AmazonGo

Long Time No C

Data Science Capstone Project
Exploratory Data Analytics Report

Date:

11/23/2020

Team Members:

- Smith Amornsaensuk
- Himanshu Jat
- Harshit Geddam
- Chingiz Mardanov

Introduction

The purpose of this report is to describe analysis of our data in several areas. The analysis of the basic metric of variables such as data types, size and the medata are described first. Afterward, multivariate image analysis and features engineering analysis are discussed. However, since our dataset contains only images and objects are manually labeled in images, missing values and outliers are not interested.

Analysis the basic metrics of variables

The data used in this project consist fully of images. So, features represent pixels, and they are all continuous variables which range from 0-255. The data consists of 366 images which are in different sizes; therefore, each data instance (image) a . Since an image contains multiple objects, the analysis of the data is performed on both original images and 2,203 cropped images of each object.

Original Images

Original images in the dataset are images that contain multiple different grocery items from different environments such as vending machines and supermarket shelves.

Figure 1: Examples of Data Instances

Metadata of original images is created based on the information in the file name, height, width, channel, height/width ratio, image size, image mean, image median, image standard deviation, minimum value, maximum value, image range and number of objects in the image. The description are as follows

file_name: The file name of an image
height: height of an image
width: width of the image
channel: the channels of an image which RGB are 3 channels and some image has the 4^{th} channel as alpha
hw_ratio: height/width ratio of an image
im_size: the size of image represents by [height, width]
img_mean: the average value of pixels forms all channels
img_median: median value of pixels from all channels
img_std: standard deviation of pixels from all channels

img_min: minimum value of pixels
img_max: maximum value of pixels
img_range: the range of pixels
num_obj: number of objects contained in an image

	file_name	height	width	channel	hw_ratio	img_size	img_mean	img_median	img_std	img_min	img_max	img_range	num_obj
45	635178.jpg	600	600	3	1.000000	[600, 600]	220.070294	255.0	72.314757	0	255	255	1
347	images.jpeg	299	168	3	1.779762	[299, 168]	102.563379	95.0	65.979369	0	255	255	2
57	7daysimages.jpeg	260	194	3	1.340206	[260, 194]	126.432461	130.0	64.674647	0	255	255	5
323	YKnrzJN.jpg	315	600	3	0.525000	[315, 600]	101.892215	78.0	63.127018	0	255	255	2
273	IMG_20190306_172745.jpg	4608	3456	3	1.333333	[4608, 3456]	82.603665	70.0	63.841788	0	255	255	5
277	IMG_20190306_173147.jpg	4608	3456	3	1.333333	[4608, 3456]	92.357249	93.0	66.463623	0	255	255	4
225	IMG_20190206_170446.jpg	4608	3456	3	1.333333	[4608, 3456]	76.730215	54.0	65.926778	0	255	255	9
118	IMG_20181218_165508.jpg	4640	3480	3	1.333333	[4640, 3480]	75.388485	47.0	66.988335	0	255	255	8
100	DSC01673.png	6000	4000	3	1.500000	[6000, 4000]	51.587464	36.0	51.645379	0	255	255	2
187	IMG_20181218_173705.jpg	4640	3480	3	1.333333	[4640, 3480]	84.119709	62.0	69.246860	0	255	255	11

Table 1: Meta Data of the dataset

Table 1 shows the example of metadata that has been created from original images.

	height	width	channel	hw_ratio	img_mean	img_median	img_std	img_min	img_max	img_range	num_obj
count	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.0	366.000000	366.000000
mean	3468.125683	3050.898907	3.005464	1.106893	93.872818	80.778689	66.543478	0.060109	255.0	254.939891	6.019126
std	1822.380856	1478.497604	0.073821	0.357759	36.201146	52.121856	6.911148	1.149958	0.0	1.149958	3.806579
min	166.000000	168.000000	3.000000	0.523333	39.083864	12.000000	37.689250	0.000000	255.0	233.000000	1.000000
25%	1200.000000	1408.000000	3.000000	0.750000	74.512368	50.000000	62.787467	0.000000	255.0	255.000000	3.000000
50%	4608.000000	3456.000000	3.000000	1.333333	84.781048	66.500000	65.882558	0.000000	255.0	255.000000	6.000000
75%	4640.000000	3480.000000	3.000000	1.333333	97.944805	88.750000	70.209762	0.000000	255.0	255.000000	9.000000
max	6000.000000	6000.000000	4.000000	2.181818	228.975856	255.000000	105.089753	22.000000	255.0	255.000000	32.000000

Table 2: Statistic Description of Metadata

```
*-------------------------------------------------------------------------------*
number of images                        |  366

dtype                                   |  uint8
channels                                |  [3, 4]
extensions                              |  ['jpeg', 'png', 'jpg']


recommended input size(by mean)         |  [3472 3048] (h x w, multiples of 8)
recommended input size(by mean)         |  [3472 3056] (h x w, multiples of 16)
recommended input size(by mean)         |  [3456 3040] (h x w, multiples of 32)

channel mean(0~1)                       |  [0.39702985 0.35444608 0.32380277]
channel std(0~1)                        |  [0.29916894 0.2966006  0.3038216 ]
*-------------------------------------------------------------------------------*
```

Table 3: More Statistic Description of Metadata

Table 2 shows descriptive information of dataset's metadata. From this table you can observe that, average height/width ratio is 1.1 with standard deviation 0.07 which means that they are mostly almost square shaped. The minimum range of image pixel value is 233 which means some images only consist of 233 different values; however, maximum values is 255 which explains that there are some images that do not contribute the all 0-255 values. The average number of objects in an image is 6 where minimum and maximum are 1 and 32 respectively.

Form Table 3, the data type of features is uint8. File types of images are jpeg, jpg and png. There are some images that have 4. For further analysis such as multivariates and feature selection, we decided not to use the fourth channel. The normalized average values for each channel RGB are 0.397, 0.354 and 0.323 with standard deviation of 0.299, 0.3 and 0.304 in order.

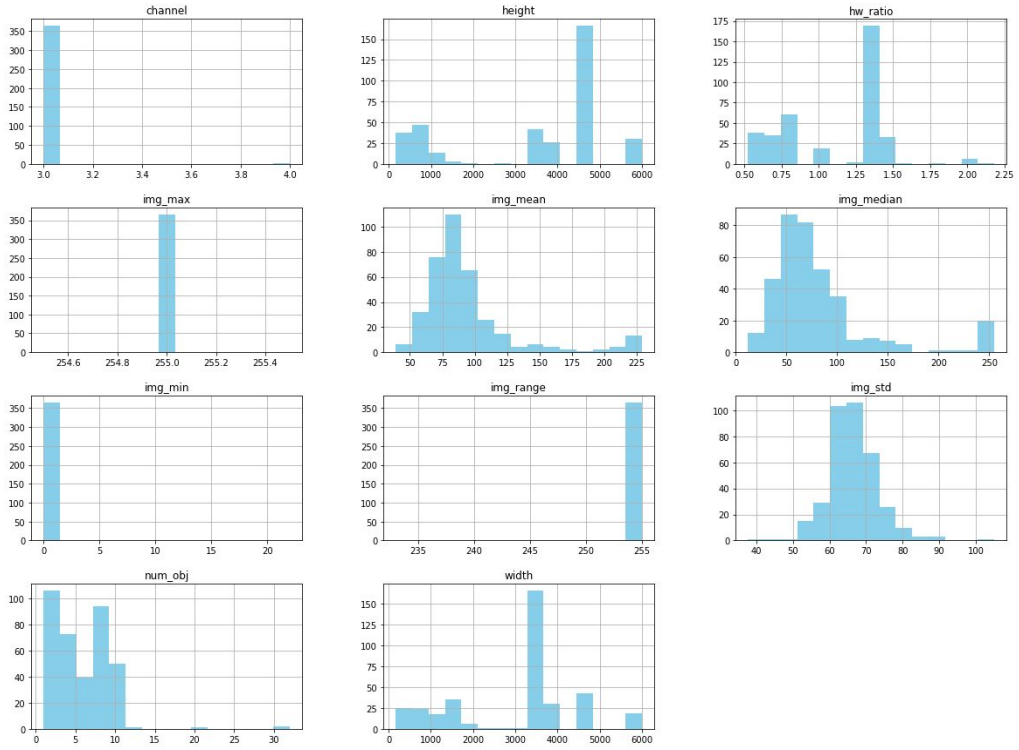

Figure 2 Distribution of Original Images

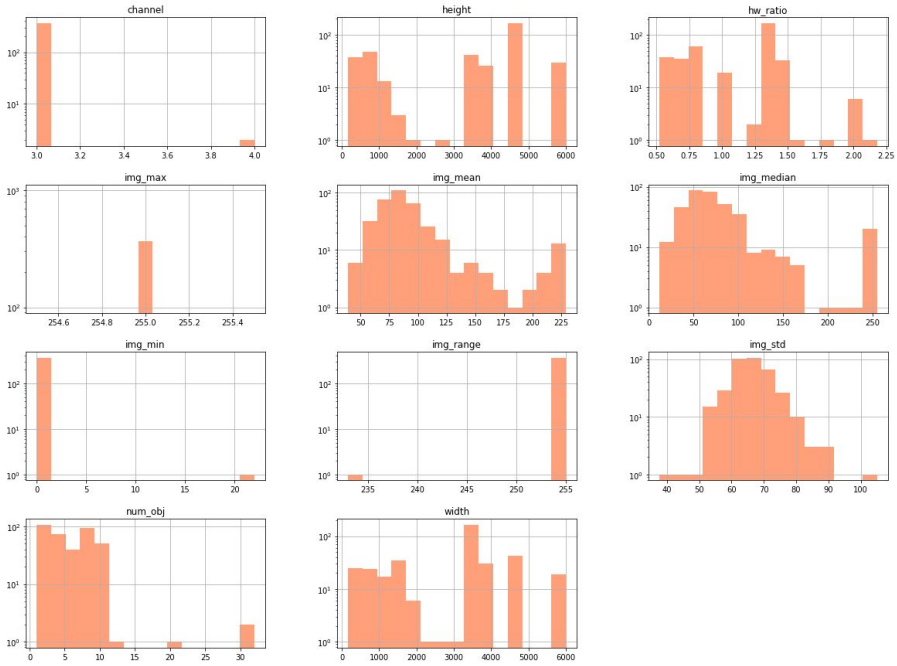

Figure 3: Distribution of Original Image log scaled on y-axis

Figure 2 and Figure 3 shows the distribution of variables in the original dataset. Note that Figure 3 has a log scale on the y-axis. In these figures, there are a few images that have 4 channels and the rest are 3 channels. The standard deviation of images is normally distributed while the mean and median of images are right skew. The height and width ratio are scattered; however, the analysis tells us that the dataset contains groups of images that have the same dimension as we observe from the distribution of hw_ratio. In addition, the dimensions of the image are distributed in two main groups i.e. height/width ratio of 0.5-0.8 and 1.25-1.5. Most images contain 1, 5 and 9 objects. The information gains us insight about how the products are placed on shelves which will affect how our models will be trained and tested.

