National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Aafaq Iqbal Khan
Student ID: X20108851

School of Computing
National College of Ireland

Supervisor: Cristina Hava Muntean

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Aafaq Iqbal Khan
Student ID: …….X2010851

Programme: …….MSc Data Analytics…… **Year:** …….2021

Module: ……Research project thesis

Lecturer: ….. Cristina Hava Muntean
Submission Due Date: ……31/01/2022

Project Title: Text and Image Based Multi-model Fashion Image Retrieval system.

Word Count: …… **Page Count:** ……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Aafaq Iqbal Khan……

Date: ……15/12/2021……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aafaq Iqbal Khan
Student ID: X20108851

1 Introduction

The configuration manual explains the instruction and guidelines to implement and reproduce the project. The document comprises of a sequence of actions done with hardware and software required for the implementation of research "*Text and Image Based Multi-model Fashion Image Retrieval system*". Later in this configuration manual, section 2 mentions the required python packages and libraries for implementation. The section 3 provides the information about the Data preparation for Experiments. The Google Colaboratory Environment Setup is described in section 4. The section 5 explains the implementation of project. We try to make this configuration manual is as simple as so that anyone from any background can reproduce our research work. For this purpose, we give a quick steps at the end in section 6.

2 Required Packages and Libraries

The project is implemented by using python 3. Following python packages are required to execute the code. The fig. 1 shows all required packages.

- Numpy Version: 1.19.5
- Pandas Version: 1.1.5
- Matplotlib Version: 3.2.2
- Pytorch (Paszke, A. et al., 2019)
- Pickle
- Argparse.

```
import argparse
import sys
import time
import numpy as np
from torch.utils.tensorboard import SummaryWriter
import torch
import torch.utils.data
from torch.utils.data import dataloader
import torchvision
from tqdm import tqdm as tqdm
import pdb
import time
import pickle
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure, imshow, axis
from matplotlib.image import imread
from matplotlib.ticker import MaxNLocator
import PIL
import datasets
import img_text_composition_models
```

Fig.1. Importing packages

A special package need to be installed for the implementation of the Efficient-Net model. The command in fig.2 shows the successful installation.

```
pip install --upgrade efficientnet-pytorch

Collecting efficientnet-pytorch
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from efficientnet-pytorch) (1.10.0+cu111)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->efficientnet-pytorch) (3.10.0.2)
Building wheels for collected packages: efficientnet-pytorch
  Building wheel for efficientnet-pytorch (setup.py) ... done
  Created wheel for efficientnet-pytorch: filename=efficientnet_pytorch-0.7.1-py3-none-any.whl size=16446 sha256=f371109a9379c740b4c276819d669f4cd56645481ac9cddf0bfe9b9dde2b1dcf
  Stored in directory: /root/.cache/pip/wheels/0e/cc/b2/49e74588263573ff778da58cc99b9c6349b496636a7e165be6
Successfully built efficientnet-pytorch
Installing collected packages: efficientnet-pytorch
Successfully installed efficientnet-pytorch-0.7.1
```

Fig.2 Efficient-Net Package installation

3 Data preparation for Experiments

This section will explain how to upload data in google drive that is connected with Colab. Firstly, unzip the *artefact.zip* file, the structure of the files should be similar as shown in fig 3. Here "*images"* folder contains the image dataset and "*datasets.py*", "*img_text_composition_models.py",* and "*text_model.py*" are the python files in which required classes and functions are written. The tree like structure shown in figure is built by (Friend, N. 2020)

```
.
└── Code Artfact/
    ├── annotation/
    │   ├── ref_caption_dress_train.json
    │   ├── ref_caption_dress_test.json
    │   ├── ref_caption_shirt_train.json
    │   └── ref_caption_shirt_test.json
    ├── public_split/
    │   ├── split.dress.train.json
    │   ├── split.dress.test.json
    │   ├── split.shirt.train.json
    │   └── split.shirt.test.json
    ├── images
    ├── datasets.py
    ├── img_text_composition_models.py
    ├── simplevocab.json
    ├── text_model.py
    ├── Dress_Notebook.ipynb
    └── shirt_Notebook.ipynb
```

Fig.3 Artefact folder structure

4 Google Colaboratory Environment Setup

For implementation of the project Google Colab is used. As our dataset Fashion–IQ has large number of images so local machine is not a feasible solution. In the google Colab, jupyter notebooks can be run freely. All of the fundamental packages or libraries are already installed in the environment. The free version of Colab provides ~12 GB Ram with 38 GB of remote hard drive.

