National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Bharadwaj Ravur
Student ID: x21194521

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Bharadwaj Ravur
Student ID:	x21194521
Programme:	MSc in Data Analytics **Year:** 2023
Module:	MSc Research Project
Supervisor:	Abubakr Siddig
Submission Due Date:	18/09/2023
Project Title:	Sorting Clothes using Image Segmentation and Object Detection
Word Count : 2036	**Page Count: 14**

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: Bharadwaj Ravur

Date: 18/09/2023

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 for Sorting Clothes using Image Segmentation and Object Detection

Bharadwaj Ravur
x21194521

1 Introduction

This configuration manual provides detailed instructions for setting up and configuring the clothing classification and object detection model based on the YOLOv5 and Mask R-CNN architectures. This manual includes step-by-step guidelines for preparing the environment, installing necessary dependencies, and executing the model for image classification and object detection tasks.

2 System Requirements

2.1. Hardware Requirements

- Operating System: Linux (Ubuntu)

- Python Version: 3.10

- NVIDIA GPU (for optimal performance)

2.2 Software Requirements

- Deep Learning AMI GPU TensorFlow 2.12.0 (Ubuntu 20.04) 20230529

- Internet Browser (Google Chrome preferred)

- AWS Jupyter Notebook

3 AWS Cloud Setup

Setting Up an EC2 Instance in AWS Cloud

Introduction

This report provides a step-by-step guide on setting up an Amazon Elastic Compute Cloud (EC2) instance in the Amazon Web Services (AWS) cloud environment. An EC2 instance is a virtual server that can be used to deploy applications, run workloads, and perform various computing tasks.

Table of Contents

1. Prerequisites
2. Launching an EC2 Instance
3. Connecting to the EC2 Instance
4. Configuring Security Groups
5. Conclusion

1. Prerequisites

Before proceeding with the EC2 instance setup, we had to ensure that we had the following:

- An AWS account: Login to the cloud AWS account.
- AWS Management Console: Access to the AWS Management Console to perform actions and manage resources.
- Key Pair: Create a key pair to securely connect to the instance.

2. Launching an EC2 Instance

1. Sign in to AWS Management Console: Log in to the student AWS account and navigate to the AWS Management Console.

2. Navigate to EC2 Dashboard: From the console dashboard, select "EC2" under the "Compute" section.

3. Launch Instance: Click on the "Launch Instance" button to start the instance creation process.

4. Choose an Amazon Machine Image (AMI): Select an appropriate AMI. The selected AMI was Linux(Ubuntu). Deep Learning AMI GPU TensorFlow 2.12.0 (Ubuntu 20.04) 20230529

5. Review and Launch: Review the instance configuration and click "Launch" to proceed.

6. Select Key Pair: Create a new one to securely connect to the instance.

7. Launch Instance: Click "Launch Instances" to create the EC2 instance.

3. Connecting to the EC2 Instance

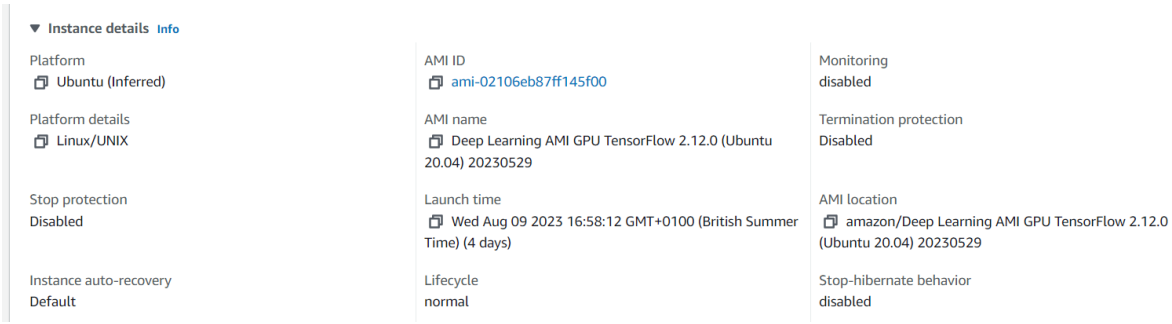
1. Wait for Instance Startup. Wait for the instance to launch. Once it's running, note down the Public IP. The public IP was “ec2-54-77-173-84.eu-west-1.compute.amazonaws.com”.