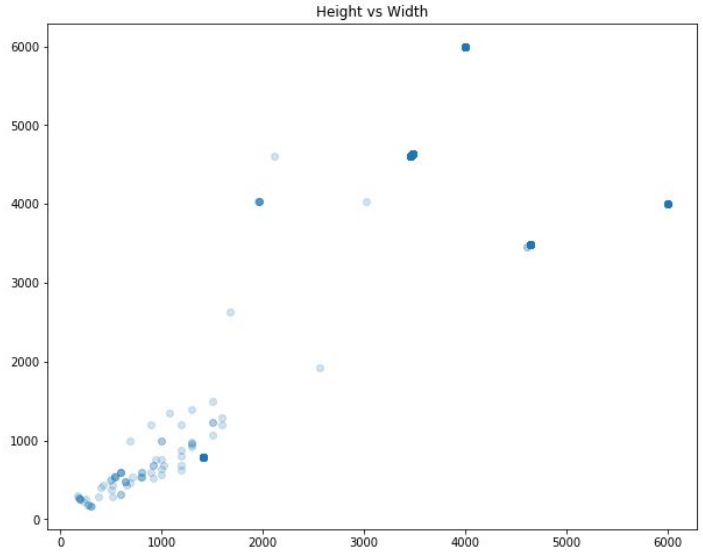

Figure 4: Scatter Plot of Height and Width of Original Images

Figure 4 shows a better picture of the original images dimension distribution. We can clearly see from the scatter plot that there are images scattered with height and width less than 1,000 pixels while some large dimension images have exact same size from solid blue dots. This brings us to find out if all images need to be scaled to the same dimension before feeding into TF Object Detection API

Cropped Images

In each image, there are objects that will be recognized by the model. Objects are emphasized by using a tool called labelImg, which creates spatial information of objects in an image by generating a corresponding xml file. The xml file contains the boundary information as xmin, xmax, ymin and ymax along with the corresponding label (object name). The information is also used to extract each object from images for analysis purposes.

Figure 5: Examples of Cropped Images

	image_name	label	xmin	xmax	ymin	ymax	width	height	hw_ratio
0	000121394	twix	12	1000	299	701	988	402	0.406883
1	0004000000435_6_A1C1_1200	twix	2	1200	346	862	1198	516	0.430718
2	127900-01_twix-fun-size-candy-bars-35-piece-bag	twix	51	539	189	350	488	161	0.329918
3	127952-01_snickers-candy-bars-48-piece-box	snickers	9	490	127	251	481	124	0.257796
4	141999779-moscow-russia-february-8-2020-red-bu...	redbull	339	605	41	792	266	751	2.823308
5	141999779-moscow-russia-february-8-2020-red-bu...	redbull	635	905	56	831	270	775	2.870370
6	20190103_103704	snickers	1596	1802	1100	1514	206	414	2.009709
7	20190103_103704	evian	785	970	1620	2154	185	534	2.886486
8	20190103_103704	redbull	1562	1688	2345	2677	126	332	2.634921
9	20190103_103704	redbull	1373	1513	2351	2694	140	343	2.450000

Table 4: Objects Information in Images

Table 4 shows the information of each bounding box in an image. The image_name is the name of the original image in the dataset and the position information represents xmax, xmin, ymax and ymin with corresponding labels as an object name. The ratios of height/width of objects are described at hw_ratio column. Objects are cropped from original images based on the information in Table 4 which results in 2,203 images of 8 objects.

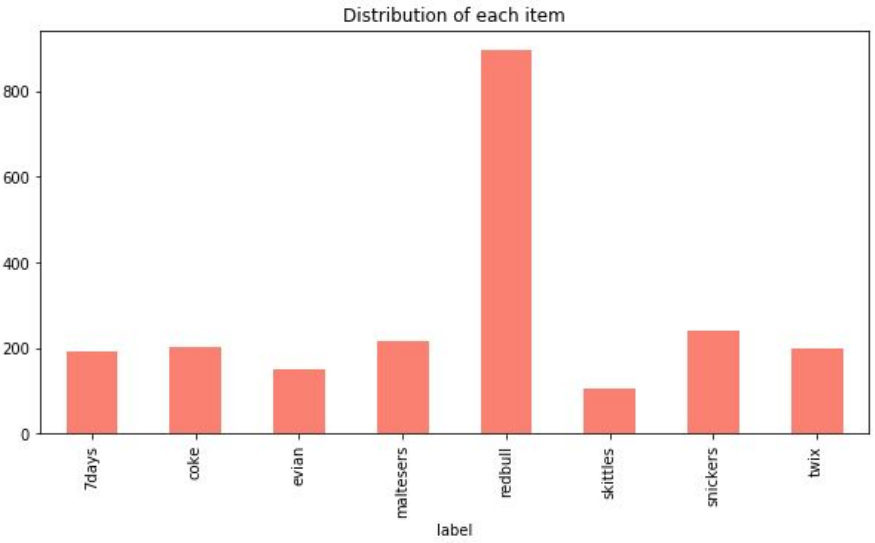

Figure 6: Distribution of Objects

Figure 6 shows the distribution of 8 items (label). The figure illustrates that most labels have about 200 images while redbull appears on many more images. From the information, we may consider up-sampling or down-sampling techniques in order to improve the performance of the model.

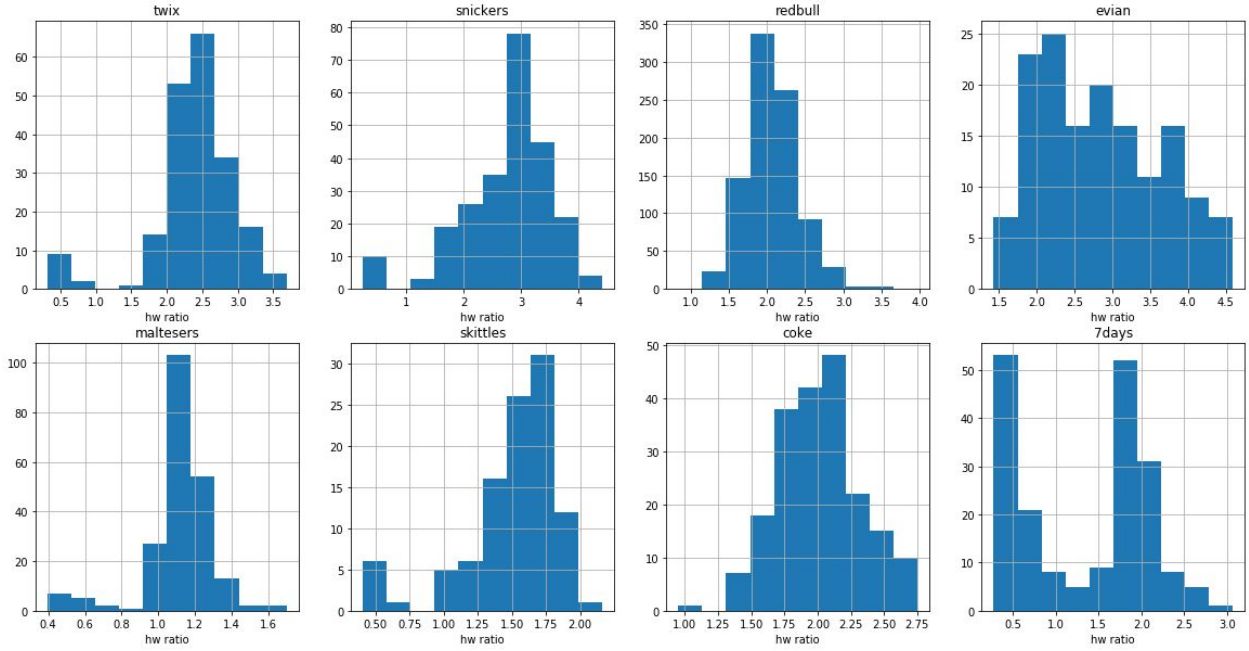

Figure 7 Distribution of Height/Width Ratio of Objects Image (cropped images)

Figure 7 shows distribution of height/width ratio for each object. The ratio greater than one represents portrait objects which correspond to how they are oriented on a shelf. On the other hand, the ratio less than one represents landscape objects which means the items are on a shelf in landscape orientation. The figure illustrates that each object has reasonable ratios which are either mostly greater than 1 or less than 1. From the above plots one can observe that most of the objects are primarily oriented in one direction -

portrait. . However, the ratio of 7days is split into 2 main subgroups i.e. 0.25-0.8 and 1.5-2.5 which illustrates that images of 7days come in both portrait and landscape orientation. As a result, we have drawn a hypothesis that the model may have a problem while detecting and classifying this class. We will consider dealing with this issue after evaluating the model performance.

