Calorie Predictor

How to teach a model to recognize objects ?

Big Picture

Assume that you want to teach a kid to recognize cats and dogs and you can't take him outside because of the quarantine, How would you go about it?

You would probably **collect** some pictures/**data** of cats and dogs, then **show each picture to him** and **tell him whether it is a cat or a dog**, after looking at a few pictures he would form an idea of what a cat or dog looks like. Then when you **show him a new image** of a cat/dog and he can recognize it, it would mean that he has **learnt/trained** well.

The process of "Teaching" a machine/model to recognize objects is similar.

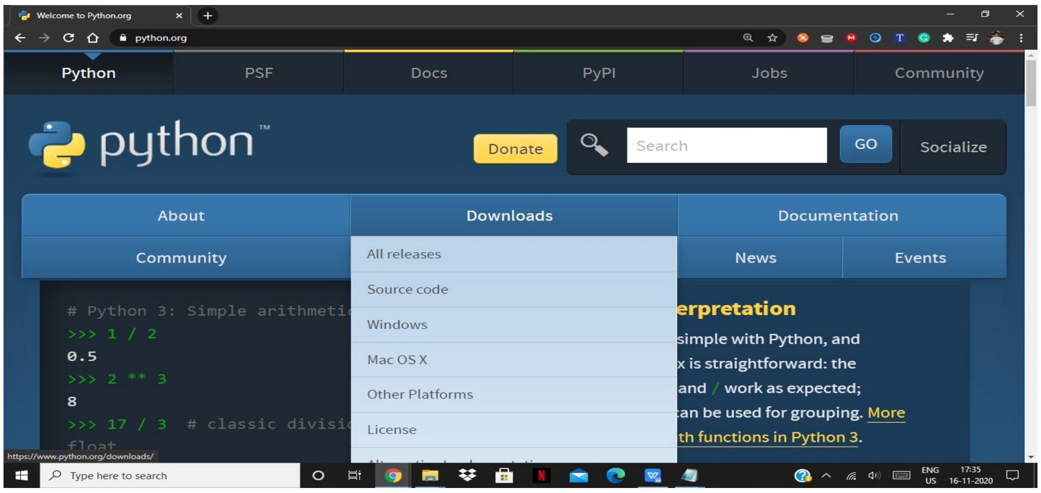
- First, You collect some images containing the objects/**classes** you want it to recognize like cats and dogs (AKA **Data Collection)**
- Next, You **label/annotate** each image, which is like writing "cat" and "dog" on top of the pictures based on what it contains so that the model knows what it's looking at. (AKA **Data Annotation**)
- Now, You show each image to the model along with its label, i.e. you **train** the model. (**Model Training**)
- Finally, to check/**evaluate** how well the machine has **learnt,** you show it a bunch of new images and see if it answers accurately. (AKA **Model Evaluation**)

Setup

In order to get started we need to install Python and PyCharm.

Python

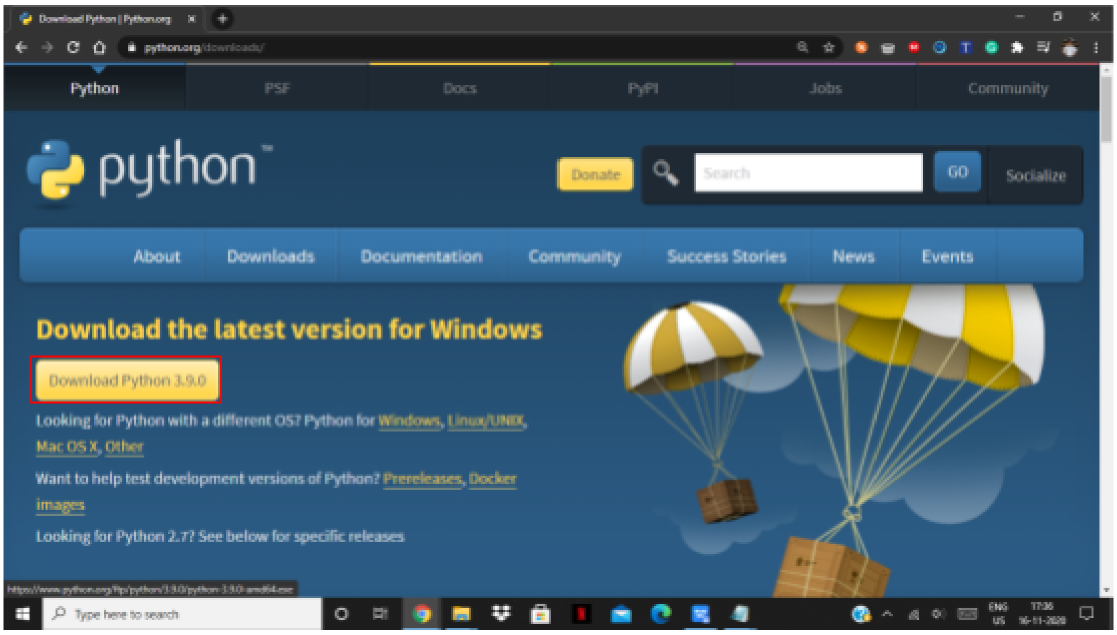
- Go to python.org on your browser and click on Downloads.



```
# Python 3: Simple arithmeti
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3  # classic divisi
float
```

erpretation
simple with Python, and
x is straightforward: the
and / work as expected;
an be used for grouping. More
th functions in Python 3.

- Click on the Download option.



Download the latest version for Windows

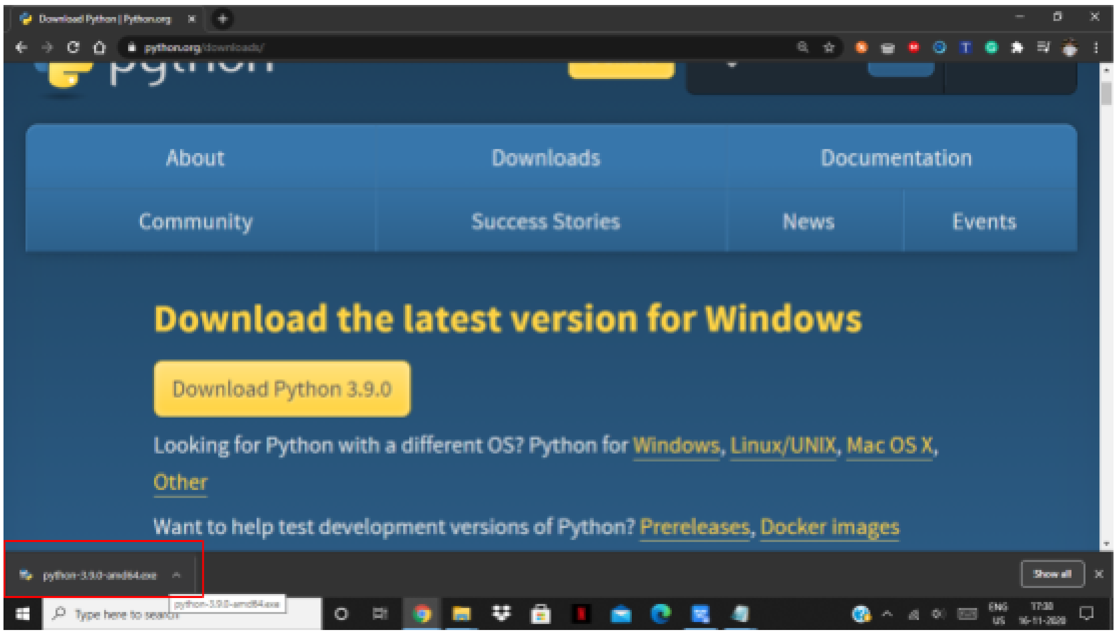
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

https://www.python.org/ftp/python/3.9.0/python-3.9.0-amd64.exe

- Double click on the downloaded python installer file.



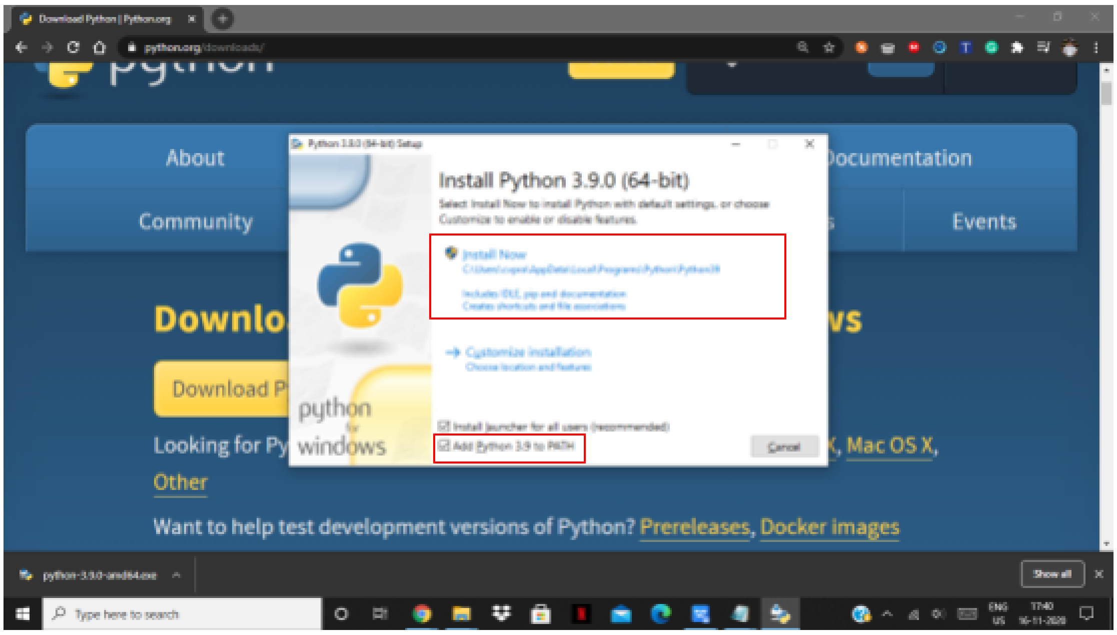
Download the latest version for Windows

Download Python 3.9.0

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

- On windows, Tick the Add Python to Path box and click install now.


