Configuration Manual

E-commerce product match detection using Product Text and Images

Hari Krishnan Kannan
x19225547@student.ncirl.ie

1 Introduction

The configuration manual document details the components for the project implementation which is designed for identifying e-commerce product similarity detection using product text and images. This document provides the brief steps required for successful implementation of the research project.

2 System Configuration

The system hardware requirements and configuration needed for implementation is as follows:

2.1 Requirement for Hardware

Hardware	Configurations
System	Lenovo Legion Y740
Operating System	Windows 10 (64bit)
RAM	16 GB
Hard Disk	1 TB (Solid State Drive)
Graphics Card	NVIDIA RTX 2060 (6 GB)
Processor	Intel Core I7-9750

Figure 1 Hardware Details

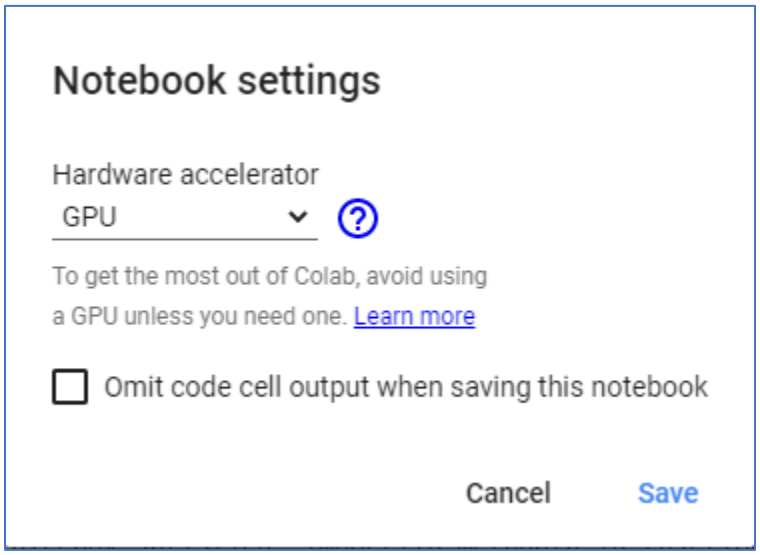
2.2 Requirement for Software

The varies software requirements are details below.

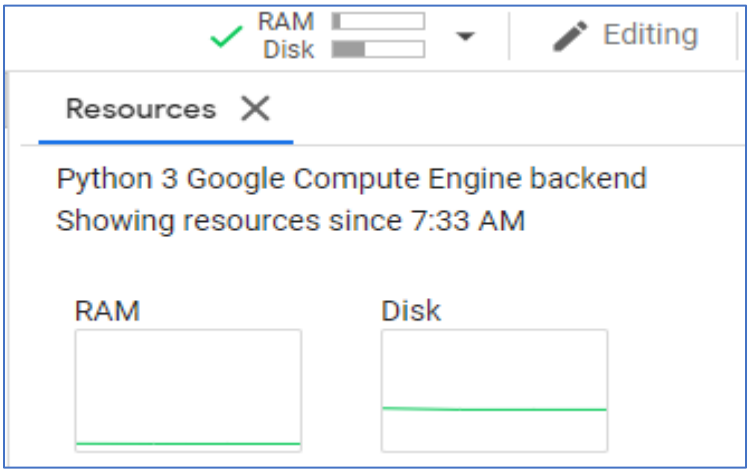
- In the operating system details used for windows 10

Figure 2 Operating System Details

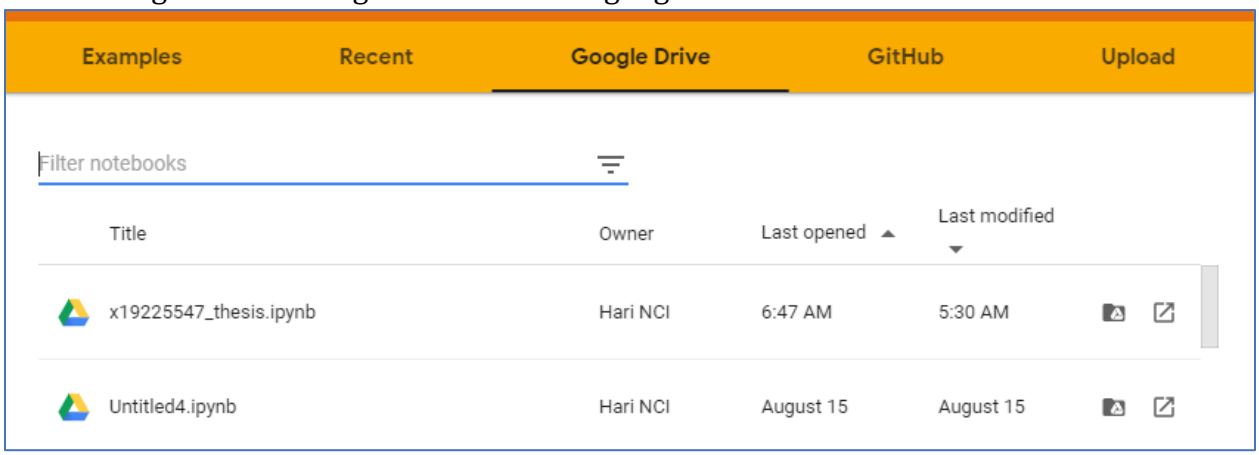
- In Google Colab, the hardware accelerator at the run time is used as GPU



