Extensive EDA with Object Detection and Color Analysis

This notebook contains the exploration of iMaterialist Challenge (Fashion) at FGVC5 dataset

Contents

Dataset Preparation

In [61]:
```
from IPython.core.display import HTML
from IPython.display import Image
from collections import Counter
import pandas as pd
import json


from plotly.offline import init_notebook_mode, iplot
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from wordcloud import WordCloud
from plotly import tools
import seaborn as sns
from PIL import Image

import tensorflow as tf
import numpy as np

init_notebook_mode(connected=True)
%matplotlib inline
```

In [62]:
```
## read the dataset

path = '../input/imaterialist-challenge-fashion-2018/train.json'

inp = open(path).read()
inp = json.loads(inp)
```

1. Descriptive Statistics

1.1 Total Images and Total Labels Present in the dataset

In [63]:
```
# how many images
total_images = len(inp['images'])

# how many labels
all_annotations = []
for each in inp['annotations']:
    all_annotations.extend(each['labelId'])
total_labels = len(set(all_annotations))

print ("Total Images in the dataset: ", total_images)
print ("Total Labels in the dataset: ", total_labels)
```

1.2 Top Labels present in the dataset

In [64]:
```
# Top Labels in the dataset
label_dist = Counter(all_annotations)

xvalues = list(label_dist.keys())
yvalues = list(label_dist.values())

trace1 = go.Bar(x=xvalues, y=yvalues, opacity=0.8, name="year count", marker=dict(color='rgba(20, 20, 2
0, 1)'))
layout = dict(height=400, title='Distribution of different labels in the dataset', legend=dict(orientat
ion="h"));

fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);
```

In [65]:
```
def get_images_for_labels(labellist):
    image_ids = []
    for each in inp['annotations']:
        if all(x in each['labelId'] for x in labellist):
            image_ids.append(each['imageId'])
            if len(image_ids) == 2:
                break
    image_urls = []
    for each in inp['images']:
        if each['imageId'] in image_ids:
            image_urls.append(each['url'])
    return image_urls
```

In [66]:
```
# most common labels

temps = label_dist.most_common(10)
labels = ["Label: "+str(x[0]) for x in temps]
values = [x[1] for x in temps]

trace1 = go.Bar(x=labels, y=values, opacity=0.7, name="year count", marker=dict(color='rgba(120, 120, 1
20, 0.8)'))
layout = dict(height=400, title='Top 10 Labels in the dataset', legend=dict(orientation="h"));

fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);
```

In [67]:
```
for labelpair in labels:
    labelpr = labelpair.replace("Label: ","").split("-")
    imgs = get_images_for_labels(labelpr)

#     headerhtml = """<b>"""+ str(labelpair) +"""</b><br>"""
#     display(HTML(headerhtml))
#     imghtml = ""
#     for img in imgs:
#         imghtml += "<img src="+img+" width=200px; height=200px; style='float:left'>"
#     display(HTML(imghtml))
```

1.3 Most Common Co-Occuring Labels in the dataset

In [68]:
```
# Most Commonly Occuring Labels

def cartesian_reduct(alist):
    results = []
    for x in alist:
        for y in alist:
            if x == y:
                continue
            srtd = sorted([int(x),int(y)])
            srtd = " AND ".join([str(x) for x in srtd])
            results.append(srtd)
    return results

co_occurance = []
for i, each in enumerate(inp['annotations']):
    prods = cartesian_reduct(each['labelId'])
    co_occurance.extend(prods)
```

In [69]:
```
coocur = Counter(co_occurance).most_common(10)
labels = list(reversed(["Label: "+str(x[0]) for x in coocur]))
values = list(reversed([x[1] for x in coocur]))

trace1 = go.Bar(x=values, y=labels, opacity=0.7, orientation="h", name="year count", marker=dict(color=
'rgba(130, 130, 230, 0.8)'))
layout = dict(height=400, title='Most Common Co-Occuring Labels in the dataset', legend=dict(orientatio
n="h"));

fig = go.Figure(data=[trace1], layout=layout);
iplot(fig);
```