Install Python 3.9.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\[illegible]\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.9 to PATH

PyCharm

Just like we use MS Word to write documents, we need to install a tool to write and run code. In our case, we are going to use a very popular tool called PyCharm.

- Follow the instructions in this video to install and start pycharm on your computer.

Steps

The instructions will take you through these four steps:

Data Collection → Data Annotation → Model Training → Model Evaluation

Data Collection

The simplest way to collect images is by downloading them from Google images!

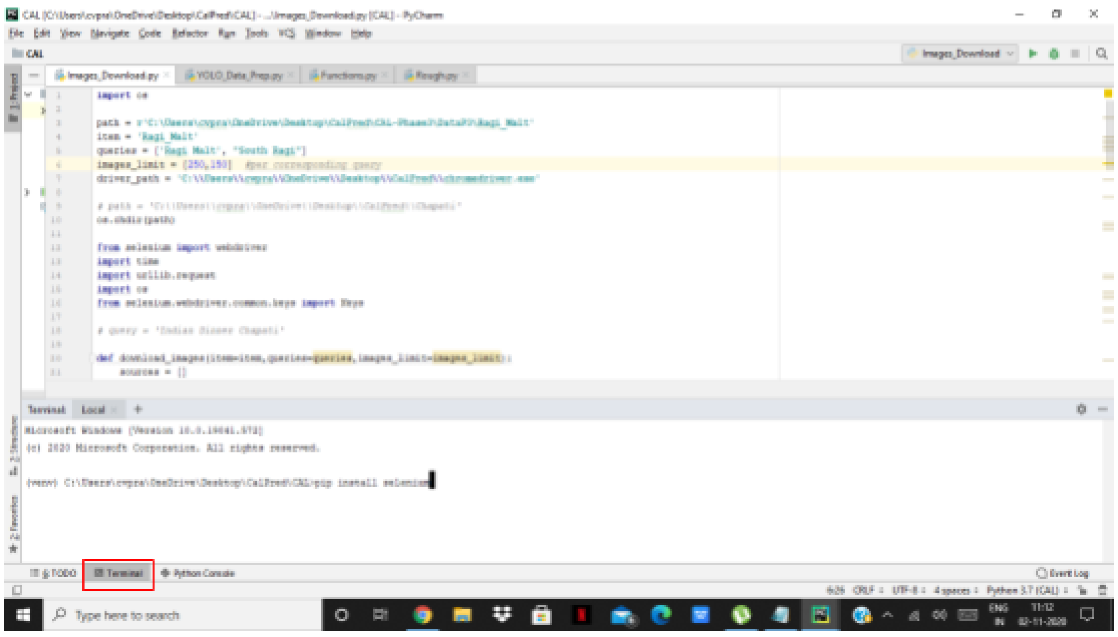
As doing this manually would be too boring and time consuming,
Use the below code/program to automatically download images from Google and store it in your computer. (AKA **web scraping**)

Step 1 : Install Selenium

Selenium is a package that lets your code perform operations like opening your browser, searching for something, downloading images etc. That is, Simulate User actions.

- Go to the terminal section at the bottom left corner in PyCharm and type the following command.

```
pip install selenium
```



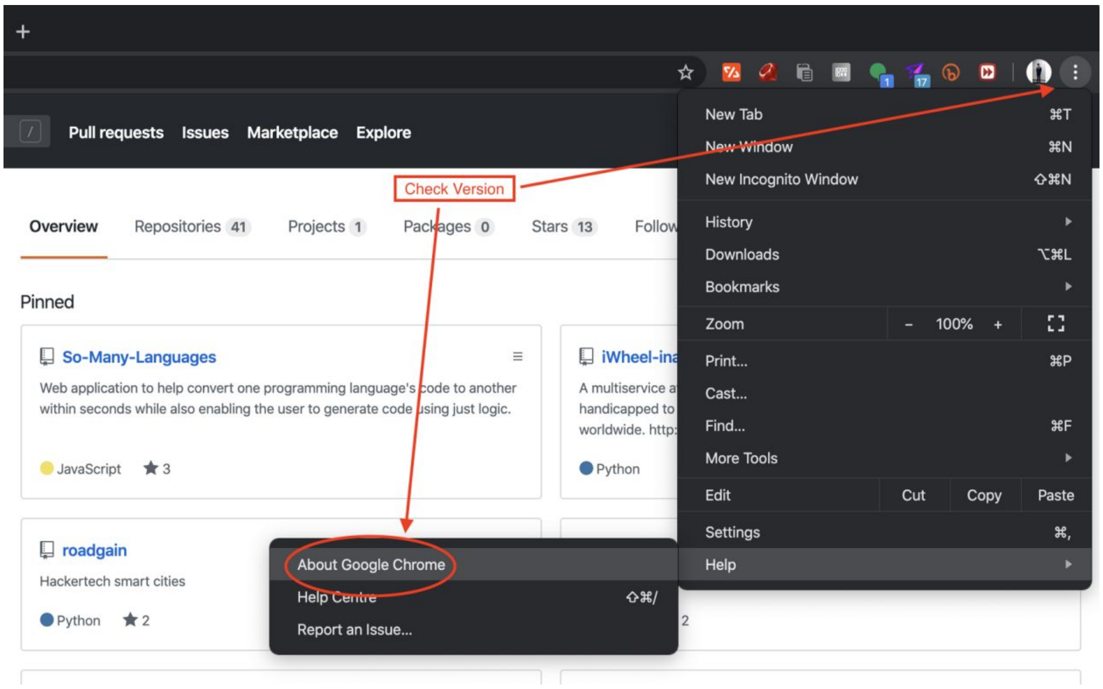
```python
import os

path = r'C:\Users\cypra\OneDrive\Desktop\CalPref\CAL-Phase2\DataP3\Ragi_Malt'
item = 'Ragi_Malt'
queries = ['Ragi Malt', 'South Ragi']
images_limit = [250,150]  #per corresponding query
driver_path = 'C:\\Users\\cypra\\OneDrive\\Desktop\\CalPref\\chromedriver.exe'

# path = 'C:\\Users\\cypra\\OneDrive\\Desktop\\CalPref\\Chapati'
os.chdir(path)

from selenium import webdriver
import time
import urllib.request
import os
from selenium.webdriver.common.keys import Keys

# query = 'Indian Dinner Chapati'

def download_images(item=item,queries=queries,images_limit=images_limit):
    sources = []
```

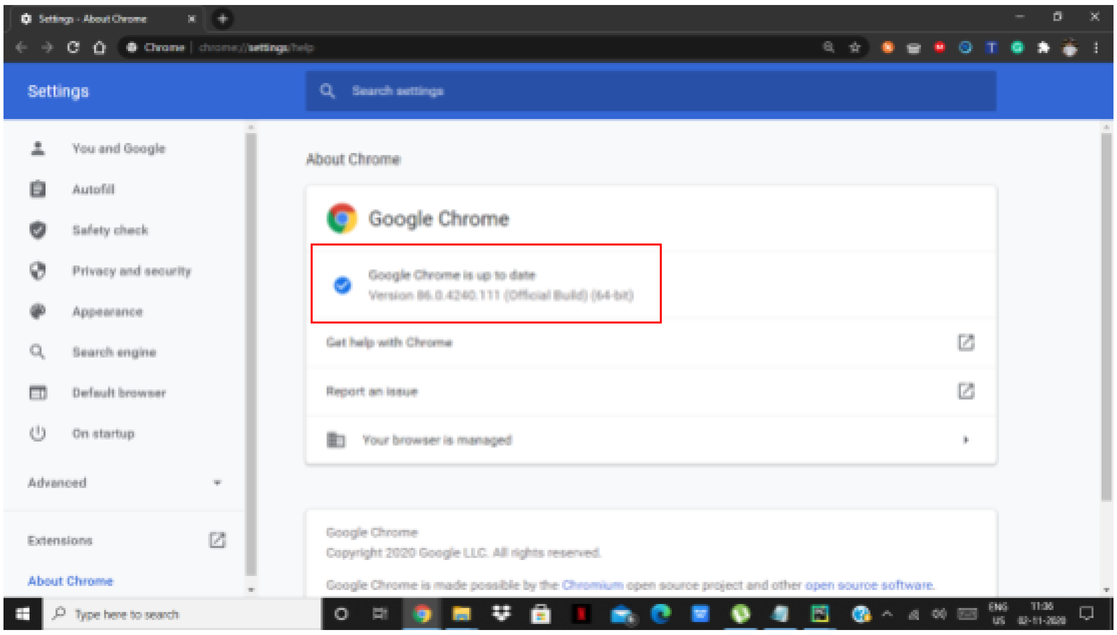
```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

(venv) C:\Users\cypra\OneDrive\Desktop\CalPref\CAL>pip install selenium
```

Step 2 : Install Chrome Driver

Before we run the code, we need to download the driver of the browser we are going to use,
This driver acts as a bridge between our code/selenium and the browser.