1. To use the Colab go to https://colab.research.google.com/ and sign in with any Gmail account

2. There are three mode of run-type are given i.e. None, GPU, and TPU. We will use GPU and select it as shown in fig. 4

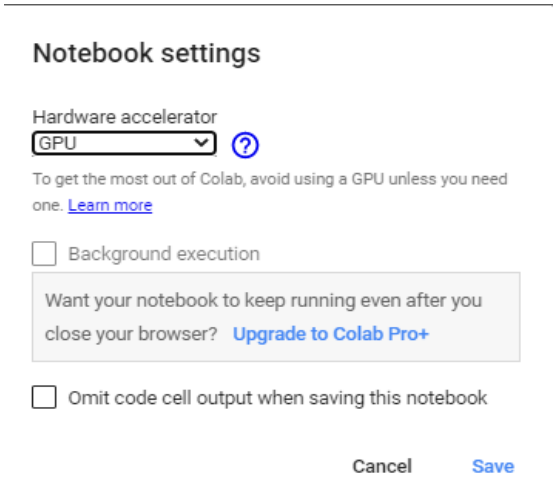

Fig.4 Run-Type selection

3. Now mount your google drive in Colab notebook as shown in fig. 5. The path directory need to be change if required. Upload the code and data file to this directory. The details of the files are explained in section 2.

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

     Mounted at /content/drive/

[ ]  import os
     os.chdir('/content/drive/MyDrive/without/data')
```

Fig.5 Drive Mount.

5 Implementation of project

After mounting the google drive in Colab and importing the required packages and files, set the system parameters and variables with appropriate values fig. (6)

```
def parse_opt():
  """Parses the input arguments."""
  parser = argparse.ArgumentParser()
  parser.add_argument('-f', type=str, default='')
  parser.add_argument('--dataset', type=str, default='fashioniq')
  parser.add_argument('--root_dir', type=str, default='./{}')
  parser.add_argument('--log_dir', type=str, default='./runs')
  parser.add_argument('--model', type=str, default='imgonly')
  parser.add_argument('--img_encoder', type=str, default='efficientnet')
  parser.add_argument('--text_encoder', type=str, default='lstm')
  parser.add_argument('--embed_dim', type=int, default=1024)
  parser.add_argument('--optimizer', type=str, default='Adam') # SGD Adam
  parser.add_argument('--learning_rate', type=float, default=1e-4)
  parser.add_argument(
      '--learning_rate_decay_patient', type=int, default=5)
  parser.add_argument('--eval_frequency', type=int, default=1)
  parser.add_argument('--lr_div', type=float, default=0.5) # 0.1
  parser.add_argument('--batch_size', type=int, default=32)
  parser.add_argument('--weight_decay', type=float, default=1e-6)
  parser.add_argument('--num_epoch', type=int, default=15)
  parser.add_argument('--loss', type=str, default='batch_based_classification')
  parser.add_argument('--loader_num_workers', type=int, default=4)
  #parser.add_argument('--is_test', default=False, action='store_true')
  #parser.add_argument('--return_test_rank', default=False, action='store_true')
  #parser.add_argument('--resume_file', default=None)
  parser.add_argument('--resume_file', default='./runs/best_checkpoint.pth')

  args = parser.parse_args()
  return args
opt = parse_opt()
```

Fig. 6 System parameters

After uploading the data on drive, now we should pre-process and transform the train and test datasets in fig. 7. A class "*FashionIQ"* is implemented in "*datasets.py*".

```
def load_dataset(opt):
  """Loads the input datasets."""
  print('Reading dataset ', opt.dataset)
  if opt.dataset == 'fashioniq':
    trainset = datasets.FashionIQ(
        anno_dir=opt.root_dir.format('annotation'),
        image_dir=opt.root_dir.format('images'),
        split_dir=opt.root_dir.format('public_split'),
        category= ['dress'],
        split='train',
        transform=torchvision.transforms.Compose([
            torchvision.transforms.Resize(256),
            torchvision.transforms.RandomCrop(224),
            torchvision.transforms.RandomHorizontalFlip(),
            torchvision.transforms.ToTensor(),
            torchvision.transforms.Normalize([0.485, 0.456, 0.406],
                                             [0.229, 0.224, 0.225])
        ]))
  else:
    print('Invalid dataset', opt.dataset)
    sys.exit()

  print('trainset size:', len(trainset))
  #print('testset size:', len(testset))
  return trainset
```

Fig. 7. Train dataset pre-processing and transformation

As dataset is large and takes too much time for pre-processing and transformation of the images so it better to make pickle file for future use as shown in fig. 8

