrowing down trend identification.

Additionally, a wider set of data could be extracted from each image, including information about color, styling techniques, fabrics, and other characteristics that can also be very relevant for customers. There is even potential to identify products based on logo recognition.

This technique is called *Image/Object Detection* and focuses specifically on the classification of the designs since our current model classifies the entire image regardless if it has a model showcasing it, a mockup of the actual product or just the design. A simple explanation of how this would work is by first, training the image to recognize different types of objects (like we just did for our current CNN for image classification), but we would go a step further by recognizing and pinpointing the positions of numerous objects within an image, in our case, the specific designs contained in the clothing. It would then, draw bounding boxes around them to indicate their exact location, which could be done using popular techniques such as YOLO (You Only Look Once)[3]. However, before detecting the objects and even before classifying the image, we need to understand what the image consists of. This is where Image Segmentation is helpful.

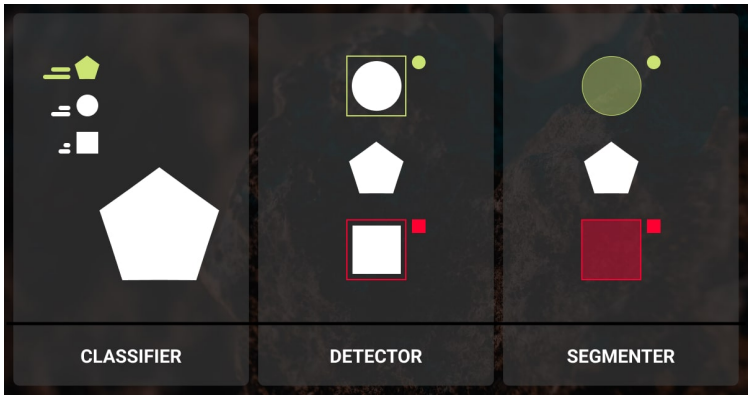


Figure 16: Process of Image Detection

We can divide or partition the image into various parts called segments. It's not a great idea to process the entire image at the same time as there will be regions in the image which do not contain any information. By dividing the image into segments, we can make use of the important segments for processing the image (the designs). In essence, we group together the pixels that have similar attributes using image segmentation which will create a pixel-wise mask for each object in the image. This technique gives us a far more granular understanding of the object(s) in the image[16], allowing us to better predict the designs in each product.

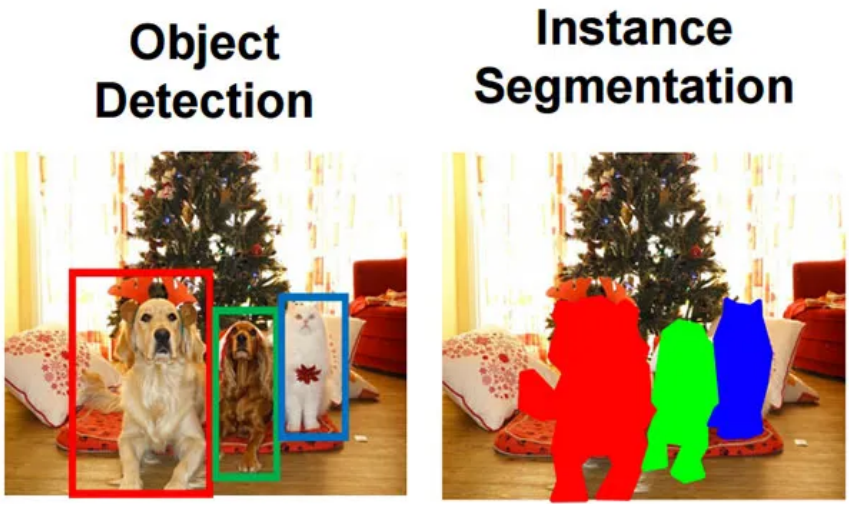


Figure 17: Image Segmentation vs Image Detection

Going back to the Random Forest for Regression we can say that more precise data coming from the CNN can have a great impact on the improvement of the model, but also some other information could be web scraped to try to have more characteristic of the product such as the

material, the transportation costs, promotions and many other. Also some NLP for the extraction of other information from the text description could be implemented to offer a even greater variety of data on the product. Not to forget is the fact that businesses will not be interested in the prediction of sales for all the categories of product and will be focusing only on some of them, making the creation of the dataset for that occasion more precise. And a data collection aimed at selecting single categories can lead to more precise results.

References

[1] Etsy landing page.

[2] Bo Chen Dmitry Kalenichenko Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam Andrew G. Howard, Menglong Zhu. Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[3] Label Your Data. What's the difference between image classification object detection?

[4] datagen. Resnet-50: The basics and a quick tutorial.

[5] deepchecks. What is densenet?

[6] Analytics for decisions. What is ensemble learning?

[7] Geeks for geeks. Residual networks (resnet) – deep learning.

[8] freeCodeCamp. What is a convolutional neural network? a beginner's tutorial for machine learning and deep learning.

[9] geeks for geeks. Introduction to convolution neural network.

[10] geeks for geeks. Understanding googlenet model – cnn architecture.

[11] OpenGenus IQ. Architecture of densenet-121.

[12] OpenGenus IQ. Introduction to inception models.

[13] Scikit learn. Ensembles: Gradient boosting, random forests, bagging, voting, stacking.

[14] MathWorks. Googlenet convolutional neural network.

[15] Medium. Ensemble: Boosting, bagging, and stacking machine learning.

[16] Medium. Image classification vs. object detection vs. image segmentation.

[17] Quoc V. Le Mingxing Tan. Efficientnet: Rethinking model scaling for convolutional neural networks.

[18] Paperspace. Introduction to bagging and ensemble methods.

[19] paperswithcode. Densenet.

[20] Statista. Planned annual halloween expenditure in the united states from 2005 to 2023.

[21] viso.ai. Vgg very deep convolutional networks (vggnet) – what you need to know.