- For Google Chrome, check the version of Google Chrome you are using in the "About Google Chrome" section and download the respective version of chrome-driver from HERE.





About Chrome

Google Chrome

Google Chrome is up to date
Version 86.0.4240.111 (Official Build) (64-bit)

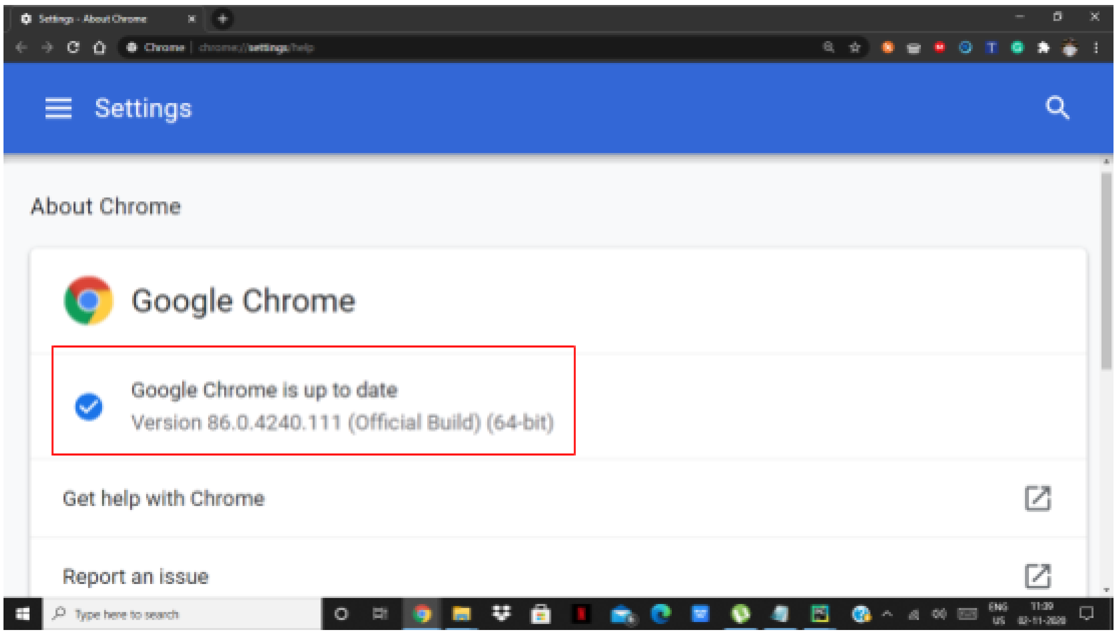
Get help with Chrome

Report an issue

Your browser is managed

Google Chrome
Copyright 2020 Google LLC. All rights reserved.

Google Chrome is made possible by the Chromium open source project and other open source software.



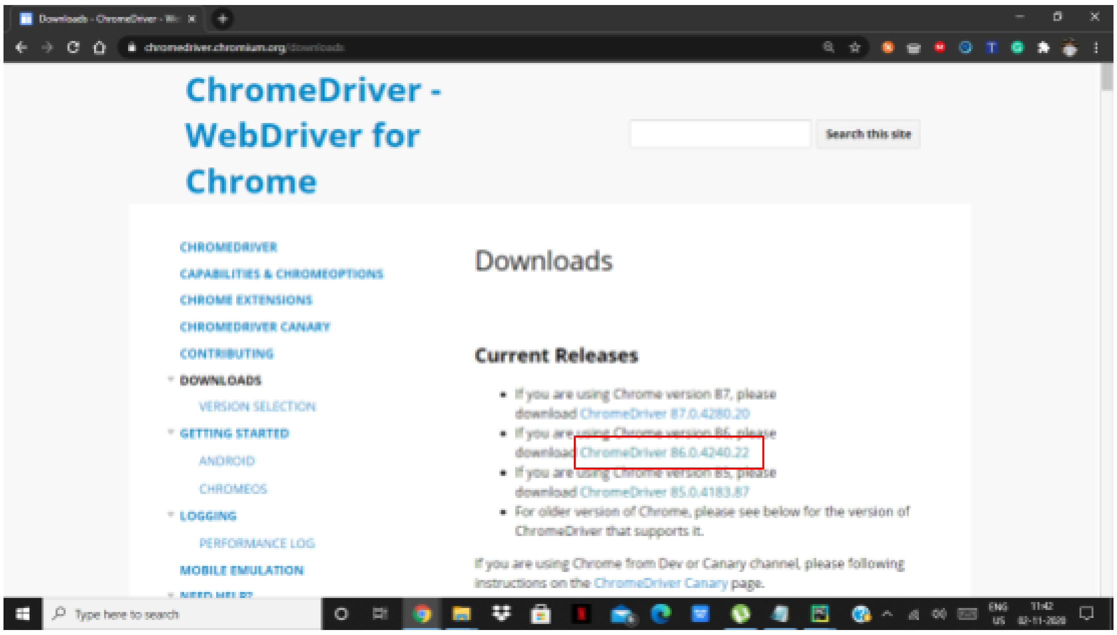
About Chrome

Google Chrome

Google Chrome is up to date
Version 86.0.4240.111 (Official Build) (64-bit)

Get help with Chrome

Report an issue



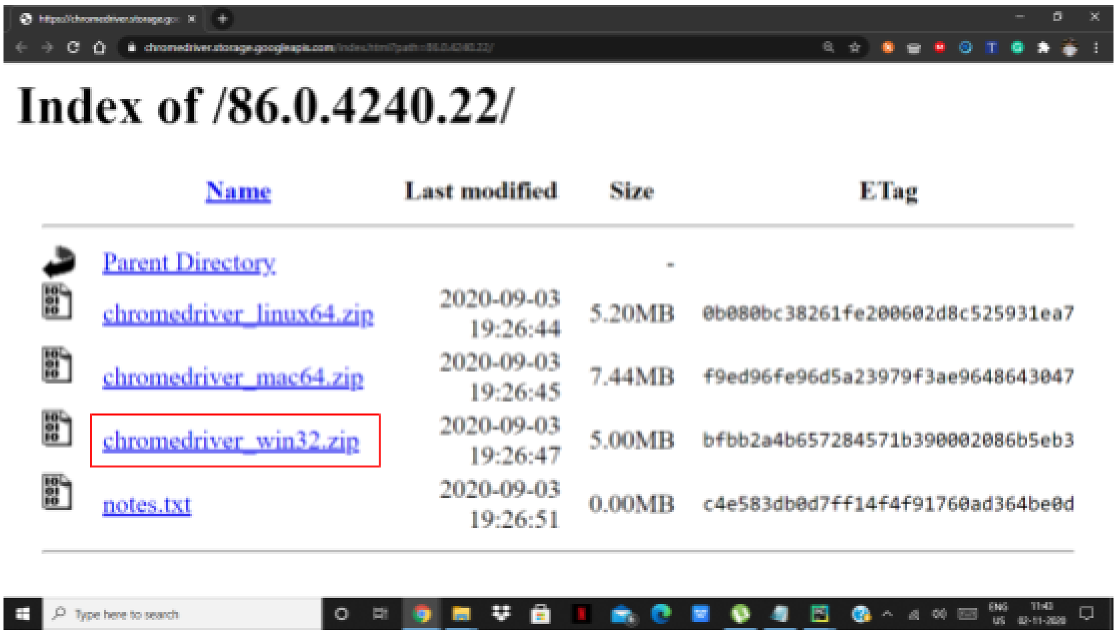
ChromeDriver - WebDriver for Chrome

Downloads

Current Releases

- If you are using Chrome version 87, please download ChromeDriver 87.0.4280.20
- If you are using Chrome version 86, please download ChromeDriver 86.0.4240.22
- If you are using Chrome version 85, please download ChromeDriver 85.0.4183.87
- For older version of Chrome, please see below for the version of ChromeDriver that supports it.

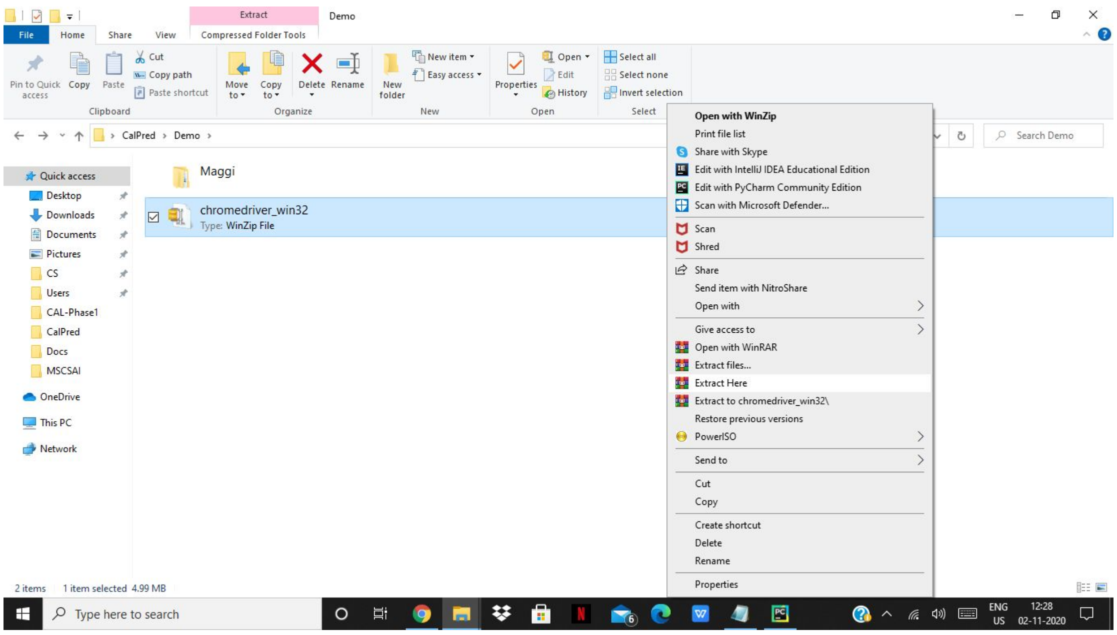
If you are using Chrome from Dev or Canary channel, please following instructions on the ChromeDriver Canary page.