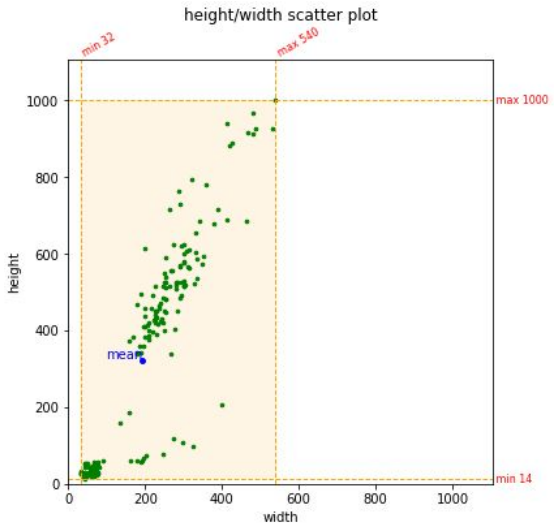

Figure 8: Scatter Plot of "7days's" Height and Width

Figure 8 illustrates an alternative perspective of "7days" images dimension. As we can see from the figure, the data is separated into 2 groups which a portrait images group located in high-resolution region i.e. more than 400px height, and a landscape images group are dense in low-resolution region. The information helps us make decisions on whether we must eliminate one of the groups in case that this issue induces a problem to the model.

Multivariate Image Analysis

MIA is performed on cropped images to investigate different features and behaviors of each class labels (8 objects). The techniques for our MIA include image channels' histogram analysis, edge detection and Principal Component Analysis.

Channels Histogram

The image channels' histogram is performed by using the BasicImageEDA module. The result shows distribution of each RGB channel for groups of images according to their labels.

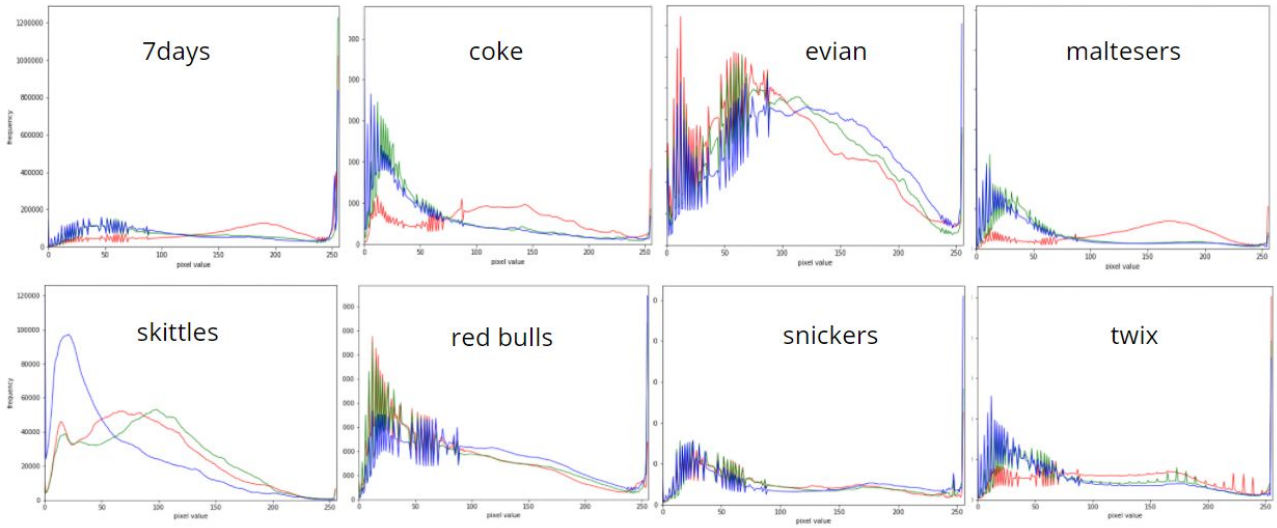

Figure 9: RGB distributions of each object images

Figure 9 shows RGB distributions of images for each group. We observe that each group (label) has different behavior of distribution for each channel. For example, evian, snickers, and red bull have similar distribution for RGB but they are different in shape i.e. very flat for snickers. Maltesers have high response for pixel values in range [0,50] for green and blue, and range [150, 200] for red. Skittles seem to have very high responses in range [0,50] while moderate responses of green and red in range [50,100]. The information from Figure 9 demonstrates the difference behavior for data in each class. From these findings we can also make some assumptions about which items our model might struggle differentiating. Plots for maltesers and twix seem very similar and we suspect that this might pose a challenge to the model.

Edge Detection: Feature Extraction

The edge detection is performed using Sobel operators which computes an approximation of the gradient of the image intensity. There are two Sobel filters used in this analysis which act as a horizontal and vertical edge detection. The results of gradient descent for each object are described below.

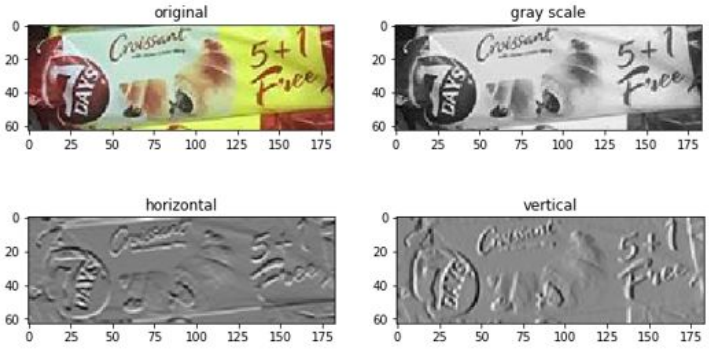

Figure 10: 7days edge detections

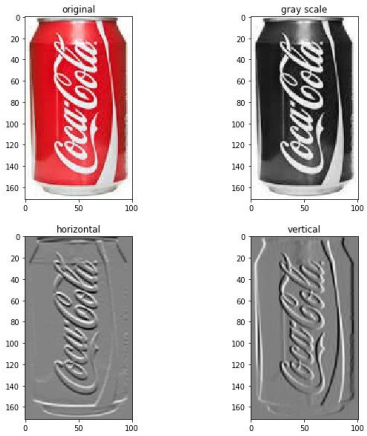

Figure 11: Coke edge detections

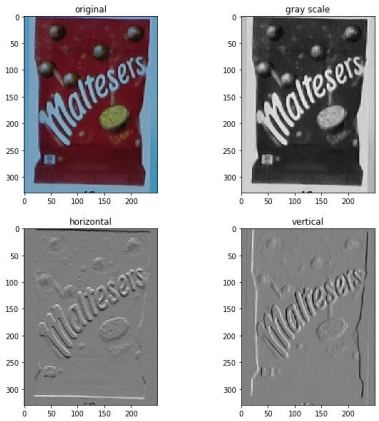

Figure 12: Maltesers edge detections

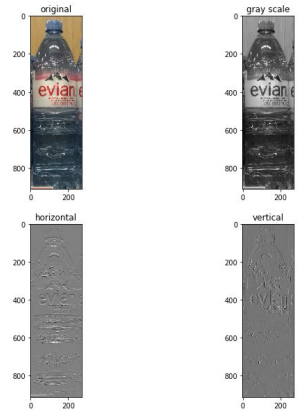

Figure 13: Evian edge detections

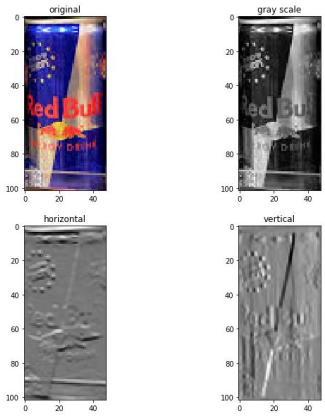

Figure 14: Redbull edge detections

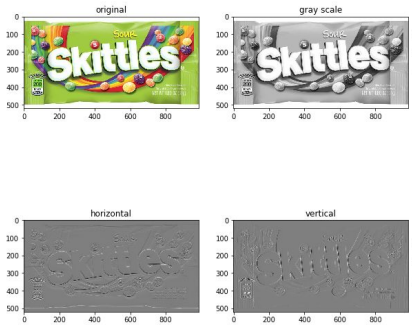

Figure 15: Skittles edge detections

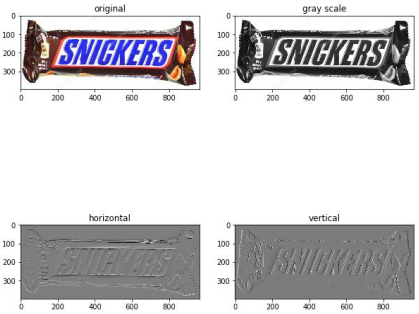

Figure 16: Snickers edge detections

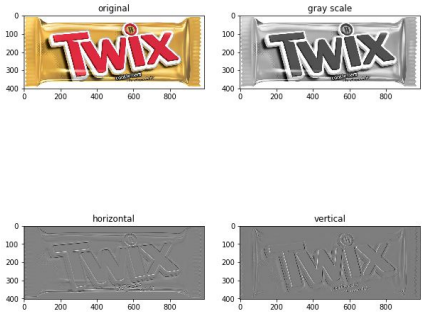

Figure 17: Twix edge detections

Figure 10-17 shows the object image itself, its gray scale version, horizontal edge detection and vertical edge detection of an object image. We can see that edge detections are able to extract important features from images. The significant information that edge detection can extract are the object trademarks and packaging designs. Furthermore, the shapes of objects such as bags, can and bottles shapes are also detected.

Principal Component Analysis: Features Selection

In order to perform PCA, the image should have the same dimension. Images are resized based on outputs from BasicImageEDA module which provides recommended input size for each group of images. Images are converted to gray scale and are trained in PCA separately according to their labels (object). Before performing PCA, average and standard deviation of images are investigated to gain basic information of each object.