```
pickle_out = open("trainset_dress.pickle","wb")
pickle.dump(trainset, pickle_out)

pickle_out1 = open("testset_dress.pickle","wb")
pickle.dump(testset, pickle_out1)

pickle_in = open("trainset_dress.pickle","rb")
trainset = pickle.load(pickle_in)
pickle_in1 = open("testset_dress.pickle","rb")
testset = pickle.load(pickle_in1)
```

Fig. 8 Pickle file code

After pre-processing and transformation of data, training of model will be done. Fig .9 shows training result of one epoch only for Text-only method, other methods will be trained in a same manner.

```
model, optimizer = create_model_and_optimizer(opt, None)
# if opt.resume_file != None:
#   load_state_dicts(opt, model)

start_time = time.time()
df_loss_TextOnly, df_R_mean_TextOnly= train_loop(opt, logger, trainset, model, optimizer)
Dress_time.append(time.time() - start_time)
logger.close()
df_loss_TextOnly.head()
```

```
Creating model and optimizer for textonly
Loaded pretrained weights for efficientnet-b0
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: TI
  cpuset_checked))
Begin training
Training for epoch 0: 100%|██████████| 187/187 [06:19<00:00,  2.03s/it]
It 187 epoch 0 Elapsed time 379.5701
    Loss batch_based_classification 3.1507
    Loss total training loss 3.1507
100%|██████████| 780/780 [00:00<00:00, 1148.98it/s]
100%|██████████| 4197/4197 [00:25<00:00, 166.25it/s]
     dress_r1 0.9
     dress_r3 2.82
     dress_r5 4.62
     dress_r10 6.79
     dress_r20 11.54
     dress_r50 20.9
     rmean 7.93
```

Fig. 9 Training of model

A test function is defined that will be used by the each model for evaluation on test dataset fig.10.

```
def test(opt, testset, model):
  print('Begin testing')
  tests = []
  rsum = 0
  for dataname in testset.data_name:
    t, sims, nn_result = testResults(opt, model, testset, dataname)
    np.save(opt.log_dir + '/val.{}.{}.scores.npy'.format(dataname, opt.model), sims)
    for metric_name, metric_value in t:
      tests += [(metric_name, metric_value)]
      rsum += metric_value

  tests += [('RecallMean', rsum / 6)]
  for metric_name, metric_value in tests:
    print('    ', metric_name, round(metric_value, 2))
  print('Finished testing')
  return nn_result
```

```
model, optimizer = create_model_and_optimizer(opt, None)
state_dicts = torch.load(opt.resume_file, map_location=lambda storage, loc: storage)['model_state_dict']
model.load_state_dict(state_dicts)
nn_result_TextOnly=test(opt, testset, model)
```

```
Begin testing
100%|██████████| 386/386 [00:00<00:00, 1086.28it/s]
100%|██████████| 1687/1687 [00:10<00:00, 166.53it/s]
     dress_r1 3.11
     dress_r3 7.51
     dress_r5 10.36
     dress_r10 15.28
     dress_r20 25.39
     dress_r50 41.45
     RecallMean 17.18
Finished testing
```

Fig. 10 Test function and results on test data

Each model retrieves the top scored similar images that related to query image and query text. The model retrieves only the id of the images and we need to define some function to display the retrieve results along with query image and query text. Fig. 11 shows the defined functions and fig 12 shows the qualitative result.

```
def get_result(array_list):
  result_files = []
  print(len(array_list))
  for i in range(len(array_list)):
    for t in test_queries:
      if t['source_img_id']==array_list[i]:
        idx = t['source_img']
        image_dir=opt.root_dir.format('images')
        img_path = image_dir + '/' +idx+ ".jpg"
        result_files += [img_path]

  return result_files
def show_result_all(result_files):


    fig = figure(figsize = (25,25))
    number_of_files = len(result_files)