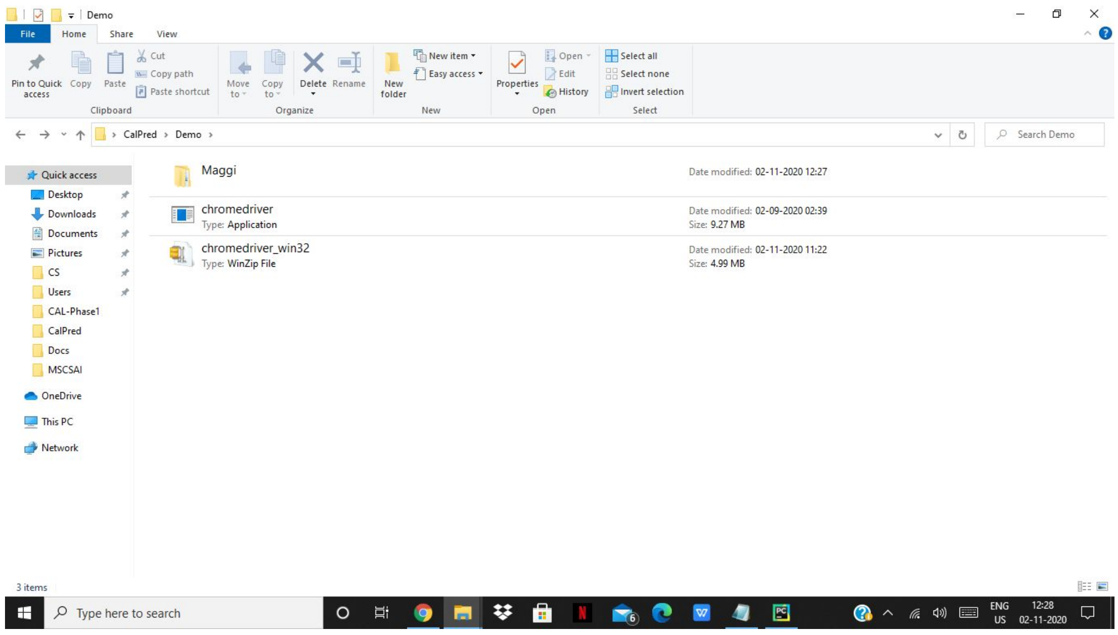


Index of /86.0.4240.22/

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2020-09-03 19:26:44	5.20MB	0b080bc38261fe200602d8c525931ea7
chromedriver_mac64.zip	2020-09-03 19:26:45	7.44MB	f9ed96fe96d5a23979f3ae9648643047
chromedriver_win32.zip	2020-09-03 19:26:47	5.00MB	bfbb2a4b657284571b390002086b5eb3
notes.txt	2020-09-03 19:26:51	0.00MB	c4e583db0d7ff14f4f91760ad364be0d

- After downloading it, unzip the file in the Demo folder.





Maggi Date modified: 02-11-2020 12:27

chromedriver Type: Application Date modified: 02-09-2020 02:39 Size: 9.27 MB

chromedriver_win32 Type: WinZip File Date modified: 02-11-2020 11:22 Size: 4.99 MB

Step 3: Edit the Variables in the Code.

Edit the variables in the first few lines of the code as follows.

- path = The location of the folder where you want to download the images.
- item = The prefix of the image names. **Make sure this is the same as the name of the item in the predefined_classes.txt list (use underscores for spaces and make the first letter of every word capital, ex - 'apple pie' = 'Apple_Pie').**
  For example, if you write "Apple" the images will be saved as "Apple1.jpg", "Apple2.jpg", "Apple3.jpg' etc.
- search_sentence = The search sentence for which you want to download the google images. For example, If you write "Rava Dosa", The code will type "Rava Dosa" in google images and download all the resulting search results/images.
- images_limit = The maximum number of images the code will download.
- driver_path = The location of the chromedriver.exe file.

Note - Do not remove the r in front of the inverted commas.

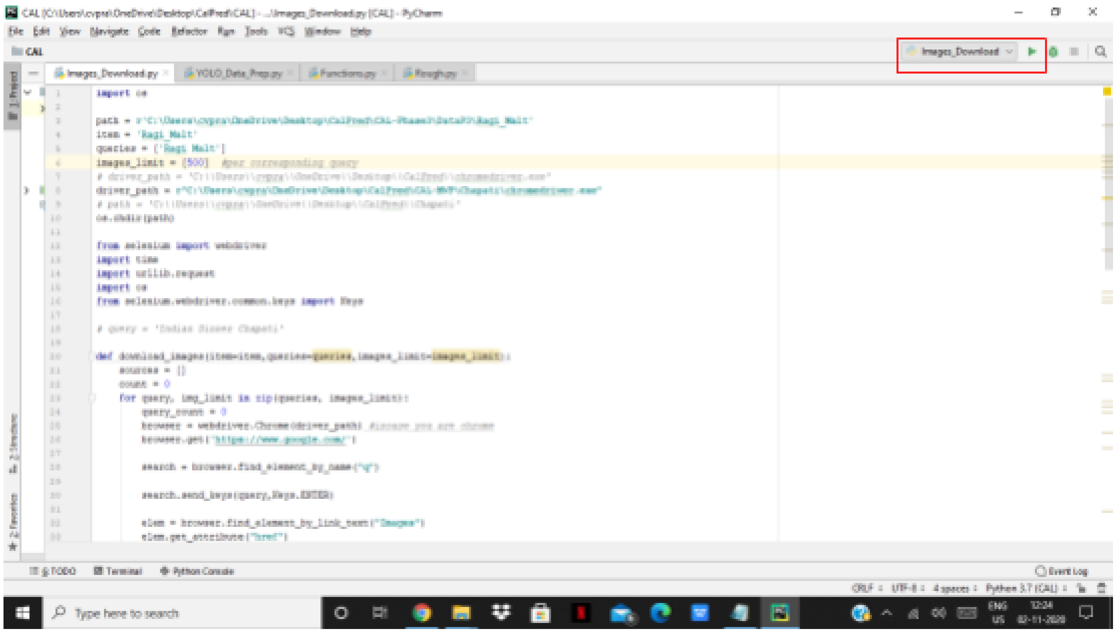
```
import os

path = r'C:\Users\cvpra\OneDrive\Desktop\CalPred\Demo\Maggi'
item = 'Maggi'
search_sentence = ['Maggi']
images_limit = [500]  #per corresponding query
driver_path = r'C:\Users\cvpra\OneDrive\Desktop\CalPred\Demo\chromedriver.exe'

os.chdir(path)
from selenium import webdriver
import time
import urllib.request
import os
```

```python
from selenium.webdriver.common.keys import Keys
def download_images(item=item,queries=search_sentence,images_limit=images_limit):
    sources = []
    count = 0
    for query, img_limit in zip(queries, images_limit):
        query_count = 0
        browser = webdriver.Chrome(driver_path) #incase you are chrome
        browser.get('https://www.google.com/')
        search = browser.find_element_by_name("q")
        search.send_keys(query,Keys.ENTER)
        elem = browser.find_element_by_link_text("Images")
        elem.get_attribute("href")
        elem.click()
        value = 0
        for i in range(20):
           browser.execute_script('scrollBy('+ str(value) +',+1000);')
           value += 1000
           time.sleep(3)
        elem1 = browser.find_element_by_id("islmp")
        sub = elem1.find_elements_by_tag_name('img')
        try:
            os.mkdir("downloads")
        except FileExistsError:
            pass
        for i in sub:
            src = i.get_attribute('src')
            try:
                if src != None:
                    src  = str(src)
                    print(src)
                    if src not in sources:
                        try:
                            urllib.request.urlretrieve(src,
os.path.join('downloads',item+str(count)+'.jpg'))
                            count+=1
                            query_count+=1
                            sources.append(src)
                            if query_count > (img_limit-1):
                                break
                        except Exception:
                            print("Exception while downloading ", Exception)
                    else: print("Already Downlaoded", src)
                else:
                    raise TypeError
            except TypeError:
                print('fail')
        print(query_count, ' images downloaded')
    print('total images downloaded: ', count)
```

```
download_images()
```

Step 4: Run the Code!