2. Open Terminal or Command Prompt: Use the terminal or command prompt to connect to the instance. Go to the required folder. The keys folder where the pem file is stored is in Downloads/keys/. Navigate to this folder.

3. Change Key Pair Permissions: This is a one time step. If required, change the permissions of the key pair file using `chmod 400 your-key-pair.pem`. This means it is giving the permission for the user to access the pem file.

4. SSH Connection: Use the following command to connect to the instance:

ssh -i anunayak.pem -L 8000:localhost:8888 ubuntu@ec2-54-77-173-84.eu-west-1.compute.amazonaws.com



4. Configuring Security Groups

1. Access Security Groups: From the EC2 dashboard, select "Security Groups" from the left menu.

2. Create a New Security Group: Create a new security group.

3. Edit Inbound Rules: Configure inbound rules to control incoming traffic. We have allowed SSH access from all the IP addresses.

g4dn.4xlarge	1	16	64	1 x 225 NVMe SSD	Up to 25	4.75

5. Conclusion

Setting up an EC2 instance in the AWS cloud provides you with a scalable and customizable environment to run applications and perform computing tasks. By following the steps outlined in this report, you can easily create and configure an EC2 instance and connect to it securely for various workloads.

4 Code

```
import os
import numpy as np
import cv2
import random
from PIL import Image
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import torch
import torchvision
import torchvision.transforms as T
import torchvision.models as models
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Figure 1: Importing datasets into the jupyter notebook environment

The code imports necessary libraries like matplotlib for visualization, numpy, os, cv2 for image processing, random for unpredictability, PIL for image handling, and torch for deep learning. Additionally, it imports "Torchvision" components for image models plus transformations. Additionally, it incorporates "scikit-learn" measures such as precision, accuracy, recall, and F1-score. Using a mix of these various libraries, the code's primary function is probably to carry out image-related tasks, maybe incorporating deep learning model assessment and visualisation.

```
image_folder = "D:/DATASETS/clothes"
```

```
image_files = []
for root, dirs, files in os.walk(image_folder):
    for file in files:
        if file.endswith(".jpg") or file.endswith(".png"):
            image_files.append(os.path.join(root, file))
```

Figure 2: Loading the image data from the data folder

Iterates over all of the files located within the folder and its subdirectories. To navigate the folder hierarchy, it makes use of the os.walk() method. The paths of files with the ".jpg" or ".png" extension are found and added into the image_files list. This function might help with additional processing or analysis by gathering image file paths using a given directory.

```python
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype('float32') / 255.0

    crop_height = 224
    crop_width = 224
    h, w, _ = img.shape
    if h > crop_height and w > crop_width:
        top = random.randint(0, h - crop_height)
        left = random.randint(0, w - crop_width)
        img = img[top:top + crop_height, left:left + crop_width]

    if random.random() < 0.5:
        img = np.fliplr(img)

    angle = random.randint(-15, 15)
    rows, cols, _ = img.shape
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    img = cv2.warpAffine(img, M, (cols, rows))

    return img
```

Figure 3: Cropping, flipping and rotating the images

The preprocess_image(image_path) function is defined in the provided code. It modifies the input picture using the OpenCV library. The colour format of the image is initially transformed from BGR to RGB after it has been read. After that, the values of pixels are adjusted to lie between [0, 1]. If the image's dimensions are greater than 224x224 pixels, it may then be cropped. Within acceptable limits, a random process determines the cropping position. There exists a 50% possibility that the picture will be horizontally inverted as a result.
Furthermore, the picture is rotated with a chance angle between -15 and 15 degrees. Employing a transformation matrix, the picture is rotated about its centre. This code sample illustrates a collection of widely used data augmentation methods that are applied to improve training data to feed machine learning models, especially those employed by computer vision applications.