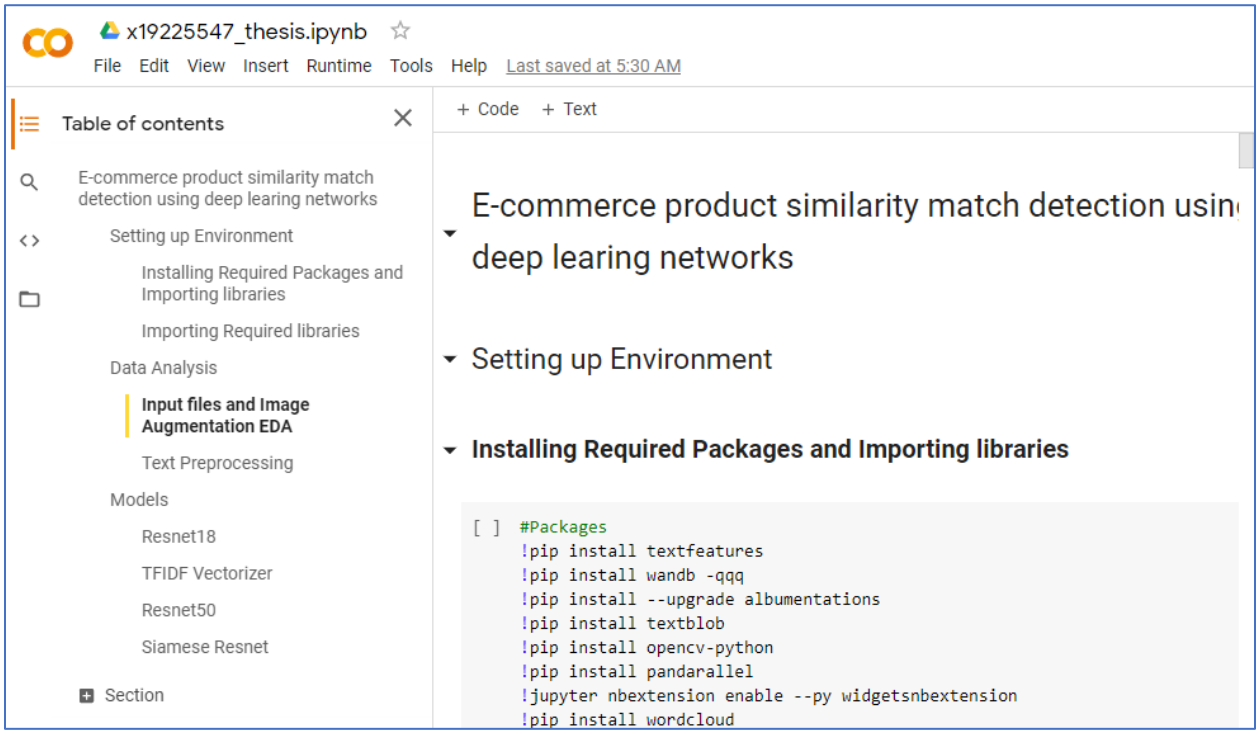
- The disk and ram space utilization during run time in google colab



- Google Colab storage is connected to google drive



- The notebook used to run the code is in format .ipynb



3 Implementation of the project

3.1 Data Extraction

The data is extracted from google drive location

3.2 Installing Required Packages

Installing Required Packages and Importing libraries

```
[ ]  #Packages
     !pip install textfeatures
     !pip install wandb -qqq
     !pip install --upgrade albumentations
     !pip install textblob
     !pip install opencv-python
     !pip install pandarallel
     !jupyter nbextension enable --py widgetsnbextension
     !pip install wordcloud
     !pip install --upgrade --force-reinstall --no-deps kaggle
     !pip install tensorflow_addons
     !pip install keras_toolkit
     !pip install torchvision
```