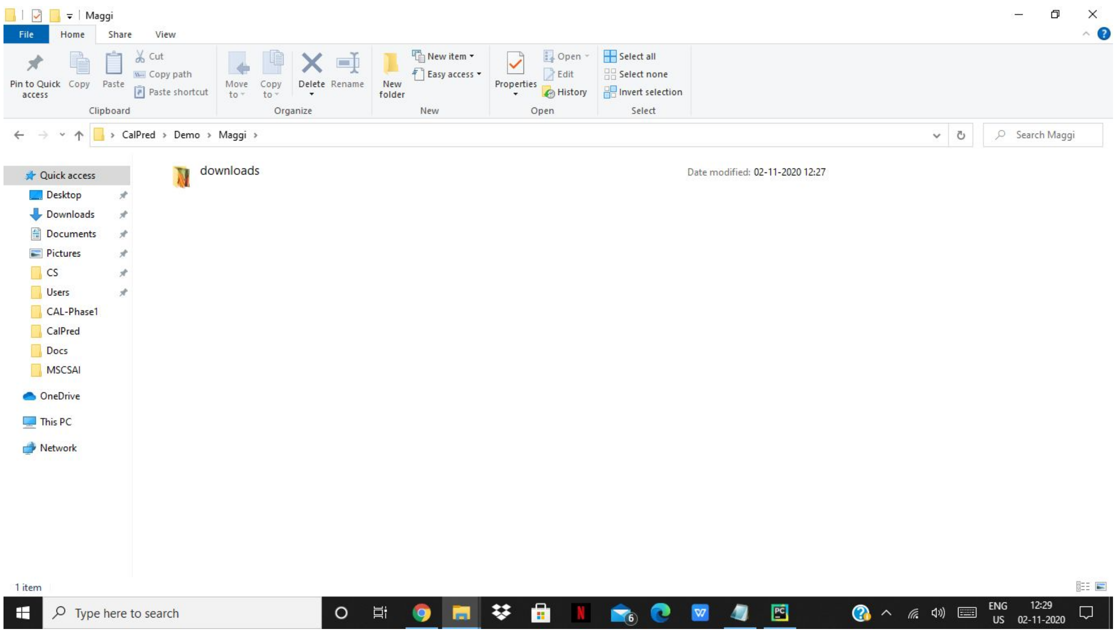
- Click the play button at the top right corner in PyCharm



CAL [C:\Users\cvpra\OneDrive\Desktop\CalPred\CAL] - ...\Images_Download.py [CAL] - PyCharm

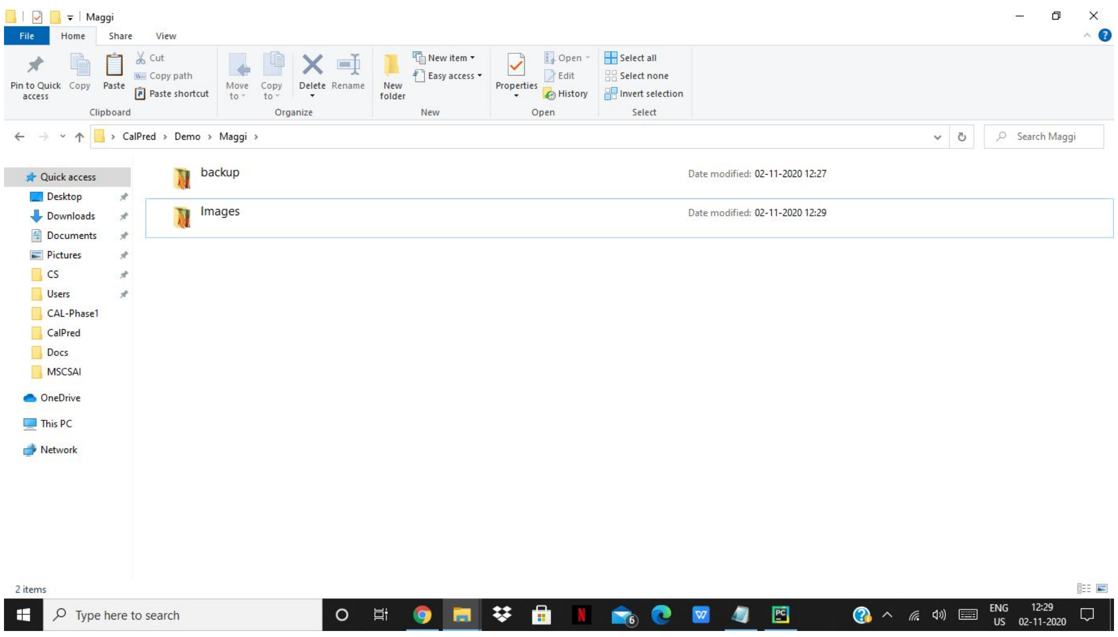
```
import os

path = r'C:\Users\cvpra\OneDrive\Desktop\CalPred\CAL-Phase1\[illegible]\Ragi_Malt'
item = 'Ragi_Malt'
queries = ['Ragi Malt']
images_limit = [500]  #per corresponding query
# driver_path = 'C:\\Users\\cvpra\\OneDrive\\Desktop\\CalPred\\chromedriver.exe'
driver_path = r"C:\Users\cvpra\OneDrive\Desktop\CalPred\CAL-MVP\Chapati\chromedriver.exe"
# path = 'C:\\Users\\cvpra\\OneDrive\\Desktop\\CalPred\\Chapati'
os.chdir(path)

from selenium import webdriver
import time
import urllib.request
import os
from selenium.webdriver.common.keys import Keys

# query = 'Indian Dinner Chapati'

def download_images(item=item,queries=queries,images_limit=images_limit):
    sources = []
    count = 0
    for query, img_limit in zip(queries, images_limit):
        query_count = 0
        browser = webdriver.Chrome(driver_path) #incase you are chrome
        browser.get("https://www.google.com/")

        search = browser.find_element_by_name("q")

        search.send_keys(query,Keys.ENTER)

        elem = browser.find_element_by_link_text("Images")
        elem.get_attribute("href")
```

- After the code has finished running, you will find a folder called "downloads" in the path specified.



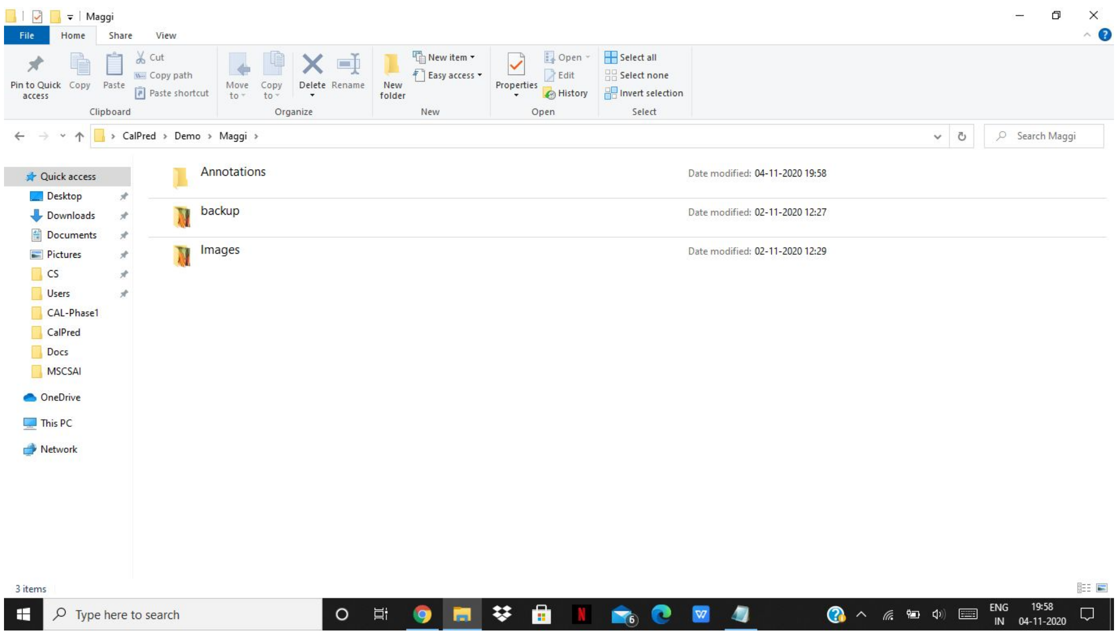
downloads	Date modified: 02-11-2020 12:27

- Rename the folder as "backup", Create a copy of that folder and rename it as "Images". We will be working with this "Images" folder.



- Finally, Open the Images folder and delete the irrelevant/useless images. But, keep in mind that we need around 200 images for each item/class. A few duplicates and low resolution images are fine, if the number of images is too low.

- Also create a folder called Annotations where you will save the annotated text files of the images.


Maggi
File
Home
Share
View
CalPred › Demo › Maggi
Annotations
Date modified: 04-11-2020 19:58
backup
Date modified: 02-11-2020 12:27
Images
Date modified: 02-11-2020 12:29
3 items

Alternate Code

If the above code does not work for any reason, you can follow the steps under the Downloading_Images section in this document.

Folder Structure

Maintain the following folder structure:
A Data Folder (here Data) inside which > you have only the class data folders (ex- Maggi) > and each class folder must contain a folder called ‘Images’ with the Images and a folder called ‘Annotations’ containing the corresponding annotations.

Data Annotation

Step 1: Install LabelImg

LabelImg is a tool to perform Image Annotation.

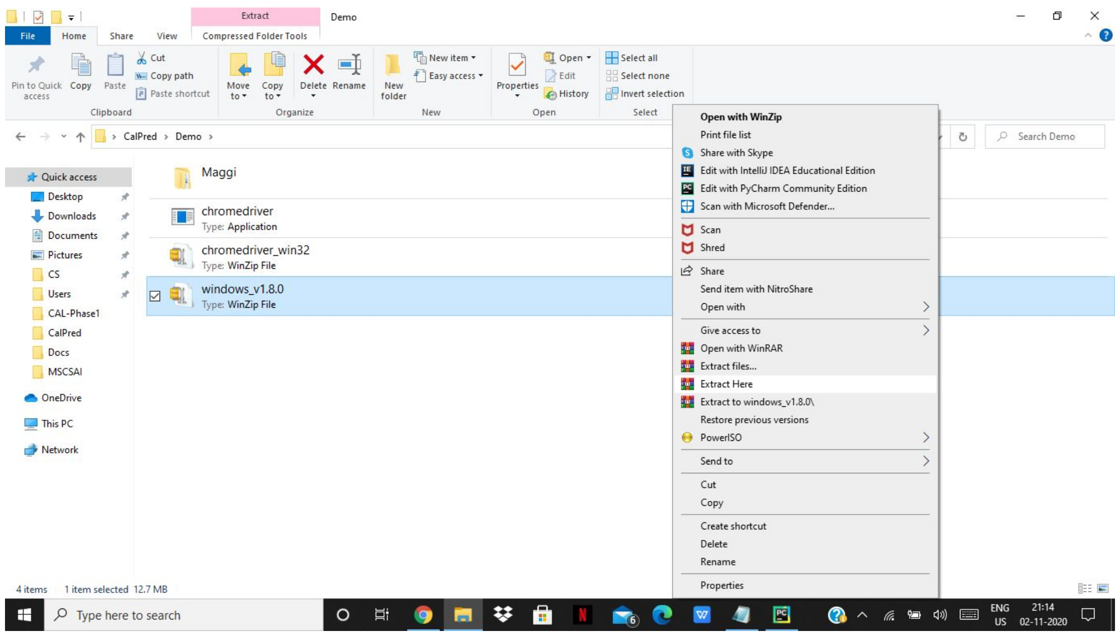
FOR WINDOWS

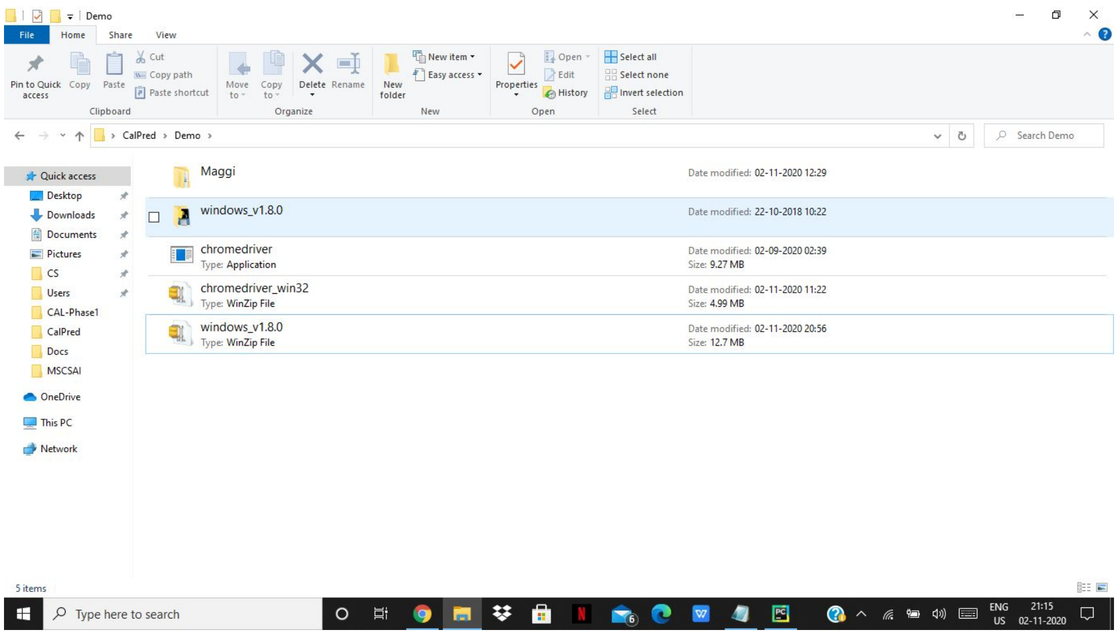
- Install PyQt5 and lxml (2 python packages which are needed to run LabelImg) running the following command in the terminal.

```
pip install pyqt5 lxml
```

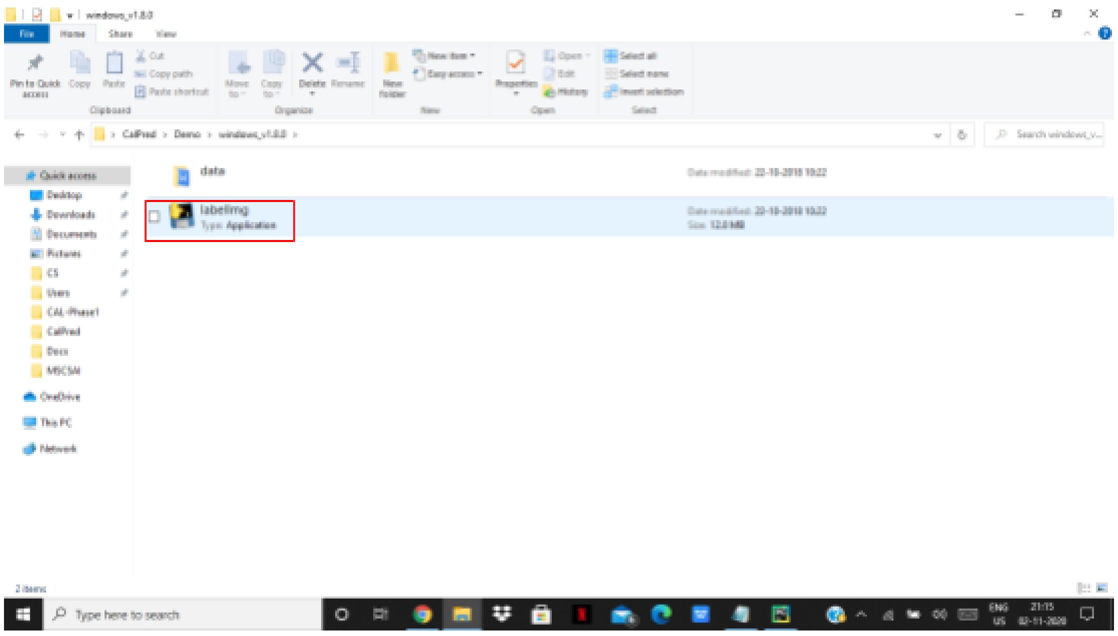
- Download, unzip and run the **LabelImg.exe** file via

  https://www.dropbox.com/s/kqoxr10l3rkstqd/windows_v1.8.0.zip?dl=1





Maggi Date modified: 02-11-2020 12:29
windows_v1.8.0 Date modified: 22-10-2018 10:22
chromedriver Type: Application Date modified: 02-09-2020 02:39 Size: 9.27 MB
chromedriver_win32 Type: WinZip File Date modified: 02-11-2020 11:22 Size: 4.99 MB
windows_v1.8.0 Type: WinZip File Date modified: 02-11-2020 20:56 Size: 12.7 MB



FOR MACOS

- Run the following commands one after the other in the PyCharm terminal:

```
git clone https://github.com/tzutalin/labelImg.git
```

```
pip3 install pyqt5 lxml
```

```
cd labelImg
make qt5py3
```

- Run this command in the terminal to open LabelImg.

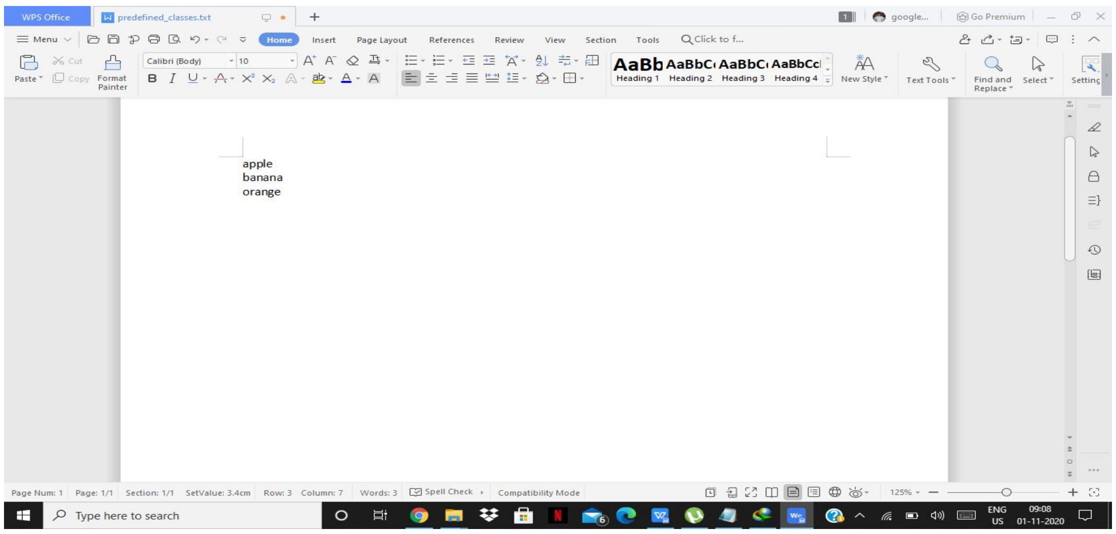
```
python3 labelImg.py
```

Step 2: Edit the predefined_classes.txt file

- Inside the LabelImg folder - **windows_v1.8.0** go to the > "**data**" folder > and edit the "**predefined_classes.txt**" file to include your list of items. Each line should contain the name of one class.

NB: Make sure that everyone is using the same predefined_classes.txt file before you start annotating. For ex - If person A is labeling apple data and is using a predefined_classes.txt file with items in the order [apple, banana, orange], then person B labeling orange should also have the items in the same order i.e. [apple, banana, orange] in his predefined_classes.txt file.

For example, if you want your model to recognize apple, banana and orange, then your file should look like this:

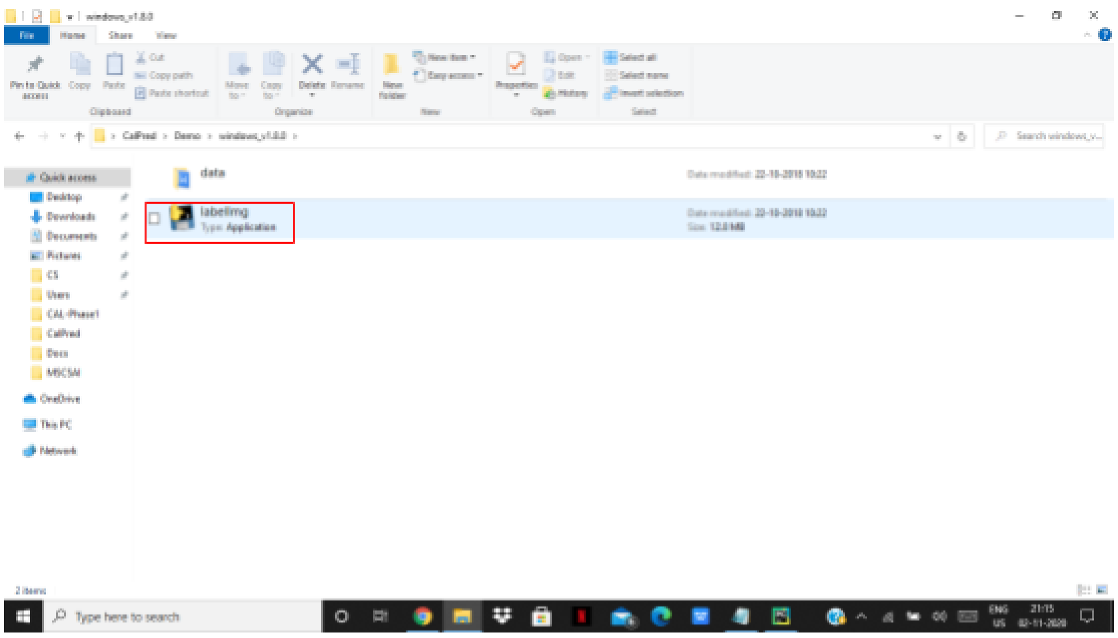


```
apple
banana
orange
```

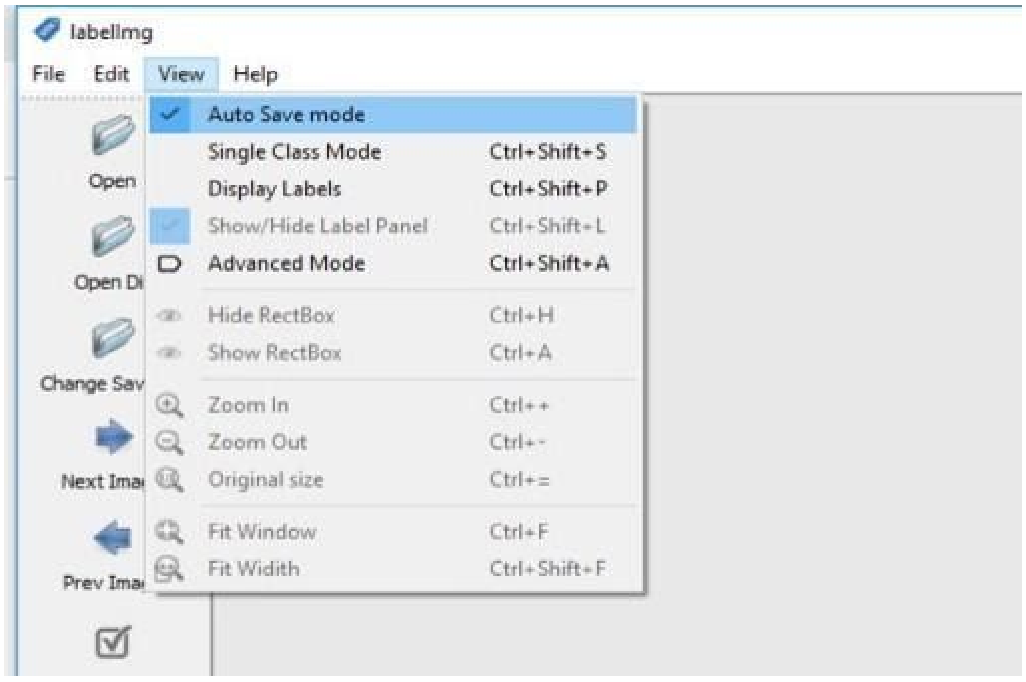
Naming Convention : For the sake of maintaining uniformity the class names should use underscores instead of space and make the first letter of every word capital. For ex - 'toasted bread' would be 'Toasted_Bread'.

Step 3: Setup LabelImg

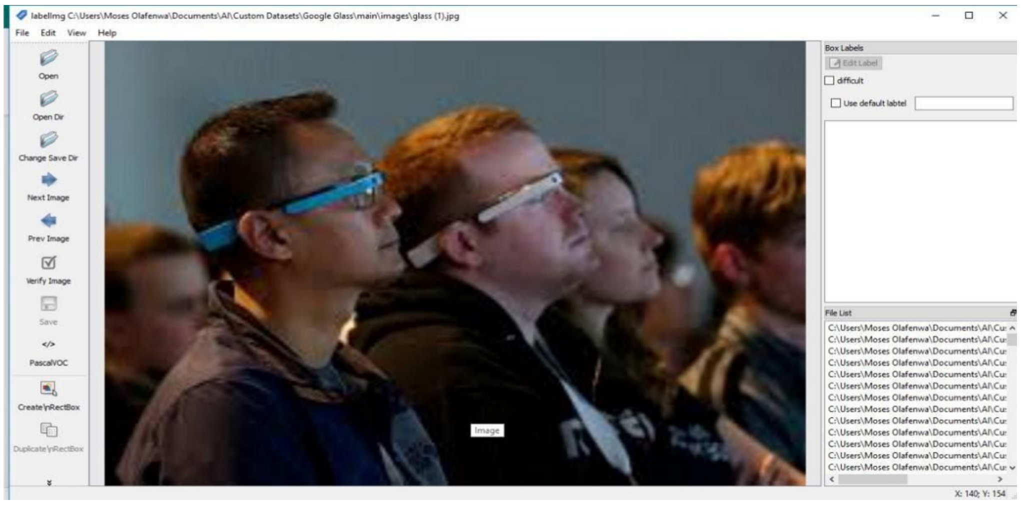
- Start LabelImg by clicking on the **LabelImg.exe** file in the **windows_v1.8.0** folder.



- Then go to your LabelImg menu, select "**View**" and make sure "**Auto Save Mode**" is checked.

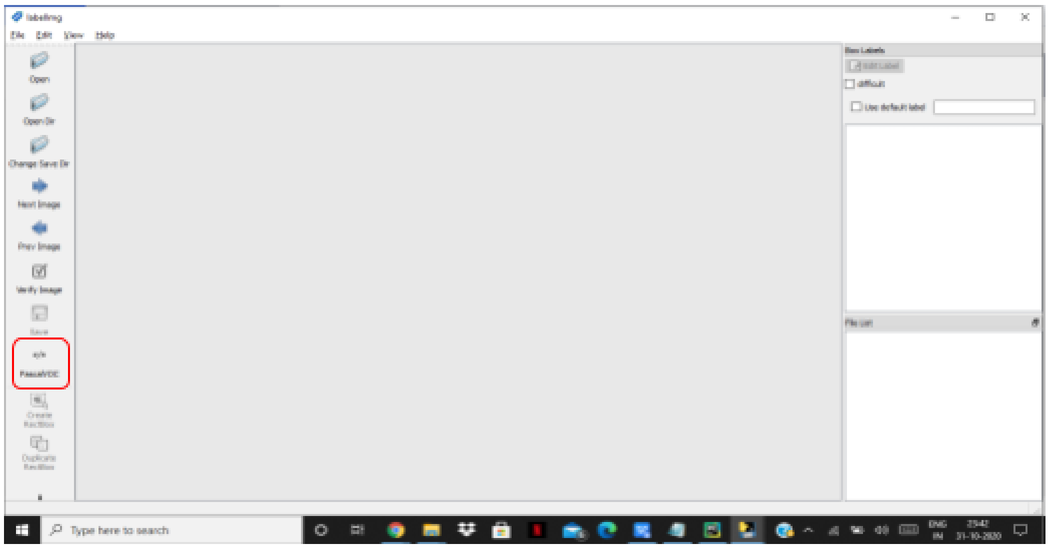


- Click on "**Open Dir**" on the top-left and select your "**Images**" directory where your images are kept. The first image in your folder will be shown as seen in the example below.