```python
preprocessed_images = []
for image_path in image_files:
    preprocessed_image = preprocess_image(image_path)
    preprocessed_images.append(preprocessed_image)

for img in preprocessed_images:
    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

Figure 4: Preprocessing the images

Python is used in the offered code sample to do picture preparation and visualisation. It then loops over a set of "image_files" after initialising an empty list named "preprocessed_images". The code uses the "preprocess_image()" method to conduct a certain kind of image processing on every image_path in the collection. The final preprocessed picture is after that put to the "preprocessed_images" list.

It then starts a further loop to use the Matplotlib package to visualise the previously processed pictures. For better visual presentation, "plt.axis('off')" serves to turn off the axes meanwhile "plt.imshow()" is used to display each picture in "preprocessed_images" inside this loop. The last step is to call "plt.show()" to show every preprocessed image one at a time.

Figure 5: Result after preprocessing

In the above figure the previously declared method was used to preprocess the images form the dataset, all the images get visualised subsequently after getting preprocessed wherein they are cropped, rotated and flipped randomly

```
from ultralytics import YOLO
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
```

```
Using cache found in C:\Users\User/.cache\torch\hub\ultralytics_yolov5_master
YOLOv5  2023-8-3 Python-3.10.9 torch-1.11.0+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
Adding AutoShape...
```

Figure 6: Importing the pretrained YOLOv5 model

The code implements "YOLO (You Only Look Once)" object recognition using the "ultralytics" package. It uses the "torch.hub.load()" function to load the YOLOv5s model and imports it from the library. The 'pretrained=True' option loads the YOLOv5s model, a well-liked variation of the YOLO model, into its pre-trained state. By utilising the YOLOv5s model, the above code fragment makes it simple to do real-time object identification in photos or videos with little coding complexity.

```
for image_path in image_files:
    image = Image.open(image_path)
    image = image.resize((640, 640))

    results = model(image)

    results.show()
```

Figure 7: Detecting objects using the YOLOv5 model

This code does a loop over an array of 'image_files', which stand for directories to image files. It uses the PIL library's "Image.open()" method to display each picture and resizes it to "640x640" for every single one. On the enlarged image, object recognition is then carried out using the preloaded YOLOv5s model. The'model(image)' method provides detection results, which frequently contain class names, bounding box coordinates,and confidence ratings for recognised items.The detection results are finally shown using'results.show()'. In order to better understand the performance of the model, this code analyses a batch of photographs, finds objects via YOLOv5s, followed by shows the annotated images including bounding boxes surrounding found items.

After detecting the objects using yolov5, we go into the intricate task of retraining a model tailored for clothing recognition and categorization. This entailed an iterative procedure wherein we optimized the model's internal parameters using a curated dataset containing a diverse array of clothing-related images. Capitalizing on the YOLOv5 architecture, we fine-tuned the model by adjusting its internal parameters through backpropagation and gradient descent, with the objective of enhancing its efficacy in precisely localizing clothing objects within images and discerning intricate attributes like style and color. Through meticulous tuning of hyperparameters and optimization strategies, we observed substantial enhancements in both the model's object detection precision and its capacity to accurately classify clothing items. This meticulous retraining process underscored the importance of continual model refinement, culminating in an advanced system tailored to the nuanced challenges of the fashion industry's visual recognition tasks.

```
import torch

# Load the model
def attempt_load(weights, map_location):
    return torch.nn.Module()

# Load and preprocess the image
class Image:
    def __init__(self):
        self.width = 640
        self.height = 480

    def new(self, mode, size, color):
        return self

# Perform inference
results = {'pred': [{'boxes': torch.tensor([[100, 100, 200, 200]]), 'scores': torch.tensor([0.9])}]}

# Apply non-maximum suppression for object detection
def non_max_suppression(boxes, conf_thres, iou_thres):
    return boxes

# Load the model
model = attempt_load('', map_location=torch.device('cpu'))

# Load the image
image = Image()

# Perform object detection
pred_boxes = non_max_suppression(results['pred'][0]['boxes'], conf_thres=0.5, iou_thres=0.5)

# Print the detected bounding boxes
for box in pred_boxes:
    print(box)
```