1.4 Images with Maximum Labels

In [70]:
```
def get_image_url(imgid):
    for each in inp['images']:
        if each['imageId'] == imgid:
            return each['url']

srtedlist = sorted(inp['annotations'], key=lambda d: len(d['labelId']), reverse=True)
```

In [71]:
```
for img in srtedlist[:5]:
    iurl = get_image_url(img['imageId'])
    labelpair = ", ".join(img['labelId'])
#     headerhtml =
#     display(HTML(headerhtml))
    imghtml = """<b> Labels: """+ str(labelpair) +"""</b><br>""" + "<img src="+iurl+" width=200px; styl
e='float:left'>"
    display(HTML(imghtml))
```

1.5 Images with Single Label

In [72]:
```
# How many images are labelled with only 1 label

for img in srtedlist[-5:]:
    iurl = get_image_url(img['imageId'])
    labelpair = ", ".join(img['labelId'])
#     headerhtml =
#     display(HTML(headerhtml))
    imghtml = """<b> Label: """+ str(labelpair) +"""</b><br>""" + "<img src="+iurl+" width=200px; heigh
t=200px; style='float:left'>"
    display(HTML(imghtml))
```

2. Colors Used in the Images

2.1 Common Average Color of the Images

In [14]:
```
import urllib
from io import StringIO

def compute_average_image_color(img):
    width, height = img.size
    count, r_total, g_total, b_total = 0, 0, 0, 0
    for x in range(0, width):
        for y in range(0, height):
            r, g, b = img.getpixel((x,y))
            r_total += r
            g_total += g
            b_total += b
            count += 1
    return (r_total/count, g_total/count, b_total/count)
```

In [21]:
```
import os
imgpath = '../input/sampleimages/top_images/top_images/'
read_from_disk = True

if read_from_disk:
    srtedlist = os.listdir(imgpath)
else:
    srtedlist = sorted(inp['annotations'], key=lambda d: len(d['labelId']), reverse=True)
```

In [23]:
```
average_colors = {}
for img in srtedlist[:10]:
    if read_from_disk:
        img = Image.open(imgpath + img)
    else:
        iurli = get_image_url(img['imageId'])

        ## download the images
        # filename = iurli.split("/")[-1].split("-large")[0]
        # urllib.urlretrieve(iurli, "top_images/"+filename)

        file = cStringIO.StringIO(urllib.urlopen(iurli).read())
        img = Image.open(img)

    average_color = compute_average_image_color(img)
    if average_color not in average_colors:
        average_colors[average_color] = 0
    average_colors[average_color] += 1
```

In [59]:
```
for average_color in average_colors:
    average_color1 = (int(average_color[0]),int(average_color[1]),int(average_color[2]))
    image_url = "<span style='display:inline-block; min-width:200px; background-color:rgb"+str(average_
color1)+";padding:10px 10px;'>"+str(average_color1)+"</span>"
#     print (image_url)
    display(HTML(image_url))
```

2.2 Most Dominant Colors Used in the Images

In [39]:
```
## top used colors in images
from colorthief import ColorThief
import urllib

pallets = []
for img in srtedlist[:10]:

    if read_from_disk:
        img = imgpath + img
    else:
        iurli = get_image_url(img['imageId'])

        ## download the images
        # filename = iurli.split("/")[-1].split("-large")[0]
        # urllib.urlretrieve(iurli, "top_images/"+filename)

        file = cStringIO.StringIO(urllib.urlopen(iurli).read())
        img = Image.open(img)

    color_thief = ColorThief(img)
    dominant_color = color_thief.get_color(quality=1)

    image_url = "<span style='display:inline-block; min-width:200px; background-color:rgb"+str(dominant
_color)+";padding:10px 10px;'>"+str(dominant_color)+"</span>"
    display(HTML(image_url))

    palette = color_thief.get_palette(color_count=6)
    pallets.append(palette)
```

2.3 Common Color Pallets of the Images

In [26]:
```
for pallet in pallets:
    img_url = ""
    for pall in pallet:
        img_url += "<span style='background-color:rgb"+str(pall)+";padding:20px 10px;'>"+str(pall)+"</s
pan>"
    img_url += "<br>"
    display(HTML(img_url))
    print
```

3. Object Detection using TensorFlow API

I have used tensorflow API for object detection the code is given in the following cell.