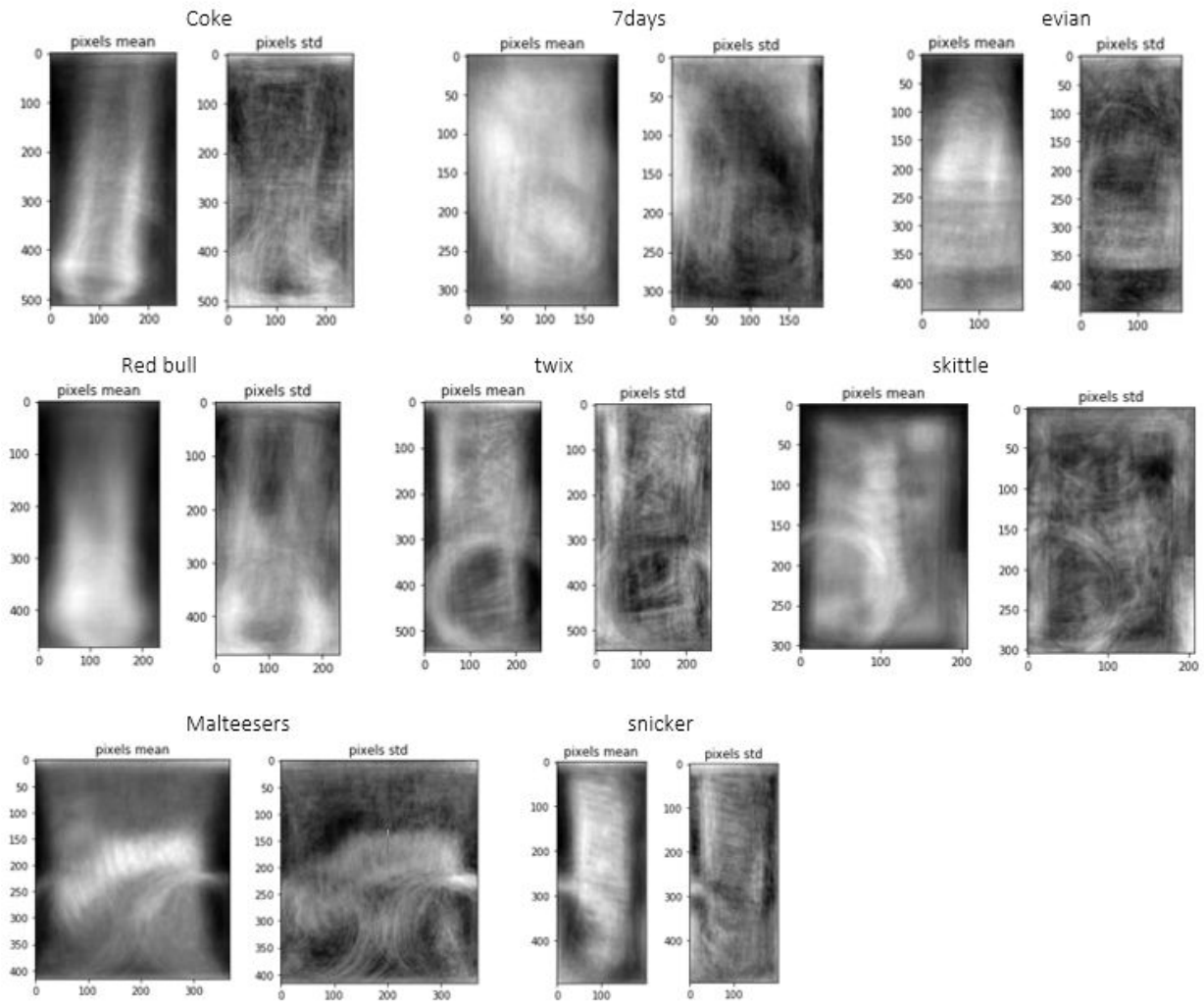

Figure 18: Image of mean and standard deviation of each object

Figure 18 shows mean and standard deviation for each image group. Means of skittles, maltesers and snickers capture the information and location of texts on the package. Furthermore, images of mean also show a clear shape of object in images for example we can see the rectangular shape of a can and packaging in coke, red bull, twix, skittles, maltesers and snickers while a shape of water bottle is somewhat observable in Evian picture. Standard deviation also captures some design details on the package. PCA is performed by keeping 70% of variances which reduces dimensions to be less than 50 features. Then components are visualized in order to observe values in eigen space which enable us to see how information is captured from each component.

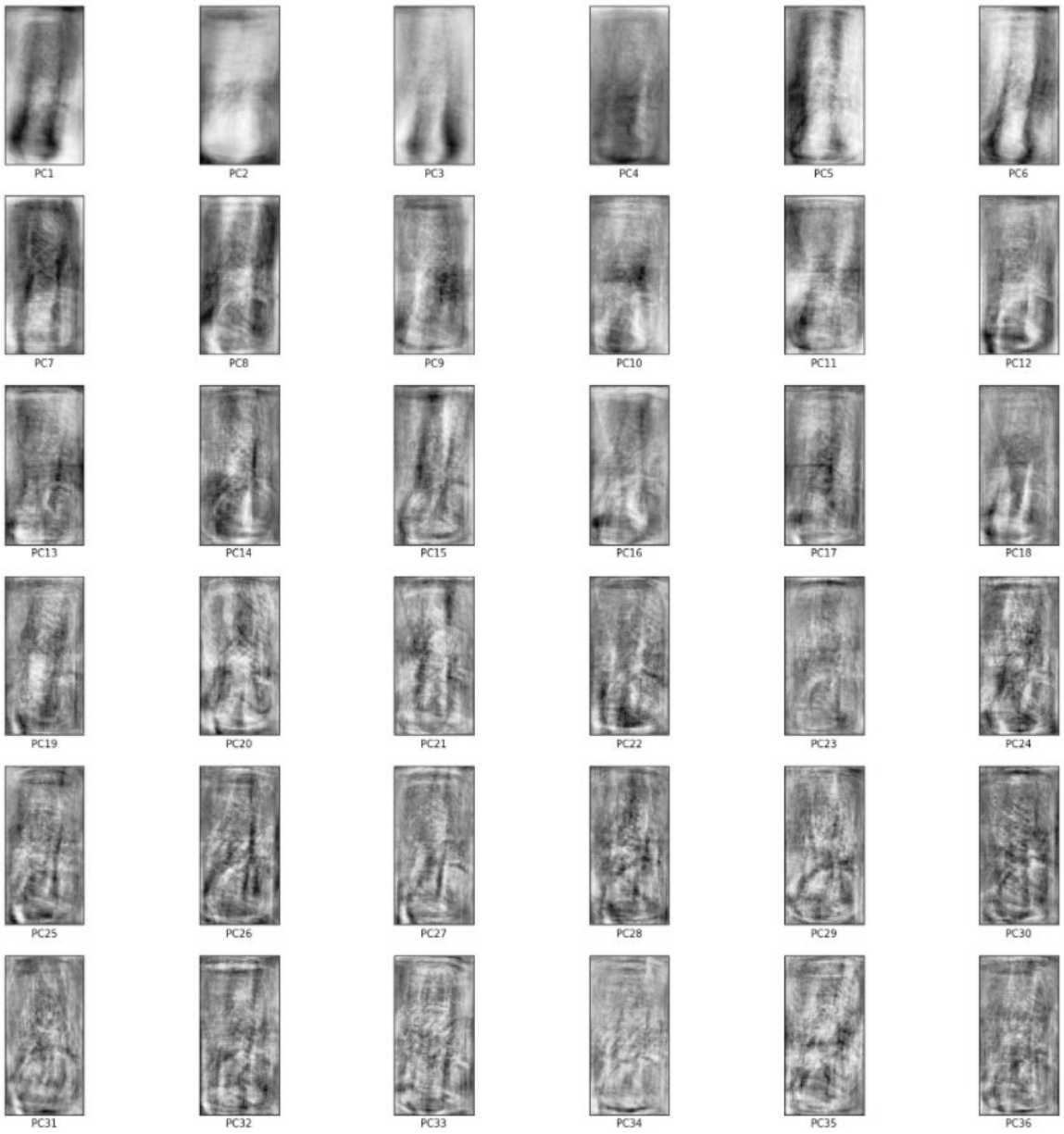

Figure 19: PCA coke

Figure 19 illustrates 36 components that can explain 70% of variance for coke images. We can observe that PC1-PC9 captures the shape of a can while PC 30 can capture the logo the most.

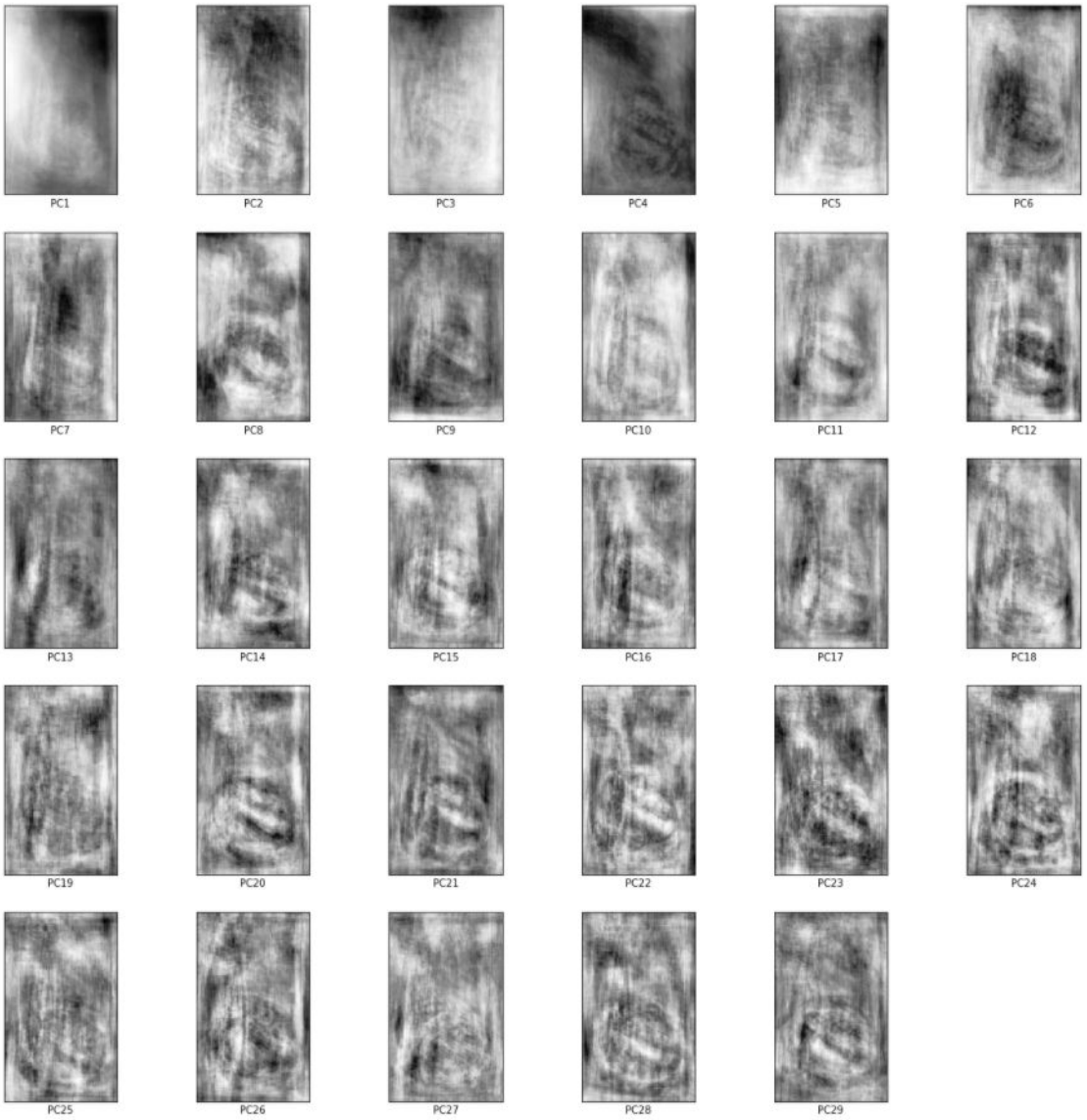

Figure 20: PCA 7days

Figure 20 illustrates 29 components that can explain 70% of variance for 7days images. The first component can capture a shape of the object in the image but there is no other information in this component. There are some components that can capture the number "7" on the logo well i.e. PC9, PC10, PC11, PC14, PC21 and PC24.