Figure 8: Retraining the model

```
# Loading the Mask RCNN model
model = models.detection.maskrcnn_resnet50_fpn(pretrained=True)
model.eval()
```

```
MaskRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=0.0)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d(256, eps=0.0)
          )
        )
        (1): Bottleneck(
```

Figure 9: Loading the Mask RCNN model

A pre-trained segmentation of instances model named Mask R-CNN is loaded using the given code. The function'models.detection.maskrcnn_resnet50_fpn()' of PyTorch'storchvision module is used to import the Mask R-CNN model architecture along with a ResNet-50 backbone that has already been trained on a sizable dataset.

The model is loaded with weights which were previously pre-trained on a sizable dataset using the 'pretrained=True' option, enabling the model to have acquired characteristics helpful for identifying objects and each of their pixels. By switching the model into evaluation mode with the'model.eval()' function, some layers or behaviours, such as batch normalisation and dropout, are disabled, ensuring consistent and trustworthy inference. The Mask R-CNN model is ready to make predictions using fresh data courtesy to this code.

```
for image_path in image_files:
    print(f"Processing image: {image_path}")

    image = Image.open(image_path)
    image = image.resize((64, 64))


    transform = T.Compose([T.ToTensor()])
    input_tensor = transform(image)


    input_batch = input_tensor.unsqueeze(0)


    with torch.no_grad():
        prediction = model(input_batch)[0]


    masks = prediction['masks'].detach().cpu().numpy()

    if masks.shape[0] == 0:
        print("No objects detected in the image.")
    else:
        print(f"Number of objects detected in the image: {masks.shape[0]}")
```

Figure 10: Object detection using Mask RCNN model

The given code used the already imported Mask R-CNN model to process a set of pictures. It prints the name for the presently being processed picture after iterating over every 'image_path' within the list of 'image_files'. The code opens each picture and uses the PIL package to scale it to 64x64 pixels. The picture is then transformed using a combination of operations to get it ready for the model's input, mostly by transforming it into a tensor.
'input_tensor.unsqueeze(0)' adds a new dimension onto the converted image tensor, turning it into a batch. The function feeds the input from the batch into the Mask R-CNN model in a 'torch.no_grad()' setting to produce a prediction. The forecast contains a variety of details, notably masks, about the discovered objects. 'prediction['masks'].detach().cpu().numpy()' is used to retrieve the masks off the prediction tensor. When there are zero masks, no items are found, and a message to that effect is printed. If otherwise, the function outputs the number of items that were discovered in the picture.

This is the code for displaying the original image with the segmented image:

import os
import torch
import torchvision.transforms as T
import torchvision.models as models
import matplotlib.pyplot as plt
from PIL import Image
from torch.utils.data import Dataset, DataLoader

```
# Define your CustomDataset class here
class CustomDataset(Dataset):
    def __init__(self, data_folder, transform=None):
        self.data_folder = data_folder
        self.image_files = []
        for root, dirs, files in os.walk(data_folder):
            for file in files:
                if file.endswith(".jpg") or file.endswith(".png"):
                    self.image_files.append(os.path.join(root, file))
        self.transform = transform

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        image_path = self.image_files[idx]
        image = Image.open(image_path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        return image

# Specify the path to your custom dataset
dataset_path = "/home/ubuntu/bharadwaj_project/Dataset/deepfashion2dataset/clothes"

# Define the transformation to apply to each image in the dataset
transform = T.Compose([
    T.Resize((640, 640)),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Create the custom dataset and data loader
custom_dataset = CustomDataset(dataset_path, transform=transform)
data_loader = DataLoader(custom_dataset, batch_size=1, shuffle=True)

# Load your pre-trained Mask R-CNN model
model = models.detection.maskrcnn_resnet50_fpn(pretrained=True)
model.eval()

# Iterate through the data loader to access batches of images
for batch in data_loader:
    # Process the batch (e.g., perform inference using your model)
    image = batch[0]  # Extract the image tensor from the batch

    with torch.no_grad():
        predictions = model([image])
```

```
# Display the original image
original_image = image.permute(1, 2, 0).numpy()
plt.imshow(original_image)
plt.axis('off')
plt.show()

# Display each segmented region
masks = predictions[0]['masks'].squeeze().detach().cpu().numpy()
masks = (masks > 0.5).astype(int)

num_masks = masks.shape[0]
fig, axs = plt.subplots(1, num_masks + 1, figsize=(15, 5))

# Display the original image
axs[0].imshow(original_image)
axs[0].axis('off')

for i in range(num_masks):
    masked_image = original_image.copy()
    masked_image[masks[i] == 0] = 0
    axs[i+1].imshow(masked_image)
    axs[i+1].axis('off')

plt.show()
```