3.3 Installing Required Libraries

Importing Required libraries

```
[ ]  # Libraries
     import os
     import cv2
     from PIL import Image
     import string
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import albumentations as alb
     from albumentations.augmentations.transforms import RGBShift, RandomGamma,
     from albumentations.augmentations.geometric.rotate import Rotate
     from albumentations.augmentations.crops.transforms import  CenterCrop, Ran
     import nltk
     nltk.download('stopwords')
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from nltk.stem import PorterStemmer, WordNetLemmatizer
```

3.4 Data Loading from google drive

Mounting Storage

+ Code + Text

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
```

```
[ ] os.chdir('/content/drive/My Drive/Thesis/')
    path = os.getcwd()
```

3.4 Data Preparation

```
# Prepare data
n = len(groups_df.iloc[::100]["group"])
labels = ["id_" + str.zfill(str(i), 3) for i in range(n)]
values = groups_df.iloc[::100]["count"]

data = [[label, val] for (label, val) in zip(labels, values)]

# Create Table & .log() the plot
table = wandb.Table(data=data, columns = ["Group_ID", "count"])
wandb.log({"group_chart" : wandb.plot.bar(table, "Group_ID", "count",
                                          title="Group Count Distribution")})
```

```
def get_group_info(group_name):
    '''This function shows a sample of 6 images within a group.
    group_name: a string representing the desired group code'''

    # Retrieve a sample of 6 images from this group
    sample_names = train_df[train_df["label_group"] == group_name]["image"].\
                   sample(6, random_state=24).values
    sample_text = train_df[train_df["label_group"] == group_name]["title"].\
                   sample(1, random_state=1).values

    # Plot
    fig = plt.figure(figsize=(16, 8))
    plt.suptitle(f"Group: {sample_group}", fontsize=20)
    plt.title(f"{sample_text}", fontsize=15)
    plt.axis("off")
```

3.5 Data Pre-processing

3.5.1 Text pre-processing

The common text processing are remove punctuation, white space, stopwords etc

Text Preprocessing

```
[ ]  # Original
     original = train_df["title"][100]
     print(color.END + "Before:" + color.END, original, "\n")
     #lower case
     lower = original.lower()
     print(color.END + "Lower case:" + color.END, lower)
     #Remove punctuation
     punct = lower.translate(str.maketrans('','',string.punctuation))
     print(color.END + "Remove punctuation:" + color.END, punct)
     #Remove whitespaces
     whitespaces = punct.strip()
     print(color.END + "Remove whitespaces:" + color.END, whitespaces)
     #Tokenize words
     tokenize = word_tokenize(whitespaces)
     print(color.END + "Tokenized:" + color.END, tokenize)
     #stopwords
     sw = [word for word in tokenize if not word in stopwords.words()]
     print(color.END + "Remove stopwords:" + color.END, sw)
```

3.5.2 Image Augmentation

The common image augmentation techniques used are rotate, flip, invert, crop etc..

```
def display_augmentations(path):

    # Read in original image
    original = cv2.imread(path)
    original = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)

    # Transformations
    transform_rgb = alb.Compose([ RGBShift(g_shift_limit=50, always_apply=True) ])
    transform_hsv = alb.Compose([ HueSaturationValue(hue_shift_limit=100, sat_shift_limit=60, always_apply=True) ])
    transform_hf = alb.Compose([ HorizontalFlip(always_apply=True) ])
    transform_clahe = alb.Compose([ CLAHE(clip_limit=10.0, always_apply=True) ])
    transform_rc = alb.Compose([ RandomCrop(height=300, width=300, always_apply=True) ])
    transform_rg = alb.Compose([ RandomGamma(gamma_limit=(200, 400), always_apply=True) ])
    transform_rot = alb.Compose([ Rotate(limit=90, always_apply=True) ])
    transform_cc = alb.Compose([ CenterCrop(height=450, width=450, always_apply=True) ])
    transform_mb = alb.Compose([ MedianBlur(blur_limit=103, always_apply=True) ])
    transform_vf = alb.Compose([ VerticalFlip(always_apply=True) ])
    transform_ii = alb.Compose([ InvertImg(always_apply=True) ])

    # Apply transformations
    transformed_rgb = transform_rgb(image=original)["image"]
    transformed_hsv = transform_hsv(image=original)["image"]
    transformed_hf = transform_hf(image=original)["image"]
    transformed_clahe = transform_clahe(image=original)["image"]
    transformed_rc = transform_rc(image=original)["image"]
    transformed_rg = transform_rg(image=original)["image"]
    transformed_rot = transform_rot(image=original)["image"]
```

icon_transcript_processing

3.5.3 Feature Extraction

The features like words, characters, average word length, stopwords, numbers etc..

```
def extract_title_features(df_prep):
    '''Extracts features from the unprocessed title column.'''