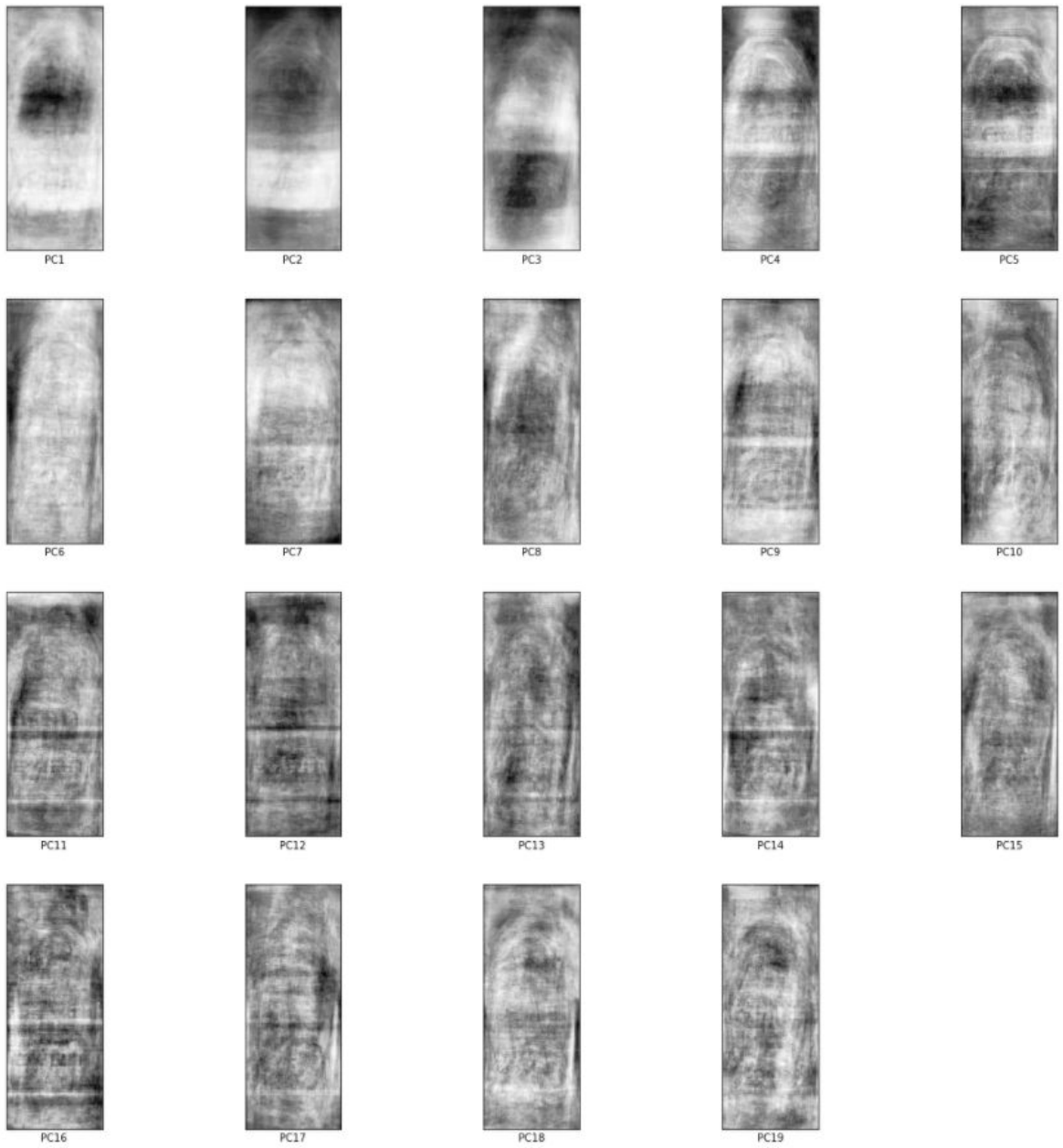

Figure 21: PCA evian

Figure 21 illustrates 19 components that can explain 70% of Evian images. The location of the bottle's label is captured by PC1 – PC3 while PC4 and PC5 capture the shape of the bottle.

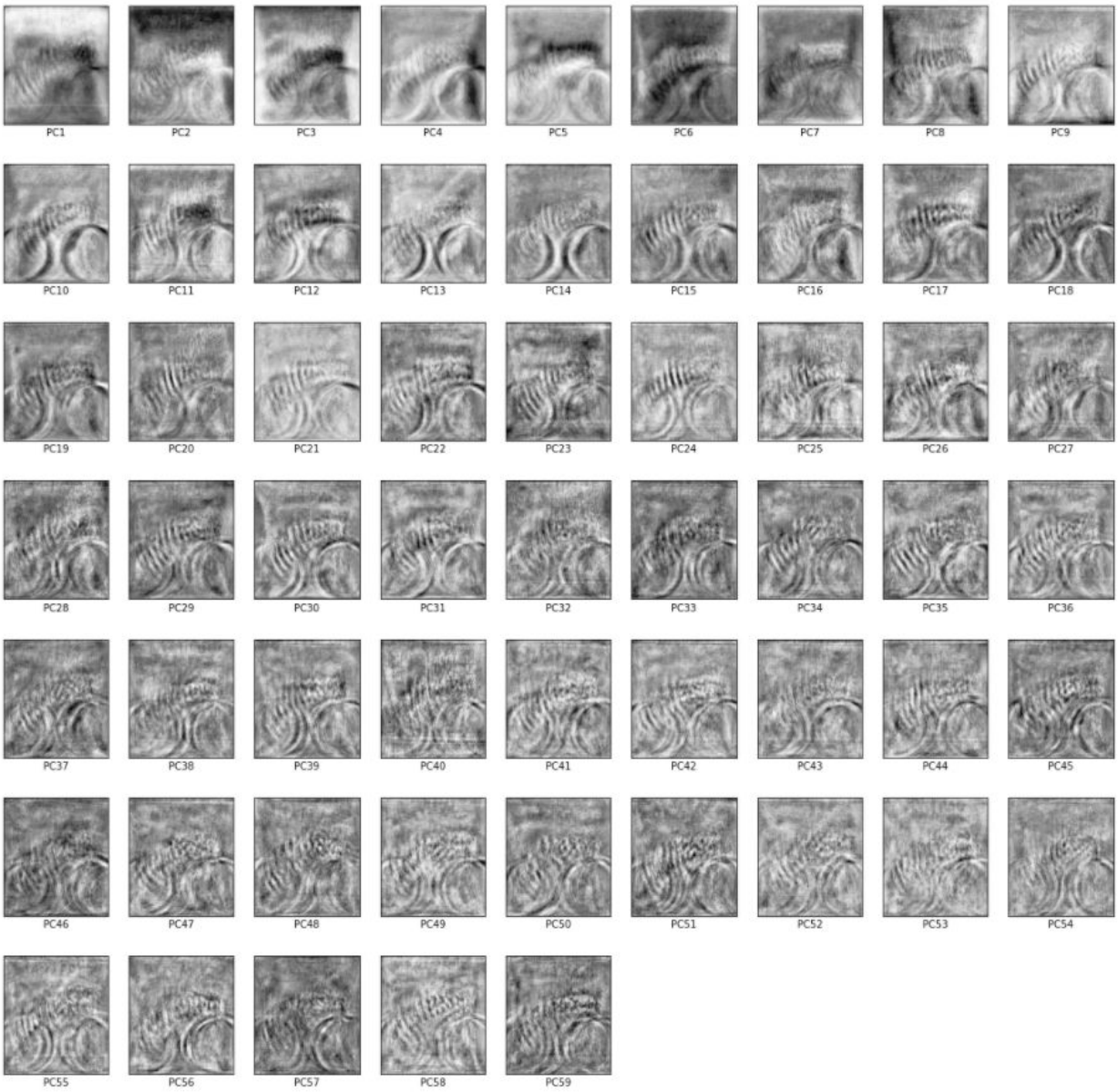

Figure 22: PCA Maltesers

Figure 22 illustrates 59 components that explain 70% variances of maltesers images. PC 6 can capture the position of the logo and the shape of an object. Two circles that are captured in most of the components are parts of the vending machine in the image. It shows that undesired problems will occur if the model considers those rings as a part of the object.

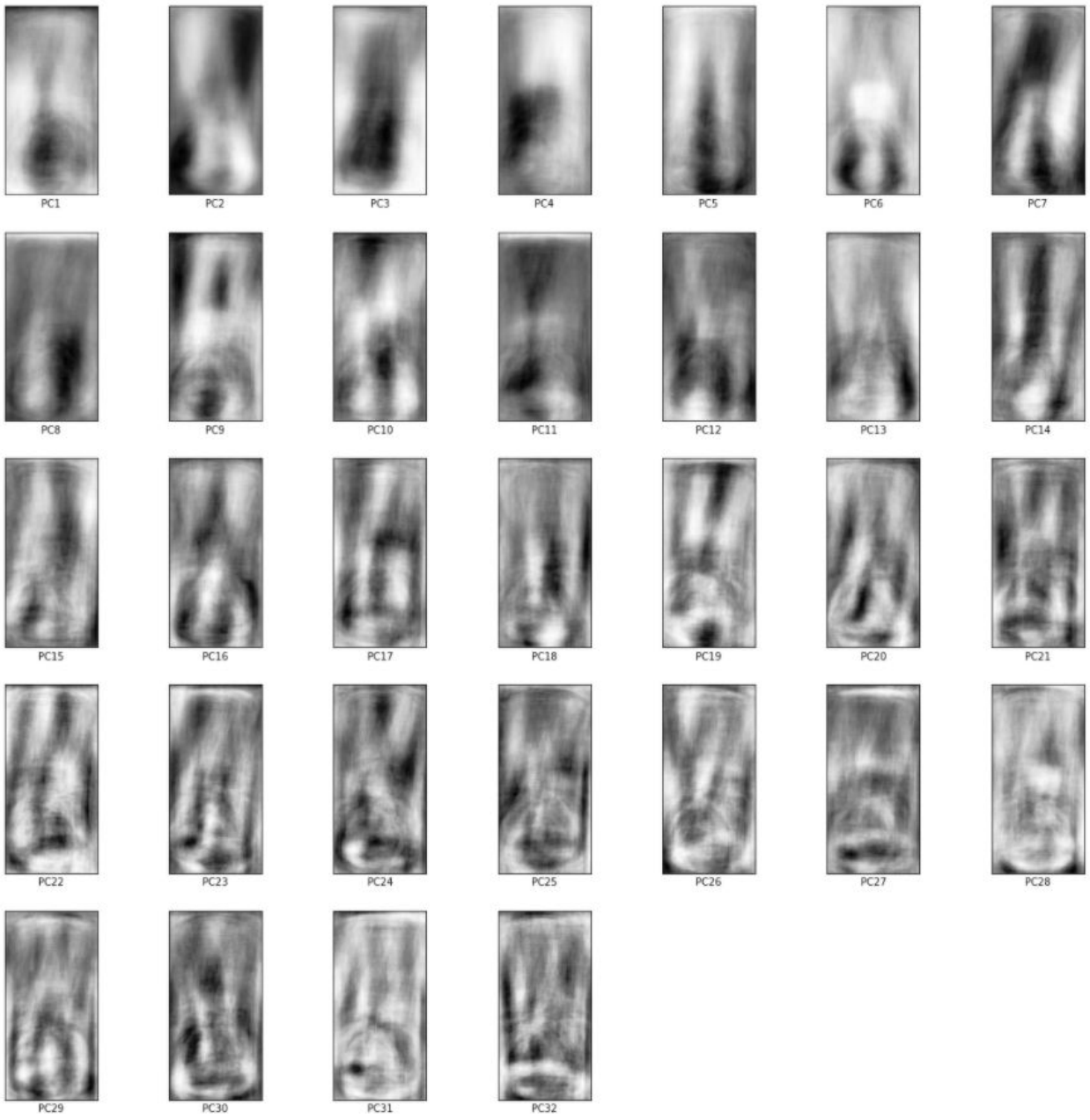

Figure 23: PCA red bull

Figure 23 illustrates components of 70% variances of redbull images. As we can see from the figure, PCA can capture only some patterns which print on the can and a shape of a can.

Figure 24: PCA Twix

Figure 25 illustrates 52 components of Twix. PC1-PC6 capture mainly captures the shape of an object. PC 12 captures logo with less noise comparing to higher components

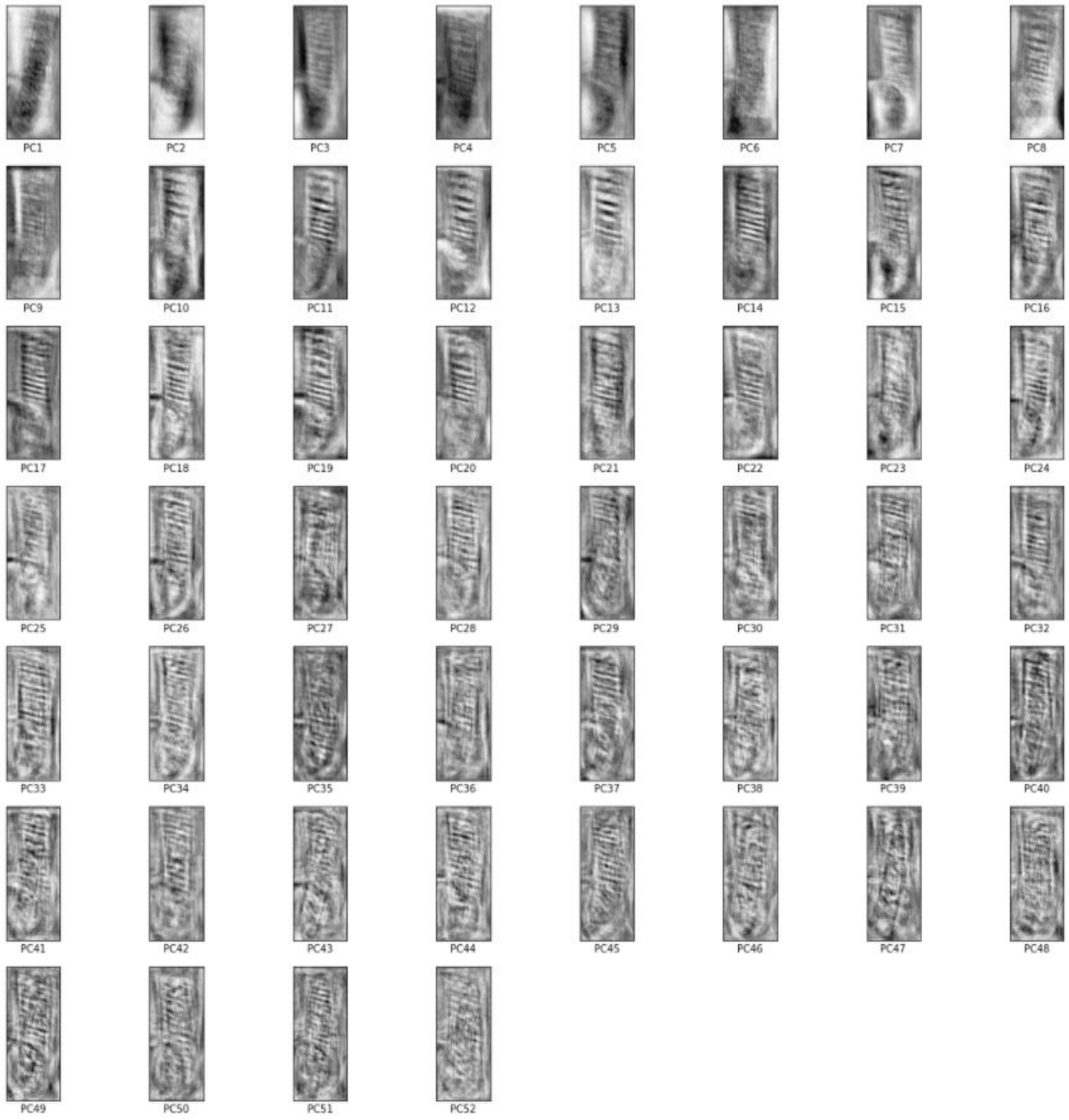

Figure 25: PCA snickers

Figure 26 illustrates components which hold 70% variance of Snickers images. As we can see from thefigure, the PCs Snickers and Twix are similar. They both have 52 components and the visualization of their eigenspaces behave in the same way. However, the details of the logo are different.

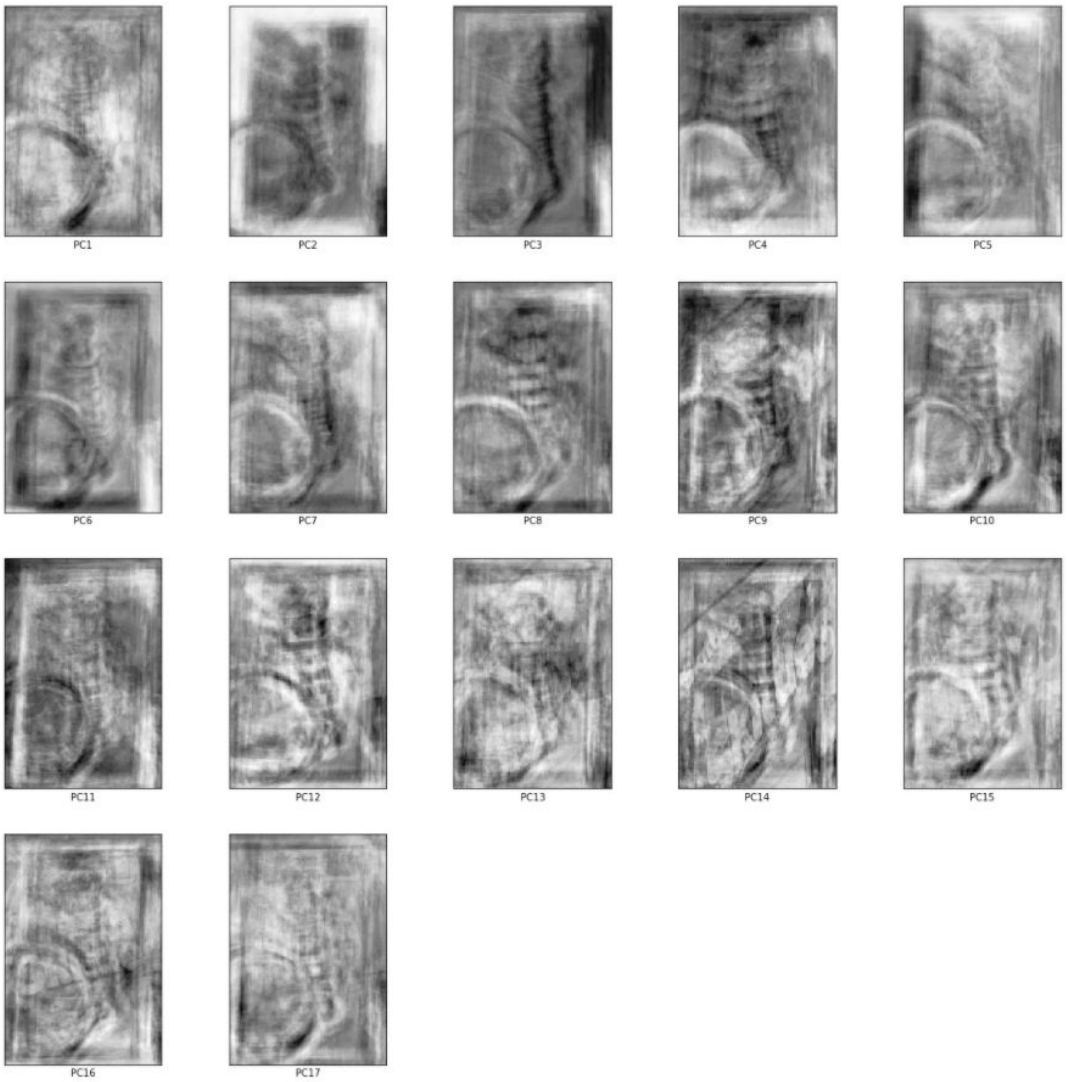

Figure 26: PCA skittles

Figure 24 illustrates PCA of Skittles which 70% of variances is explained by 17 components. We can clearly see the position and alignment of a text logo in PC12, and PC14. However, we also see a logo that is oriented in a different direction which results from objects that are oriented both landscape and portrait in the dataset.