    for i in range(5):
        a=fig.add_subplot(1,5,i+1)
        image = imread(result_files[i])
        imshow(image)
        if i==4:
          plt.title("Results ")
        axis('off')
```

Fig. 11 Qualitative results display function

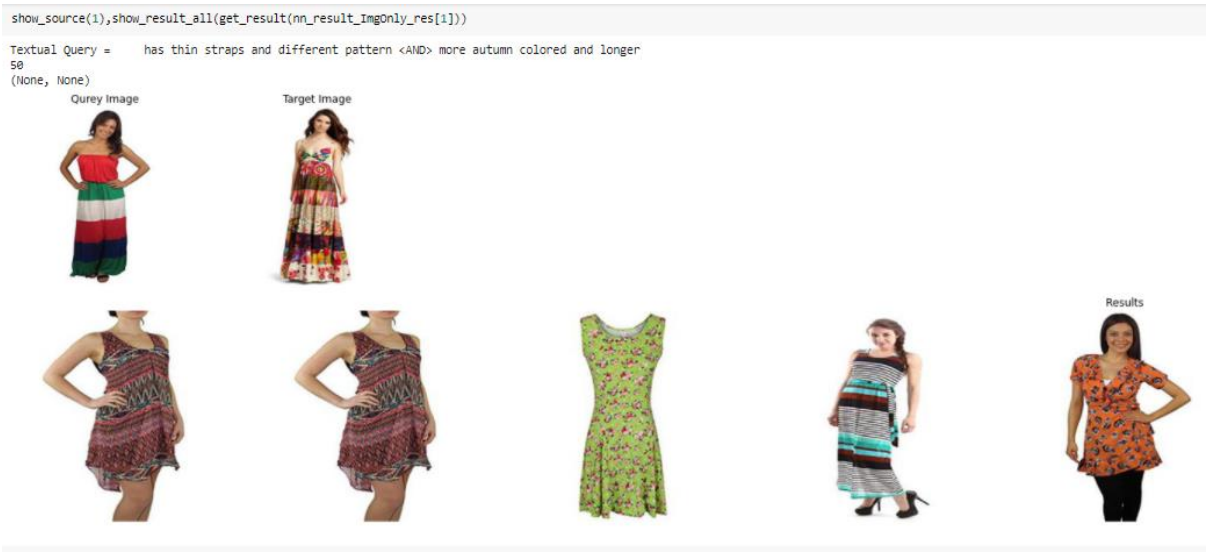

Fig. 12. Qualitative results

We tried to show the important steps and code here, but it's not possible to show complete code here due to length of code.

6 Quick Steps to Reproduce the Project.

Step-1

Unzip the artefact file and upload in intended directory of your google drive. The folder structure of unzipped folder should be same as shown in fig. 3.

Step-2

Go to https://colab.research.google.com/ and sign in with Gmail account.

Step-3

Now go back to google drive and right click on notebook file *"Dress_Notebook.ipynb"* and select open with "*Google Colaboratory"* as shown in fig 13.

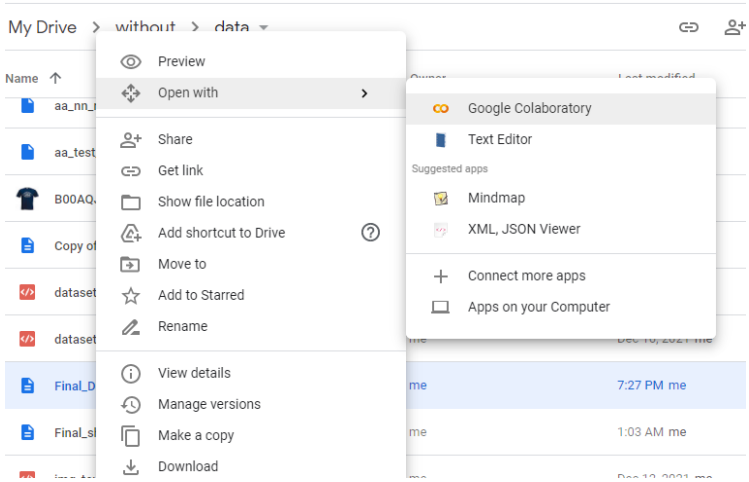

Fig. 13 open notebook files

Step-4

Mount your drive in Colab as shown in fig 5. The path directory should be change as per your path if required. Make sure GPU is selected as run-type as shown in fig. 4

Step-5

After uploading the files on google drive and mounting it with Colab, now everything is in-line to execute the notebook just run every cell one by one. All four methods text-only, image-only Efficient-net, image-only Resnet-50, and TIRG will be executed one by one.

Step-6

Same instruction repeat for "*shirt_Notebook.ipynb*".

References

Friend, N. (2020) tree-online: An online tree-like utility for generating ASCII folder structure diagrams. Written in TypeScript and React.

Paszke, A. et al., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.