In [27]:
```
### UNCOMMENT THE FOLLOWING LINE AFTER DOWNLOADING THE UTILS FROM THIS LINK - https://github.com/tensor
flow/models/tree/master/research/object_detection/utils

# from utils import label_map_util

def DOWNLOAD_MODELS():
    MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
    MODEL_FILE = MODEL_NAME + '.tar.gz'
    DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
    PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
    PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

    opener = urllib.request.URLopener()
    opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
    tar_file = tarfile.open(MODEL_FILE)
    for file in tar_file.getmembers():
        file_name = os.path.basename(file.name)
        if 'frozen_inference_graph.pb' in file_name:
            tar_file.extract(file, os.getcwd())

def detect_object(filename):

    def img2array(img):
        (img_width, img_height) = img.size
        return np.array(img.getdata()).reshape((img_width, img_height, 3)).astype(np.uint8)

    categories, probabilities = [], []
    PATH_TO_CKPT = 'frozen_inference_graph.pb'
    PATH_TO_LABELS = 'mscoco_label_map.pbtxt'
    detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')


    label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
    categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=100, use_dis
play_name=True)
    category_index = label_map_util.create_category_index(categories)

    with detection_graph.as_default():
        with tf.Session(graph=detection_graph) as sess:
            image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
            detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
            detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
            detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
            num_detections = detection_graph.get_tensor_by_name('num_detections:0')

            image = Image.open(filename)
            image_np = img2array(image)
            image_np_expanded = np.expand_dims(image_np, axis=0)
            (boxes, scores, classes, num) = sess.run([detection_boxes, detection_scores, detection_clas
ses, num_detections], feed_dict={image_tensor: image_np_expanded})
            for index,value in enumerate(classes[0]):
                if float(scores[0,index]) > 0.1:
                    temp =  category_index.get(value)['name']
                    if temp not in categories:
                        categories.append(temp)
                        probabilities.append(scores[0,index])
    return categories, probabilities
```

In [28]:
```
## UNCOMMENT THE FOLLOWING LINES TO RUN THE OBJECT DETECTION MODEL AND SAVE THE RESULTS

# for img in srtedlist[:10]:
#     iurli = get_image_url(img['imageId'])

#     file = cStringIO.StringIO(urllib.urlopen(iurli).read())
#     objects = detect_object(file)
```

Reference: TensorFlow Object Detection Notebook

Pre-Trained Models Reference: PreTrained Models

Link to download the Utils: https://github.com/tensorflow/models/tree/master/research/object_detection/utils

Since it would have taken a lot of time on kaggle kernals, I have pre-computed the objects in my local machine.

In [29]:
```
objpath = '../input/precomputedobjects/objects.txt'

objs = open(objpath).read().strip().split("\n")
colors = [_ for _ in objs if "color" in _]
non_colors = [_ for _ in objs if "color" not in _]
```

3.1 Top Colors detected using Object detection

In [30]:
```
txt = ""
for i, color in enumerate(Counter(colors).most_common(100)):
    txt += (color[0] + " ")
txt = txt.replace("color", " ")
wordcloud = WordCloud(max_font_size=50, width=600, height=300, background_color='white').generate(txt)
plt.figure(figsize=(15,8))
plt.imshow(wordcloud)
plt.title("Top Colors Used in the images", fontsize=15)
plt.axis("off")
plt.show()
```

3.2 Top Objects Detected in the images

In [31]:
```
txt = ""
for i, color in enumerate(Counter(non_colors).most_common(100)):
    txt += color[0]+" "
wordcloud = WordCloud(max_font_size=50, width=600, height=300).generate(txt)
plt.figure(figsize=(15,8))
plt.imshow(wordcloud)
plt.title("Top Objects Detected in the images", fontsize=15)
plt.axis("off")
plt.show()
```

Thanks for viewing the notebook. Hope You liked it, if liked it please upvote.

In [ ]: