# Configuration Manual

MSc Research Project
Data Analytics

Priyal Narendra Patil
Student ID: x20193394

School of Computing
National College of Ireland

Supervisor: Dr. Giovani Estrada

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | | | |
|---|---|---|---|
| **Student Name:** | Priyal Narendra Patil | | |
| **Student ID:** | X20193394 | | |
| **Programme:** | MSc in Data Analytics | **Year:** | 2021-2022 |
| **Module:** | Research Project | | |
| **Supervisor:** | Dr. Giovani Estrada | | |
| **Submission Due Date:** | | | |
| **Project Title:** | Hierarchical Classification of Insects using a Combination of Resnet and VGG Networks | | |
| **Word Count:** | 2785 words **Page Count** 16 Pages | | |

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:** Priyal Narendra Patil

**Date:** 15th August 2022

**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

Priyal Narendra Patil
X20193394

## 1 Hardware/Software Requirements

This Configuration manual encompasses the steps which must be followed to run the python notebooks. These deep learning models require certain minimum hardware requirements to be executed. Due to the nature of the scripts, a single experiment is divided into 2-3 parts where 1 -2 parts are of the training of the model and the last part is about the prediction.

## 2 System Specification

The entire project had been developed on the "Google Colab" which is a service by google where everyone who needs to run the python code can be provided with the online python notebook where they can run any python code. And it also provides a high ram as well as GPU. Basically, cloud-based Jupyter notebook.

### 2.1 Hardware Requirements

If someone wants to run on their personal computer, they should have at least the following hardware to make sure the script doesn't run out of resources. These are not the exact specification because google collab in every session and allocates the system based on available resources.

- Processor: Single Core 2.2-2.3 GHz
- RAM: 12GB
- GPU: Nvidia Tesla T4, 16GB memory

Any system equivalent to the above can run the script

### 2.2 Software Requirements

The following program should be required

- Google Collaboratory (or Jupyter Notebook for on-system execution)
- Python 3
- Microsoft Excel

## 3 Setting up the environment

This section information about the enabling of Google Collab in the Gmail account.

First using the Gmail account's Google drive, create a folder named "Project" And there upload all the "Jupyter notebooks" and the "ipynb" files are uploaded.

By Default google, the collab is not installed in google drive. To install that right click and select more> Connect more apps as follows

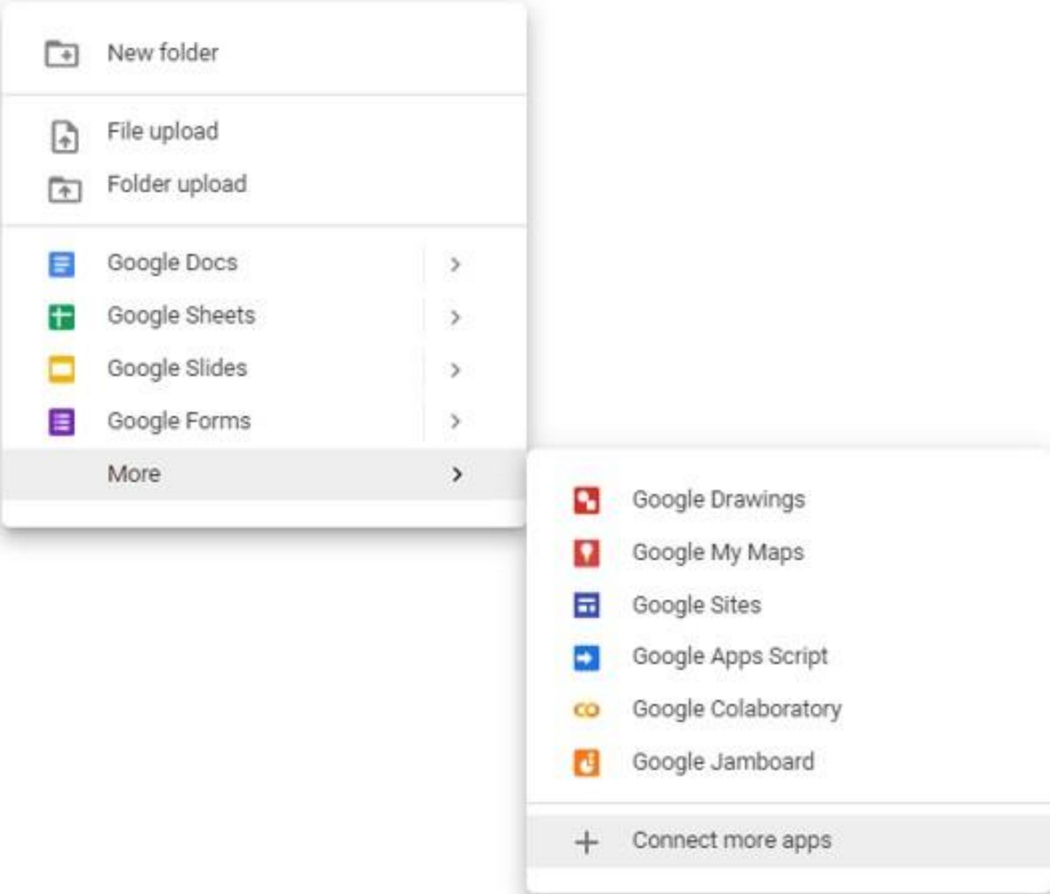


**Figure 1 Options to add more apps**

Search for Collaboratory

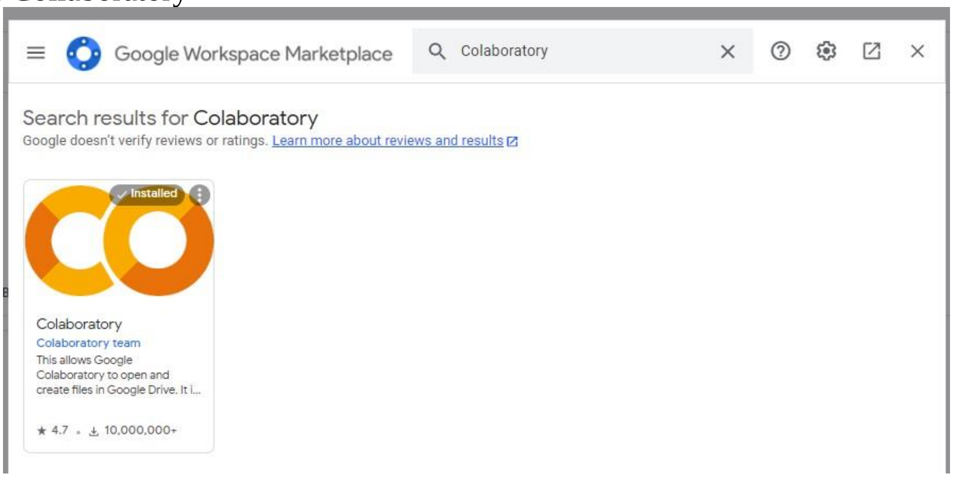


**Figure 2 Search Collaboratory**

On clicking on the Collaboratory select an option to install it, if it shows uninstall button that means the tool has been installed,

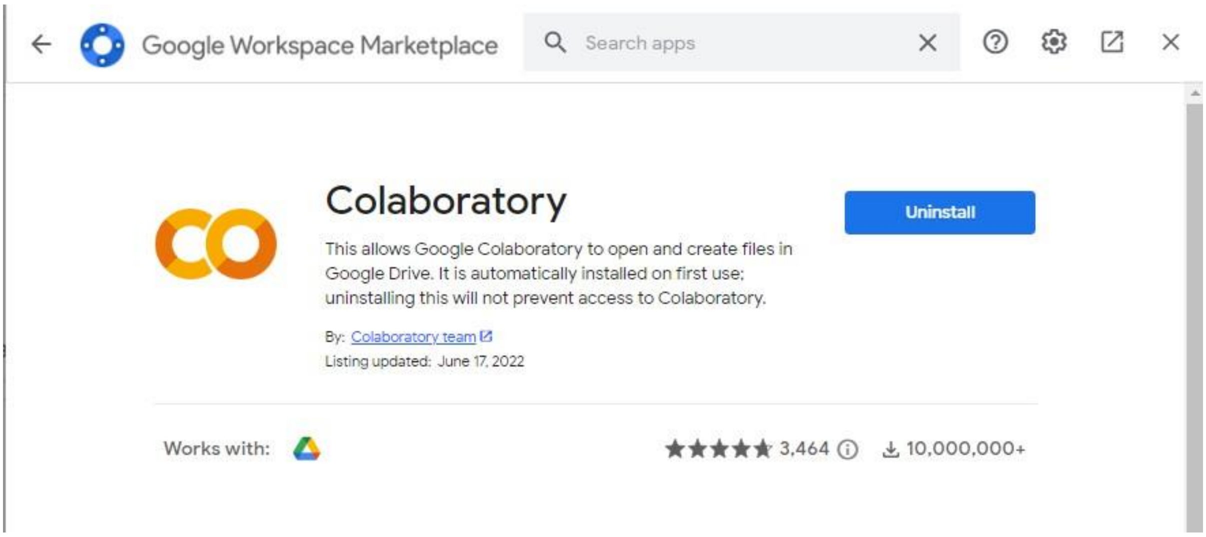


**Figure 3 Installed Collaboratory**

Now once that's done. One can open the "ipynb" notebooks by right-clicking on it and selecting the "Open with > Google Collaboratory" as shown in the following window.

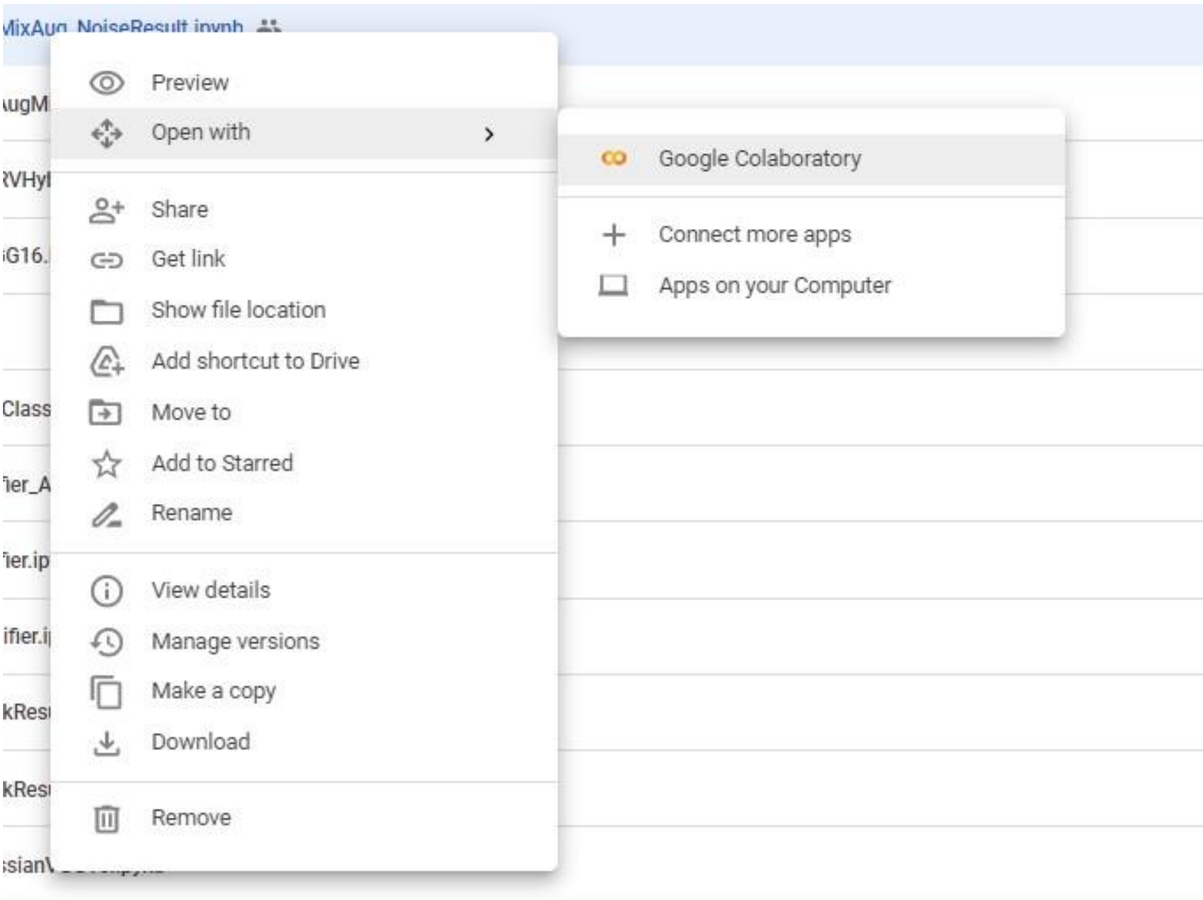


**Figure 4 Option to open Jupyter notebook files**

# 4 Data Selection

Data can be collected from the following GitHub link. The author had provided the link to google drive, from where data either can be downloaded or can directly be used if using the google Collaboratory

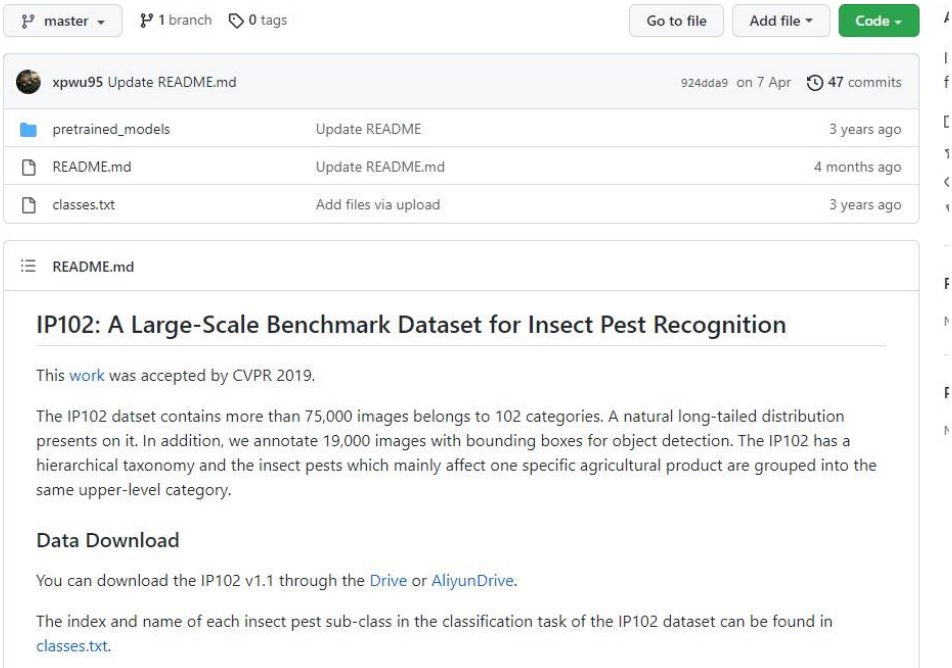


**Figure 5 IP102 Dataset Download Location Link: [https://github.com/xpwu95/IP102](https://github.com/xpwu95/IP102)**

This data can be downloaded as well as it will show up in the "Shared With Me" section in Google Drive as well,

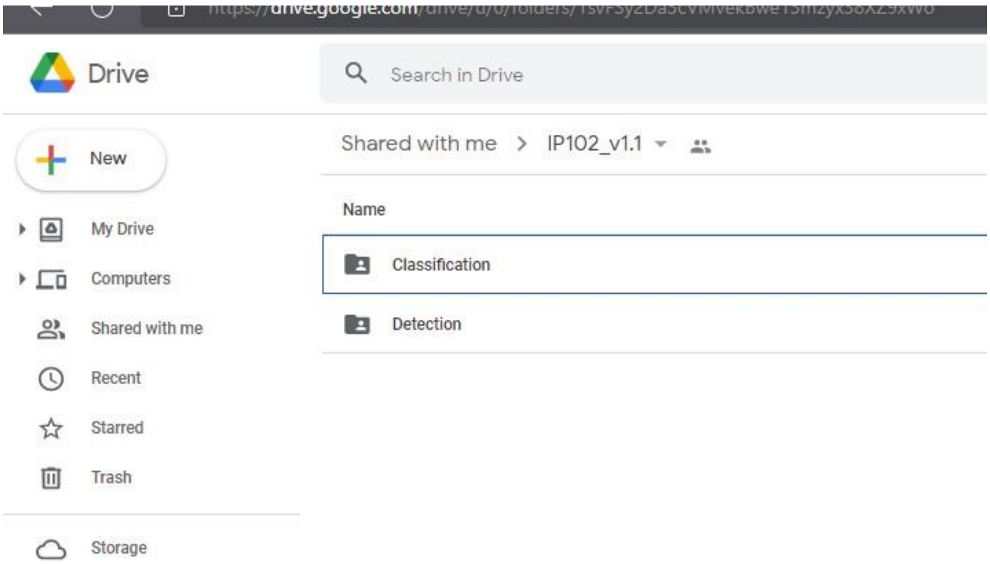


**Figure 6 IP102 Data Files**

# 5 Implementation

Following libraries are required to run the script, however, most of it will be already installed in the google collaboratory session, and 1-2 will not will but the installation commands are integrated into the scripts already. But others should be installed if used into the local system.

- Numpy
- SkLearn
- PyTorch
- Matplotlib
- Seaborn
- Shutil
- Barbar

```
import os
import pandas as pd
import shutil
```

```
import torch
import os
import numpy as np
import torch
from torch import nn
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets import ImageFolder
from torchvision import transforms
import torch.optim as optim
import argparse
from distutils.util import strtobool

from barbar import Bar
import copy
import time
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

**Figure 7 Libraries**

## 5.1 Importing Data

File “ip102_v1.1.tar” and “classes.txt” should be placed into the “Project” Folder which was created earlier in google drive. One can also make the shortcut of the same file from the shared with me section.

And Executing the following command copies, the content of that “tar” file to the disk space provided by the google collab session.

```
!tar -xf /content/drive/MyDrive/Project/ip102_v1.1.tar
```

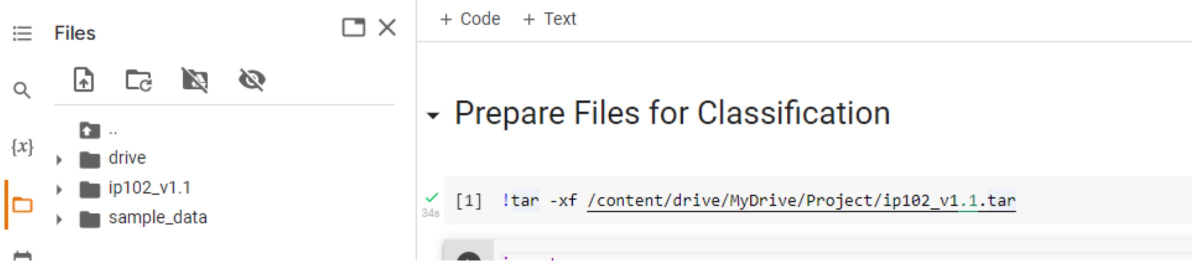


**Figure 8 Extracting the IP102 data and Keeping in the disk space**

## 5.2 Reading Data

Once the data is imported into the Google Collaboratory Session, can be arranged in folders such that it can be accessed via the Image Loader of the PyTorch.

First, the train, test and val folders are created.

```
try:
  os.mkdir("train")
  os.mkdir("test")
  os.mkdir("val")
except Exception as e:
  pass
```

**Listing 1 To create Train, Test and Validation folders**

Following has the functions to move and copy files in the respective folders which is necessary for the image data loaded.

```
def read_actual_labels(file_name):
    # list to store the names of the image files
    file_names = []
    # List tostore the actual lables of each image
    actual_labels = []
    # Read train, test and val file to get the list of files names for
each categories
    train_file = open("ip102_v1.1/"+file_name)
    for l in train_file:
        file_names.append(l.split(" ")[0])
        actual_labels.append(int(l.split(" ")[-1][:-1]))
    train_file.close()
    return file_names,actual_labels # Return the pair of the list with
same size but content different
# Creating list to convert the label to actual name of categories
super_calss = ["Rice", "Corn", "Wheat", "Beet", "Alfalfa", "Vitis", "Ci
trus", "Mango"]
super_class_count = [ 14, 13, 9, 8, 13, 16, 19, 10]

# Create the subclass in the train, test and val folders to store image
s
```

```
for c in super_calss:
    try:
        os.mkdir("train/"+c)
    except Exception as e:
        continue
    try:
        os.mkdir("test/"+c)
    except Exception as e:
        continue
    try:
        os.mkdir("val/"+c)
    except Exception as e:
        continue
# This will continue previous process
class_cetegories = []
prev = 0
for i in range(0,len(super_class_count)):
    cl = list(range(prev,prev+super_class_count[i]))
    class_cetegories.append(cl)
    prev = prev + super_class_count[i]

    # Define function to map the label number ot the image name
def find_category(inp_cat):
    for i in range(len(class_cetegories)):
        if inp_cat in class_cetegories[i]:
            return super_calss[i]
# Function to move files to the respective folder
def movefiles(f_type,f_name,f_label):
    root_image_folder = "ip102_v1.1/images/"
    # Following for loop to move images to the trianing folder
    for i in range(len(f_name)):
        name = f_name[i]
        label = f_label[i]
        subfolder = find_category(label)
        shutil.move(root_image_folder+name,f_type+"/"+subfolder+"/"+nam
e)
# Create Folders for the sub classification
error = ""
try:
    os.mkdir("new_train")
    os.mkdir("new_train/train")
    os.mkdir("new_train/test")
    os.mkdir("new_train/val")
    for c in super_calss:
        os.mkdir("new_train/train/"+c)
        os.mkdir("new_train/test/"+c)
        os.mkdir("new_train/val/"+c)
except Exception as e:
```

```
    error = e
# File to Read the Actual Class of the image
classes_path = "/content/drive/MyDrive/Project/classes.txt"
dataclass = {}
class_file = open(classes_path)
for l in class_file:
    class_idx = int(l.split()[0])-1
    class_name = " ".join(l.split()[1:])
    dataclass[class_idx] = class_name
class_file.close()
# Following is the function to create the folder,
def copy_images_to_sub_categories(r_folder,dict_file):
    r_folder = r_folder + "/"
    super_class_names = os.listdir(r_folder)

    for sc in super_class_names:
        for img_n in os.listdir(r_folder+sc):
            image_label = dict_file[img_n]
            sub_name = dataclass[image_label]
            if not os.path.exists("new_train/"+r_folder+"/"+sc+"/"+sub_
name):
                os.mkdir("new_train/"+r_folder+"/"+sc+"/"+sub_name)
            shutil.copy(r_folder+sc+"/"+img_n,"new_train/"+r_folder+"/"
+sc+"/"+sub_name+"/"+img_n)
```

**Listing 2 Functions and Code to create the folders**

```
f_train,l_train = read_actual_labels("train.txt")
f_val,l_val = read_actual_labels("val.txt")
train_f = {f:l for f,l in zip(f_train,l_train)}
val_f = {f:l for f,l in zip(f_val,l_val)}
# Move files to the super categories
movefiles("train",f_train,l_train)
movefiles("val",f_val,l_val)
# Move files to the sub categories
copy_images_to_sub_categories("train",train_f)
copy_images_to_sub_categories("val",val_f)
```

**Listing 3 Script to move files for the training**

```
f_test,l_test = read_actual_labels("test.txt")
test_f = {f:l for f,l in zip(f_test,l_test)}
# Move files to the super categories
movefiles("test",f_test,l_test)
# Move files to the sub categories
copy_images_to_sub_categories("test",test_f)
```

**Listing 4 Script to move files for the prediction or test**

These last 2 Scripts are executed depending on it if it is training or prediction.

## 5.3 Data Processing

```
data_transforms = {
            'train': transforms.Compose([
               transforms.Resize(256),
               transforms.AugMix(),
                transforms.RandomCrop(input_size),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.2
24, 0.225])
            ]),
            'val': transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(input_size),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.2
24, 0.225])
            ]),
        }
```

**Listing 5 This is the data preprocessing enclosed into the data transform for training**

```
data_transforms = {
            'test': transforms.Compose([
                transforms.Resize(size=(input_size,input_size)),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.2
24, 0.225])
            ]),
        }
```

**Listing 6 This is the data preprocessing enclosed into the data transform for testing/ prediction**

## 5.4 Data Splitting

Since the dataset is a benchmark dataset, it comes with pre slited and, as in the previous part. Respective data copied in respective folder based on the split provided by the owner of the dataset.

## 5.5 Resnet50

Using the pytoch library a ready built structure is used which can be imported using following script. And is modified based on the number of target class.

```
# Load Pretrained ResNet50 Model
res_net_model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pr
etrained=True)


dropout = 0.5
num_ftrs = res_net_model.fc.in_features
```

**Listing 7 Script to create ResNet50 model for the super class classification**

```
models = {}
for m in super_calss:
  models[m] = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pret
rained=True)
  dropout = 0.5
  num_ftrs = models[m].fc.in_features
  # Add new layer to make it to classify for 8 super class
  models[m].fc = nn.Sequential(
                nn.Dropout(dropout),
                nn.Linear(num_ftrs, len(os.listdir("/content/new_train/
train/"+m)))
```

**Listing 8 Script to create ResNet50 model for the sub class classification**

## 5.6 VGG16

Here again similar to the ResNet50 method, for the VGG16 porch library is used to get the pre bult structure along with the pretrained weights.

```
def get_model(super_category_data,type_m):
  model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16', pretrained=
True)
  tc = len(os.listdir("new_train/"+type_m+"/"+super_category_data))
  model.classifier = nn.Sequential(model.classifier,nn.Linear(1000, tc)
)
  torch.cuda.empty_cache()
  model = model.to("cuda")
  return model
```

**Listing 9 Script to create the VGG16 model for the sub class**

```
# Load Pretrained ResNet50 Model
model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16', pretrained=Tr
ue)

# Add new layer to make it to classify for 8 super class
model.classifier = nn.Sequential(model.classifier,
    nn.Linear(1000, 8))
```

**Listing 10 Script to create the VGG16 model for the super class**

## 5.7 Training

Initially there are individual script for training of the models for the sub class but later a single code was used which had following logic in it.

```
def get_model(super_category_data,type_m):
  model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16', pretrained=
True)
  tc = len(os.listdir("new_train/"+type_m+"/"+super_category_data))
  model.classifier = nn.Sequential(model.classifier,nn.Linear(1000, tc)
)
  torch.cuda.empty_cache()
  model = model.to("cuda")
  return model

def add_noise(inputs):
  noise = torch.randn_like(inputs)*0.2
  return inputs + noise

# Following is to train the model
def train_model(in_model,train_set,valid_set,n_epochs):
  weight_decay =  0.00001
  val_acc_history = []
  train_acc_history = []
  val_loss_history = []
  train_loss_history = []
  dataloaders = {'train': train_set, 'val': valid_set}
  criterion = nn.CrossEntropyLoss()
  params_to_update = in_model.parameters()
  optimizer_ft = optim.Adam(params_to_update, lr= 0.0001, betas= (0.9,
0.999),
                                  eps= 1e-
08, weight_decay= weight_decay)
  scheduler = optim.lr_scheduler.ExponentialLR(optimizer_ft, gamma= 0.9
6)
  optimizer = optimizer_ft
  is_save_checkpoint = False
  is_inception = False
  ckpepoch = 0
  since = time.time()
  best_acc = 0.0
  num_epochs = n_epochs
  device = "cuda"
  for epoch in range(ckpepoch, num_epochs):

      print('Epoch {}/{}'.format(epoch, num_epochs - 1))
      print('-' * 10)

      for phase in ['train', 'val']:
          if phase == 'train':
              in_model.train()
          else:
              in_model.eval()
```

**Listing 9 Functions for performing the training of the both model**

```
        running_loss = 0.0
        running_corrects = 0

        for data in Bar(dataloaders[phase]):
            if len(data) > 2:
                inputs, _ , labels = data
                inputs = add_noise(inputs)
                inputs = inputs.to(device)
            else:
                inputs, labels = data
                inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                if is_inception and phase == 'train':
                    outputs, aux_outputs = in_model(inputs)
                    loss1 = criterion(outputs, labels)
                    loss2 = criterion(aux_outputs, labels)
                    loss = loss1 + 0.4*loss2
                else:
                    outputs = in_model(inputs)
                    loss = criterion(outputs, labels)

                _, preds = torch.max(outputs, 1)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            inputs.to("cpu")
            torch.cuda.empty_cache()
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(dataloaders[phase].dataset)
        epoch_acc = running_corrects.double() / len(dataloaders[phase
].dataset)

        print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss,
 epoch_acc))

        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(in_model.state_dict())
            # if is_save_checkpoint:
```

```
  valid_set = ImageFolder(root= "new_train/val/"+s_cname, transform= da
ta_transforms['val'])
  train_set = DataLoader(train_set, batch_size= batch_size, shuffle= Tr
ue,
                              num_workers= 8, pin_memory=True)
  valid_set = DataLoader(valid_set, batch_size= batch_size, shuffle= Fa
lse,
                                num_workers= 8, pin_memory= True)
  train_model(model_d,train_set,valid_set,10)
  torch.cuda.empty_cache()
  torch.save(model_d.state_dict(), "/content/drive/MyDrive/Project/"+s_
cname+"_vgg16")
```

**Listing 10 Training loop for the Sub class models**

## 5.8 Prediction

Prediction are carried out by using the test folder, and using the stored models, Follwowing scripts shows the loading models from the stored weights.

```
models = {}
for m in super_calss:
  models[m] = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pret
rained=True)
  dropout = 0.5
  num_ftrs = models[m].fc.in_features
  # Add new layer to make it to classify for 8 super class
  models[m].fc = nn.Sequential(
              nn.Dropout(dropout),
              nn.Linear(num_ftrs, len(os.listdir("/content/new_train/
test/"+m))))
  superModelDict = torch.load("/content/drive/MyDrive/Project/submodels
/"+m)
  models[m].load_state_dict(superModelDict)
  models[m].eval()
```

**Listing 13 Resnet50 loading wights stored on the drive**

```
actual_super_class = []
predicted_super_class = []
actual_subclass = []
predicted_subclass = []
images =[]
for sc in reverse_classmap:
  test_images = image_data_gen[reverse_classmap[sc]]
  y_pred_l = []
  y_act_l = []
  i = 0
  for test_data in iter(test_images):
    input_img,target_img = test_data
```

```
    input_img = add_noise(input_img)

    ac =list(target_img.numpy())

    y_act_l.extend(ac)

    # input_img.to("cuda")
    pred = model(input_img.cuda())
    torch.cuda.empty_cache()
    input_img.cpu()

    predicted_classss= list(np.argmax(pred.cpu().detach().numpy(),axis=
1))
    y_pred_l.extend(predicted_classss)


    # process to generate the sub classification
    unique_predicted =np.unique(predicted_classss)
    for u in unique_predicted:
      image_index = np.array(predicted_classss) == u  # Find the index
of the given class in the predicted
      actual_class_ofthose = len(list(np.array(ac)[image_index]))*[sc]
 # add the actual categories of the super class out of 8 class
      predicted_class_ofthose = list(np.array(predicted_classss)[image_
index]) # This is the predicted class of the super class
      c_images = input_img[image_index]  # Images of the u super class
      act_labels = list(target_img[image_index].numpy()) # it is the ac
tual sub class of images
      actual_super_class.extend(actual_class_ofthose)  # adding the act
ual super class labels
      predicted_super_class.extend(predicted_class_ofthose) # Adding th
e predicted super class labels
      actual_subclass.extend(act_labels)  # Adding the actual sub class
      predy = models[reverse classmap[u]](c images.cuda())  # Predictin
g the sub class of the images by selecting the model and storing image
on the GPU to speedup the process
      c_images.cpu() # Move images to the CPU() to release the GPU emeo
ry
      predicted_sub_classss= list(np.argmax(predy.cpu().detach().numpy(
),axis=1)) # get  the predicted subclass
      predicted_subclass.extend(predicted_sub_classss) # Add the predic
ted sub class to larger array
      images.append(c_images)
      torch.cuda.empty_cache()
```

**Listing 11 Loop for performing the prediction of the test data.**

Following script are used to show the confusion matrix as well as the classification report for the super class as well as the sub class.

```
total_count = {}
for c in os.listdir("/content/new_train/test"):
  total_count[c] = len(os.listdir("/content/new_train/test/"+c))

as_c = [reverse_classmap[c] for c in actual_super_class]
ps_c = [reverse_classmap[c] for c in predicted_super_class]
len(as_c),len(ps_c)

from sklearn.metrics import classification_report
print(classification_report(as_c,ps_c))

reverse_category = {}
for c in image_data_cetegory:
  tmp = {}
  for cc in image_data_cetegory[c]:
    tmp[image_data_cetegory[c][cc]] = cc
  reverse_category[c] = tmp
actual_y = []
predicted_y = []

for i in range(len(as_c)):
  actual_sname = as_c[i]
  predicted_sname = ps_c[i]
  actual_sub_class = actual_subclass[i]
  predicted_sub_class = predicted_subclass[i]
  actual_y.append(reverse_category[actual_sname][actual_sub_class])
  predicted_y.append(reverse_category[predicted_sname][predicted_sub_cl
ass])


from sklearn.metrics import classification_report
print(classification_report(actual_y,predicted_y))
torch.cuda.empty_cache()
```

**Listing 12 Script for creating the classification report**

```
t = classification_report(actual_y,predicted_y,output_dict=True)
import pandas as pd
pd.DataFrame(t).T.to_csv("/content/drive/MyDrive/Project/classification
Result_50.csv")
```

**Listing 13 Script to store the Classification report to drive**

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
cm = confusion_matrix(as_c,ps_c)
plt.figure(figsize=(5,5))
sns.heatmap(pd.DataFrame(cm,columns=np.unique(as_c),index=np.unique(as_
```

**Listing 14 Script for the confusion matrix**

Now following scripts were used to get the image visualization,

```
MEAN = torch.tensor([0.485, 0.456, 0.406])
STD = torch.tensor([0.229, 0.224, 0.225])

# Perform inverse Normalization for the visualization
new_image_list_for_vis = []
for set_img in images:
  new_image_list_for_vis.extend([imag * STD[:, None, None] + MEAN[:, No
ne, None] for imag in set_img])
new_image_list_for_vis = [im.T for im in new_image_list_for_vis]

sub_class_correct_idx = np.array(actual_y) == np.array(predicted_y)
sub_class_incorrect_idx = np.array(actual_y) != np.array(predicted_y)
super_class_correct_idx = np.array(as_c) == np.array(ps_c)
super_inclass_correct_idx = np.array(as_c) != np.array(ps_c)

correct_sublcass_idx = np.array(range(len(actual_y)))[sub_class_correct
_idx]
incorrect_sublcass_idx = np.array(range(len(actual_y)))[sub_class_incor
rect_idx]

correct_suplcass_idx = np.array(range(len(actual_y)))[super_class_corre
ct_idx]
incorrect_suplcass_idx = np.array(range(len(actual_y)))[super_inclass_c
orrect_idx]
a,b,c,d = correct_sublcass_idx[0],incorrect_sublcass_idx[0],correct_sup
lcass_idx[1],incorrect_suplcass_idx[1]

figure, axis = plt.subplots(2, 2)
figure.set_figwidth(15)
figure.set_figheight(15)
plt.figure(figsize=(12,12))
axis[0,0].imshow(new_image_list_for_vis[a])
axis[0,0].set_title("Actual Super Class: {}, Predicted Super Class: {}\
nActual Sub Class: {}, Predicted Sub Class: {}".format(as_c[a],ps_c[a],
actual_y[a],predicted_y[a]))

axis[0,1].imshow(new_image_list_for_vis[b])
axis[0,1].set_title("Actual Super Class: {}, Predicted Super Class: {}\
nActual Sub Class: {}, Predicted Sub Class: {}".format(as_c[b],ps_c[b],
actual_y[b],predicted_y[b]))

axis[1,0].imshow(new_image_list_for_vis[c])
axis[1,0].set_title("Actual Super Class: {}, Predicted Super Class: {}\
nActual Sub Class: {}, Predicted Sub Class: {}".format(as_c[c],ps_c[c],
actual_y[c],predicted_y[c]))
```

**Listing 15 Script to depict the result**

# 6 Other Software's

Apart from those mentioned, A local jupyter notebook from the anaconda was used to modify the notebooks sometime. And MS Word was used for making the documentation.