Object detection and Classification

This is the final and the biggest component of our project in which we've created 2 jupyter notebooks using online resources and Tensorflow object detection API documentation. We have trained 4 models by utilizing cloud computing resources including 3 virtual machines and 1 colab notebook. The specifications of the virtual machine are 2vCPU 4gb memory 50 storage and 2 VMs of 8vCPU 32gb memory. For the training we have used the tensorflow object detection API.

The images that we have collected were labelled using labelimg, and then we have got xml files, where we have converted the xml files to the csv format, which when were converted to tf record files, which were fed into our algorithm for the training.

We have trained our model for 8000 steps, and we have all the training observations with us for each step, the model has been trained.

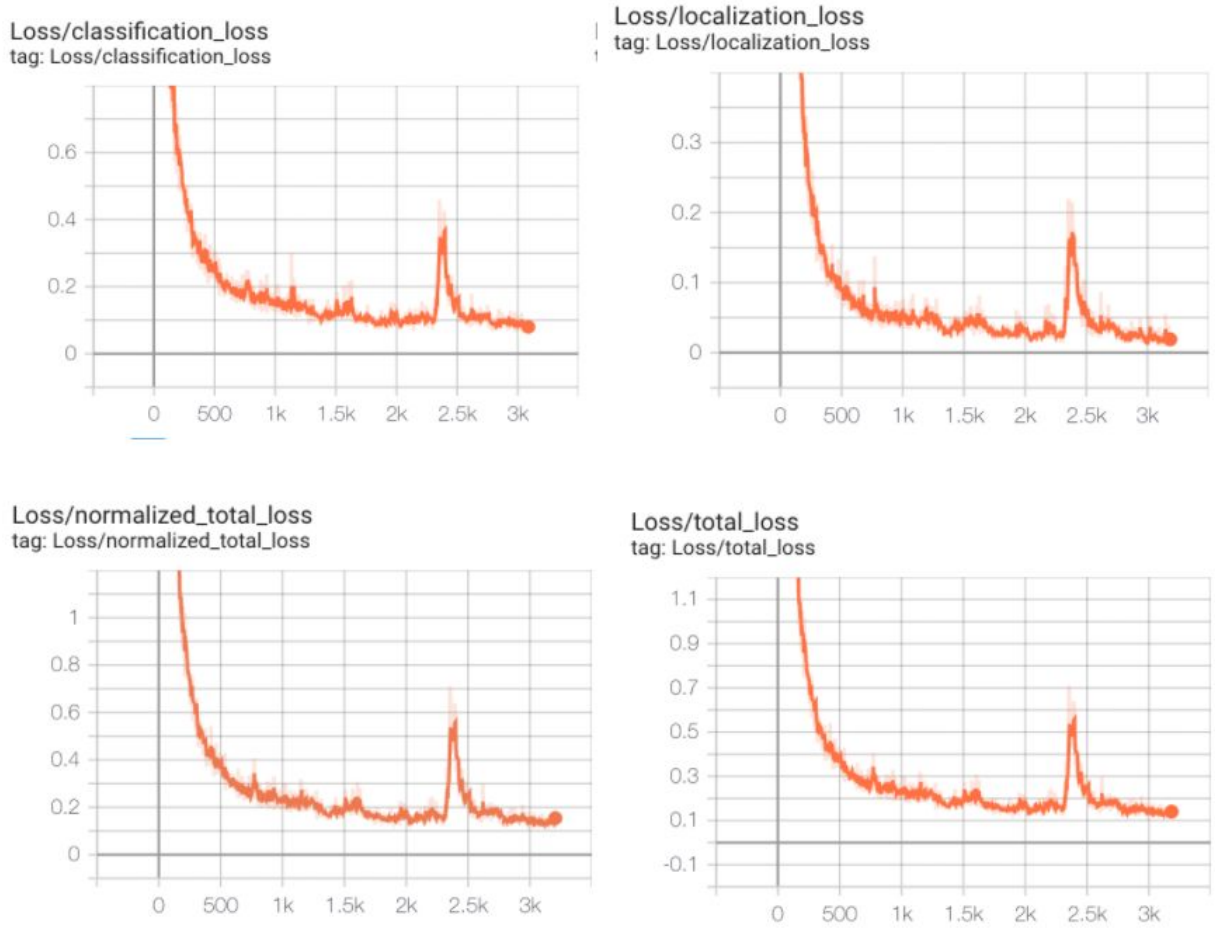

Figure 27: Training losses

Figure 27 shows classification loss, normalized total loss, localization loss and the total loss. The classification loss is a loss for the classification of detected objects into our target classes. The total loss is derived from localization_loss + normalized_total_loss. As we can see from the plots, all losses have the same behavior and they have a small peak around epoch 2.4k. The small peak indicates that total loss has spiked up then decreased in that area. Furthermore, since the magnitude of slope is still high at the end of the graph, the model still has the potential to reduce the loss with more epochs. However, to determine where the training should be stopped, it is better to consider both validation loss and training loss together.

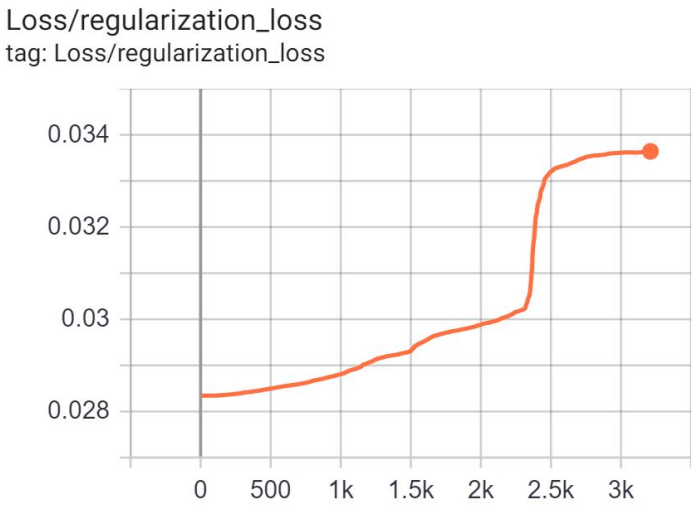

Figure 28: Training losses

Figure 28 shows regularization loss for each epoch. We observe that the regularization loss rapidly increases at 2.4k to 2.5k which corresponds to the position of a small peak in total loss. Afterward, the regularization loss is flat after the total loss reduces.

Figure 29: result on testing individual object images

Figure 29 shows the result of the model on testing images which contain a single object. We observe that the model detects objects and correctly classify them with high probability ie. 100% and 98%.

Figure 30: result on testing multiple objects images

Figure 30 illustrates the result of the model from testing on multiple object images. We observe that the model is able to detect and classify objects. Although the model correctly classifies objects, the probability of the classification of some object is low such as 58% and 60%. In addition, as we can see from the left image, the model missed a detection of maltesers. This is our future work to improve the performance of the model based on hypotheses we have made during the EDA process.

The two notebooks we created in gdrive as well as in the github for training are (see Appendix B for the link to the code):

- model_train_test.ipynb Installing dependencies, downloading TF object detection API repo, downloading the dataset, training the model, exporting the frozen model
- making_predictions_using_tf_model.ipynb : installing dependencies, downloading TF object detection API repo, using the frozen model to make predictions

Conclusion

Several analyses are performed in our EDA step which allow us to gain insights on the data. The analysis of the metadata led us to a decision of appropriately resizing images before training in order to reduce the computation cost while the distortion of original data are minimized. The observation of the channel's histogram shows us the different RGB responses for each target class. Afterward, the features engineering analysis demonstrates the important features for each class which models will learn. Finally, the trained models create information of loss during train which tensorboard is used to visualize them. Also, the objection and classification provide a decent result.