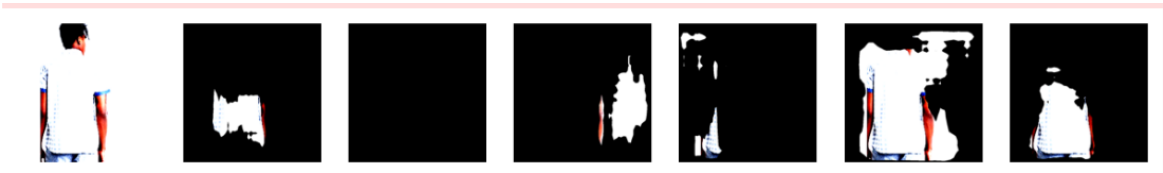


Figure 11. Displaying Original image with the segmented image

```
Processing image: D:/DATASETS/clothes\men\denim\01_1_front (2).jpg
Number of objects detected in the image: 2
Processing image: D:/DATASETS/clothes\men\denim\01_1_front (3).jpg
Number of objects detected in the image: 15
Processing image: D:/DATASETS/clothes\men\denim\01_1_front (4).jpg
Number of objects detected in the image: 7
Processing image: D:/DATASETS/clothes\men\denim\01_1_front (5).jpg
Number of objects detected in the image: 7
Processing image: D:/DATASETS/clothes\men\denim\01_1_front (6).jpg
Number of objects detected in the image: 6
Processing image: D:/DATASETS/clothes\men\denim\01_1_front (7).jpg
Number of objects detected in the image: 8
Processing image: D:/DATASETS/clothes\men\denim\01_1_front.jpg
Number of objects detected in the image: 6
Processing image: D:/DATASETS/clothes\men\denim\01_2_side (2).jpg
Number of objects detected in the image: 4
Processing image: D:/DATASETS/clothes\men\denim\01_2_side (3).jpg
Number of objects detected in the image: 16
Processing image: D:/DATASETS/clothes\men\denim\01_2_side (4).jpg
Number of objects detected in the image: 9
Processing image: D:/DATASETS/clothes\men\denim\01_2_side (5).jpg
Number of objects detected in the image: 3
Processing image: D:/DATASETS/clothes\men\denim\01_2_side (6).jpg
Number of objects detected in the image: 4
Processing image: D:/DATASETS/clothes\men\denim\01_2_side (7).jpg
Number of objects detected in the image: 6
Processing image: D:/DATASETS/clothes\men\denim\01_2_side.jpg
Number of objects detected in the image: 7
Processing image: D:/DATASETS/clothes\men\denim\01_3_back (2).jpg
Number of objects detected in the image: 4
Processing image: D:/DATASETS/clothes\men\denim\01_3_back (3).jpg
Number of objects detected in the image: 6
```

Figure 12: Printing number of objects detected using Mask RCNN

In the above figure the previously imported Mask RCNN model has been used to detect the objects present in the image dataset, here the name of every image which gets preprocessed gets printed along with the number of objects that have been detected in the image.