    # Extract Features
    df_prep = textfeatures.word_count(df_prep, "title", "words")
    df_prep = textfeatures.char_count(df_prep, "title", "characters")
    df_prep = textfeatures.avg_word_length(df_prep, "title", "avg_word_length")
    df_prep = textfeatures.stopwords_count(df_prep, "title", "stopwrds")
    df_prep = textfeatures.numerics_count(df_prep, "title", "numbers")

    return df_prep

train_df_prep = extract_title_features(df_prep=train_df_prep)

title_features = ['words', 'characters', 'avg_word_length',
                  'stopwrds', 'numbers']
```

3.5.5 Duplicates in dataset

The data duplicates are identified by label counts based on image id.

```
n = len(image_count_duplicates.iloc[::16]["image"])
labels = ["id_" + str.zfill(str(i), 2) for i in range(n)]
values = image_count_duplicates.iloc[::16]["count"]

data = [[label, val] for (label, val) in zip(labels, values)]

# Create Table & .log() the plot
table = wandb.Table(data=data, columns = ["Image_ID", "count"])
wandb.log({"image_chart" : wandb.plot.bar(table, "Image_ID", "count",
                                          title="Duplicated Images: How many apparitions?")})
```

3.6 Models

The list of models used are TF-Vectorizer, ResNet-18, ResNet-50, Siamese ResNet-50, TF-IDF + ResNet 18

3.6.1 TFIDF Vectorizer

```
tfidf_model = TfidfVectorizer(stop_words=None, binary=True, max_features=55000)
text_embeddings = tfidf_model.fit_transform(train.title).toarray()
print(text_embeddings.shape)
```

```
(34250, 25069)
```

```
text_embeddings = torch.from_numpy(text_embeddings)
text_embeddings = text_embeddings.cuda()
```

```
predictions = []
CHUNK = 1024*4
print('Search similar titles')
CTS = len(train)//CHUNK
if len(train)%CHUNK!=0: CTS += 1
CTS index = 0
```

3.6.2 Modelling ResNet 18

```
predictions = []
CHUNK = 1024*4

print('Search similar images')
CTS = len(imagefeat)//CHUNK
if len(imagefeat)%CHUNK!=0: CTS += 1
for j in range( CTS ):
    a = j*CHUNK
    b = (j+1)*CHUNK
    b = min(b, len(imagefeat))
    print('chunk',a,'to',b)
    distances = torch.matmul(imagefeat, imagefeat[a:b].T).T
    distances = distances.data.cpu().numpy()
    for k in range(b-a):
        IDX = np.where(distances[k,]>0.95)[0][:]
        o = train.iloc[IDX].posting_id.values
        #o = train.iloc[cupy.asnumpy(IDX)].posting_id.values
        predictions.append(o)
```

3.6.3 ResNet-50

```
train['resnet50_pred'] = predictions

if CV:
    train['f1_score'] = train.apply(getMetric('resnet50_pred'),axis=1)
    print('CV Score for Resnet50',train.f1_score.mean())
```

```
if CV:
    label_count = train.groupby(['label_group']).size().reset_index()
    label_count.columns = ['label_group', 'count']
    label_count.sort_values(by='count', ascending=False, inplace=True)
    label_count
```

```
if CV:
    t1 = train[train['label_group'] == 1163569239].index
    t2 = train[train['label_group'] == 159351600].index
    cts = torch.matmul(imagefeat[t1], imagefeat[t2].T).T
    cts = cts.data.cpu().numpy()
    vs = []
    for dist in cts:
        vs.extend(dist)
```

3.6.4 Siamese ResNet

```
class SiameseModel(Model):
    """The Siamese Network model with a custom training and testing loops.

    Computes the triplet loss using the three embeddings produced by the
    Siamese Network.

    The triplet loss is defined as:
       L(A, P, N) = max(‖f(A) - f(P)‖² - ‖f(A) - f(N)‖² + margin, 0)
    """

    def __init__(self, siamese_network, margin=0.5):
        super(SiameseModel, self).__init__()
        self.siamese_network = siamese_network
        self.margin = margin
        self.loss_tracker = metrics.Mean(name="loss")

    def call(self, inputs):
        return self.siamese_network(inputs)

    def train_step(self, data):
```

3.6.5 TF-IDF + ResNet 18

```
if CV:
    tmp = train.groupby('label_group').posting_id.agg('unique').to_dict()
    train['target'] = train.label_group.map(tmp)
    train['cv'] = train.apply(to_print,axis=1)
    train['f1_score'] = train.apply(getMetric('cv'),axis=1)
    print('Resnet18 TFIDF CV Score =', train.f1_score.mean() )

train['matches'] = train.apply(to_file,axis=1)
```

```
Resnet18 TFIDF CV Score = 0.7341838727212323
```

The name of the python notebook file is x19225547_thesis.ipynb