Appendix A

Codes for EDA process

```
%matplotlib inline
from skimage.color import rgb2gray
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io, viewer
from scipy import ndimage
import glob
from collections import OrderedDict, Counter, defaultdict
import xmltodict
import matplotlib.pyplot as plt
import os
import pandas as pd
import cv2
from PIL import Image, ImageOps
from basic_image_eda import BasicImageEDA
import re
import xmltodict
from sklearn.decomposition import PCA
import random

def edge_detection(image):
    '''
    arg: image
    return: a dictionary of results
    '''
    im_gray = rgb2gray(image)
    #sobel filter for horizontal edge detection
    sobel_h = np.array([[1, 2, 1],
                        [0, 0, 0],
                        [-1, -2, -1]])
    #sobel filter for vertical edge detection
    sobel_v = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]])
    # apply filters to the images
    h_edge = ndimage.convolve(im_gray, sobel_h, mode='reflect')
    v_edge= ndimage.convolve(im_gray, sobel_v, mode='reflect')

    imgs = OrderedDict()
    imgs['original'] = image
    imgs['gray scale'] = im_gray
    imgs['horizontal'] = h_edge
    imgs['vertical'] = v_edge
    return imgs

def plot_images(imgs):
    '''
    args: dictionary of images (result from  edge_detections())
    '''
    fig, axes = plt.subplots(2,2,figsize=(10,10))
    for im, ax in zip(imgs,axes.ravel()):
        ax.imshow(imgs[im],cmap='gray')
        ax.set_title(im)
```

```
def create_meta(img_files,xml_files):
    meta_dict = defaultdict(list)
    for im_file,xml_file in zip(img_files,xml_files):
        _, name = os.path.split(im_file)
        img = np.array(Image.open(im_file))
        height = img.shape[0]
        width = img.shape[1]
        channel = img.shape[2]
        img_max = np.max(img)
        img_min = np.min(img)
        meta_dict['file_name'].append(name)
        meta_dict['height'].append(height)
        meta_dict['width'].append(width)
        meta_dict['channel'].append(channel)
        meta_dict['hw_ratio'].append(height/width)
        meta_dict['img_size'].append([height, width])
        meta_dict['img_mean'].append(np.mean(img))
        meta_dict['img_median'].append(np.median(img))
        meta_dict['img_std'].append(np.std(img))
        meta_dict['img_min'].append(img_min)
        meta_dict['img_max'].append(img_max)
        meta_dict['img_range'].append(img_max - img_min)
        data = xmltodict.parse(open(xml_file,'r').read())
        if 'object' in data['annotation']:
            if isinstance(data['annotation']['object'],list):
                meta_dict['num_obj'].append(len(data['annotation']['object']))
            else:
                meta_dict['num_obj'].append(1)
    return meta_dict

def cropped_img_meta(img_files):
    meta_dict = defaultdict(list)
    for im_file in img_files:
        _, name = os.path.split(im_file)
        img = np.array(Image.open(im_file))
        height = img.shape[0]
        width = img.shape[1]
        channel = img.shape[2]
        img_max = np.max(img)
        img_min = np.min(img)
        meta_dict['file_name'].append(name)
        meta_dict['height'].append(height)
        meta_dict['width'].append(width)
        meta_dict['channel'].append(channel)
        meta_dict['hw_ratio'].append(height/width)
        meta_dict['img_size'].append([height, width])
        meta_dict['img_mean'].append(np.mean(img))
        meta_dict['img_median'].append(np.median(img))
        meta_dict['img_std'].append(np.std(img))
        meta_dict['img_min'].append(img_min)
        meta_dict['img_max'].append(img_max)
        meta_dict['img_range'].append(img_max - img_min)
    return meta_dict


def plot_pca(pca, size):
    n = pca.n_components_
    fig = plt.figure(figsize=(20, 20))
```

```
    r = int(n**.5)
    c = np.ceil(n/ r)
    for i in range(n):
        ax = fig.add_subplot(r, c, i + 1, xticks = [], yticks = [])
        ax.imshow(pca.components_[i].reshape(size[1],-1),cmap='Greys_r')
        ax.set_xlabel(f'PC{i+1}')
    plt.show()


def img_components(object_name, object_size):
    objects_data = []
    for file in object_name:
        n = np.array(ImageOps.
grayscale(Image.open(file).resize(object_size))).reshape(-1)
        objects_data.append(n)

    objects_data = np.array(objects_data)
    plt.figure()
    plt.imshow(objects_data.mean(axis=0).reshape(object_size[1],-1),cmap='gray')
    plt.title('pixels mean')

    plt.figure()
    plt.imshow(objects_data.std(axis=0).reshape(object_size[1],-1),cmap='gray')
    plt.title('pixels std')
    object_pca = PCA(n_components=0.7)
    object_pca.fit(objects_data)
    plot_pca(object_pca,object_size)



# load and create metadata
img_files = glob.glob('./dataImageFinal/*')
xml_files = glob.glob('./dataXMLFinal/*')

xml_files.sort()
img_files.sort()

original_data = create_meta(img_files,xml_files)
df_orig = pd.DataFrame(original_data)
df_orig.iloc[[random.randint(0,366) for i in range(10)],:]
print(df_orig.describe())



# Distribution of high and with ratio
_ = df_orig.hist(figsize=(20,15),bins=15,color='skyblue')
ax = df_orig.hist(figsize=(20,15),bins=15,log=True,color='lightsalmon')


# Input Images
data_dir = "./dataImageFinal"
# BasicImageEDA.explore(data_dir)

# # or

extensions = ['png', 'jpg', 'jpeg']
threads = 0
dimension_plot = True
channel_hist = True
```

```
nonzero = False
hw_division_factor = 1.0

BasicImageEDA.explore(data_dir, extensions, threads, dimension_plot, channel_hist, nonzero,
hw_division_factor)
```

label of bojects

```
labels = ['7days',
          'object',
          'evian',
          'maltesers',
          'redbull',
          'skittles',
          'snickers',
          'twix']
count_items = Counter()
xmls = glob.glob('./dataXmlFinal/*')

# meta data for each object in images
data_dict = {
    'image_name':[],
    'label':[],
    'xmin':[],
    'xmax':[],
    'ymin':[],
    'ymax':[]
}
for i, file in enumerate(xmls):
    _, file_name = os.path.split(file)
    file_name = file_name[:-4]
    data = xmltodict.parse(open(file,'r').read())
    if 'object' in data['annotation']:
        if isinstance(data['annotation']['object'],list):
            for item in data['annotation']['object']:
                count_items[item['name']] += 1
                data_dict['image_name'].append(file_name)
                data_dict['label'].append(item['name'])
                data_dict['xmin'].append(item['bndbox']['xmin'])
                data_dict['xmax'].append(item['bndbox']['xmax'])
                data_dict['ymin'].append(item['bndbox']['ymin'])
                data_dict['ymax'].append(item['bndbox']['ymax'])
        else:
            item = data['annotation']['object']
            count_items[item['name']] += 1
            data_dict['image_name'].append(file_name)
            data_dict['label'].append(item['name'])
            data_dict['xmin'].append(item['bndbox']['xmin'])
            data_dict['xmax'].append(item['bndbox']['xmax'])
            data_dict['ymin'].append(item['bndbox']['ymin'])
            data_dict['ymax'].append(item['bndbox']['ymax'])
df = pd.DataFrame(data_dict)
for col in df.columns[-4:]:
    df[col] = df[col].apply(int)

df['width'] = (df.xmax - df.xmin)
df['height'] = (df.ymax - df.ymin)
df['hw_ratio'] = df.height / df.width
df.head(10)
```

```
gdf = df.groupby('label').image_name.count()
ax = gdf.plot(kind='bar', figsize=(10,5), title='Distribution of each item', color='salmon')
fig, axes = plt.subplots(2,4,figsize=(20,10))
for label, ax in zip(df.label.unique(),axes.ravel()):
    df[df['label']==label].hw_ratio.hist(ax=ax)
    ax.set_title(label)
    ax.set_xlabel('hw ratio')

data_dir = "./cropPics"
# BasicImageEDA.explore(data_dir)

# # or

extensions = ['png', 'jpg', 'jpeg']
threads = 0
dimension_plot = True
channel_hist = True
nonzero = False
hw_division_factor = 1.0

BasicImageEDA.explore(data_dir, extensions, threads, dimension_plot, channel_hist, nonzero,
hw_division_factor)
folders = glob.glob('./cropPicGroup/*')
for folder in folders:
    print('*-------------------------------------------------------------------------------*')
    print(folder)
    data_dir = folder

    extensions = ['png', 'jpg', 'jpeg']
    threads = 0
    dimension_plot = True
    channel_hist = True
    nonzero = False
    hw_division_factor = 1.0

    BasicImageEDA.explore(data_dir, extensions, threads, dimension_plot, channel_hist, nonzero,
hw_division_factor)

# RGB Histogram for each object (single image)
file_names = glob.glob('./eachObjPic/*')
for file in file_names:
    im = Image.open(file)
    r, g, b = im.split()
    plt.figure(figsize=(10,8))
    plt.bar(range(256), r.histogram(), color='r', alpha=0.5)
    plt.bar(range(256), g.histogram(), color='g', alpha=0.4)
    plt.bar(range(256), b.histogram(), color='b', alpha=0.3)
    plt.title(file)
    plt.show()

file_names = glob.glob('./eachObjPic/*')

for file in file_names:
    image = plt.imread(file)
    d = edge_detection(image)
    plot_images(d)

coke = glob.glob('./cropPicGroup/coke/*')
coke_size = (256,512)
```

```
img_components(coke,coke_size)

seven_days = glob.glob('./cropPicGroup/7days/*')
seven_days_size = (192,320)
img_components(seven_days,seven_days_size)

evian = glob.glob('./cropPicGroup/evian/*')
evian_size = (176,448)
img_components(evian, evian_size)

maltesers = glob.glob('./cropPicGroup/maltesers/*')
maltesers_size = (368,416)
img_components(maltesers, maltesers_size)

redbull = glob.glob('./cropPicGroup/redbull/*')
redbull_size = (232,472)
img_components(redbull, redbull_size)

skittles = glob.glob('./cropPicGroup/skittles/*')
skittles_size = (208,304)
img_components(skittles, skittles_size)

twix = glob.glob('./cropPicGroup/twix/*')
twix_size = (256,544)
img_components(twix, twix_size)

snickers = glob.glob('./cropPicGroup/snickers/*')
snickers_size = (200,496)
img_components(snickers, snickers_size)
```

Appendix B

The link to the model notebook

https://drive.google.com/drive/folders/13nimb-6YAj1SsIHnXzvAuCmk9QvVOzYj?usp=sharing

Table of Contributions

The table below identifies contributors to various sections of this document.

	Section	Writing	Editing
1	Analysis the basic metrics of variables	Smith Harshit	Chigniz Himanshu
2	Histogram analysis	Himanshu	Harshit
3	Feature engineering and analysis	Himanshu Chingiz Harshit	Smith Chingiz
4	Appendix	Smith	

Grading

The grade is given on the basis of quality, clarity, presentation, completeness, and writing of each section in the report. This is the grade of the group. Individual grades will be assigned at the end of the term when peer reviews are collected.