- Click on the "**Change Save Dir**" on the top-left and select your "**Annotations**" folder. This is where the generated txt file containing the annotation for your images will be stored.

- Change the format from PascalVOC to YOLO by clicking on it.



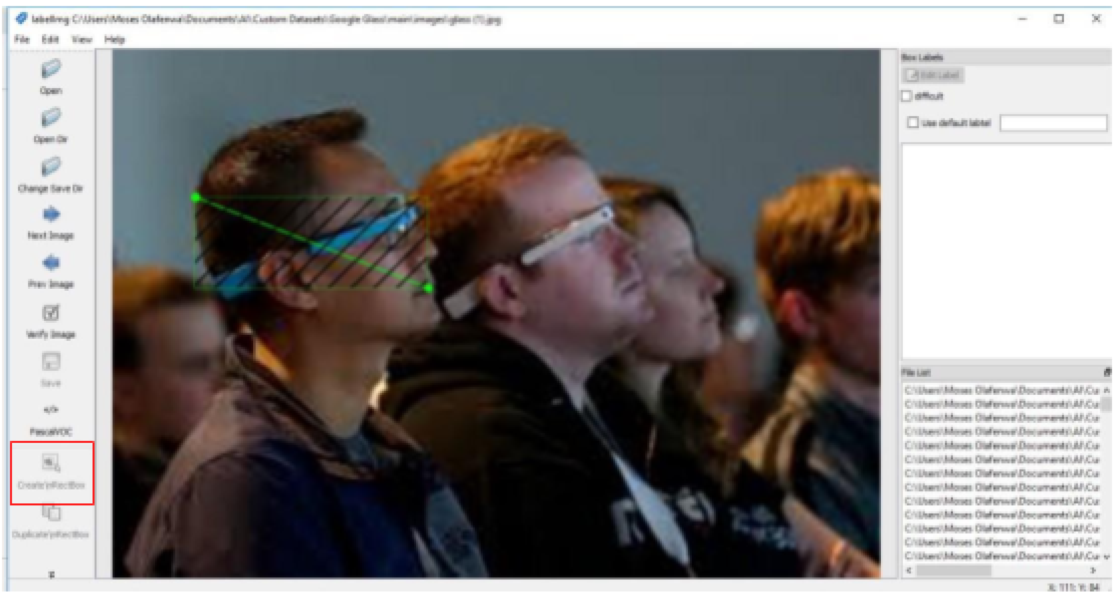
Step 4: Start Annotating!

Keep the following points in mind while you annotate:

1)The images should only have dishes which are present in the predefined_classes.txt file.

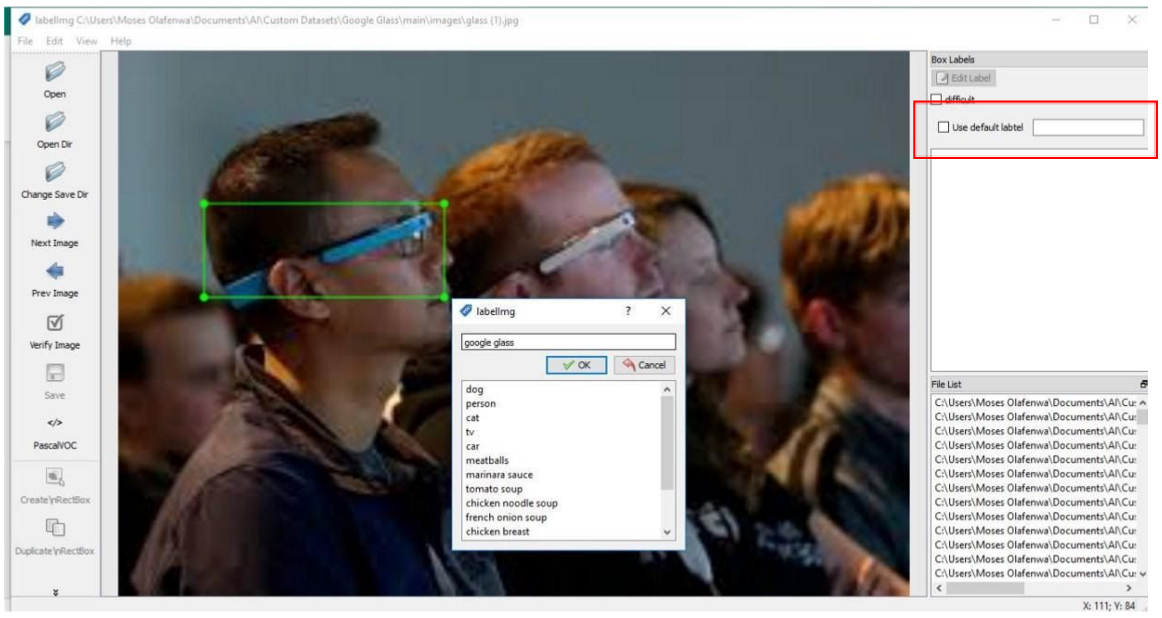
2) You must annotate all the dishes in the image which are present in the predefined_classes file. For ex - if your image of chole has puri in it, and even puri is in the predefined_classes file, you must draw a box around puri as well.

- Click on the "**Create RectBox**" button (or press '**w**') on the left-bottom and draw a box around the objects you want to annotate as seen in the images below.



- Then once drawn, enter the name of the object in the **pop-up** box and select the "**OK**" button as seen in the example below.

Note - If you are annotating the same object in all the images, tick the **"Use default label"** option and type the label in the box beside it.



- Once you are done, click the "**Next Image**" button (or press "**d**") on the middle-left to annotate another image.

- Continue this process until you are done annotating all your images.

As you are annotating your images, the txt file containing your box annotations are saved for each image in the "**annotations**" folder.

N.B: Take note that the annotation txt file for each image is saved using the name of the image file. For example:

- you have images **image_1.jpg**, **image_2.jpg** …… **image_z.jpg**
- the txt annotations file will be saved as **image_1.txt**, **image_2.txt,**…. **image_z.txt**

Note: Once you have completed annotating all the food images (Chole, Puri etc), create a back-up by making a copy of the Data folder or make a zip file of the folder by right clicking on it > select Send to > Compressed (zipped) folder.

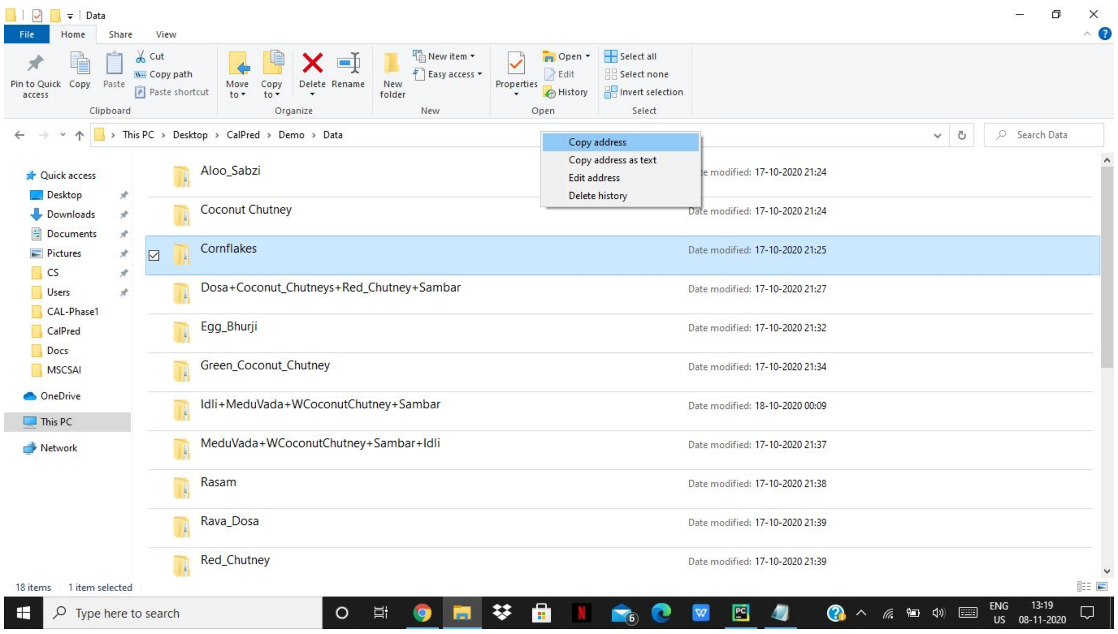
Model Training

Step 1: Train and Test Split

Once we have finished annotating, we need to put the images and the corresponding annotation files in the same folder. Furthermore, we need to split the dataset for training training (consisting about 80% of the dataset) and testing (consisting of about 20% of the dataset)

- To do this, run the code below after filling the 3 variables given in the first few lines as follows.

  directory_path = The path to the directory where the data for each class is present.



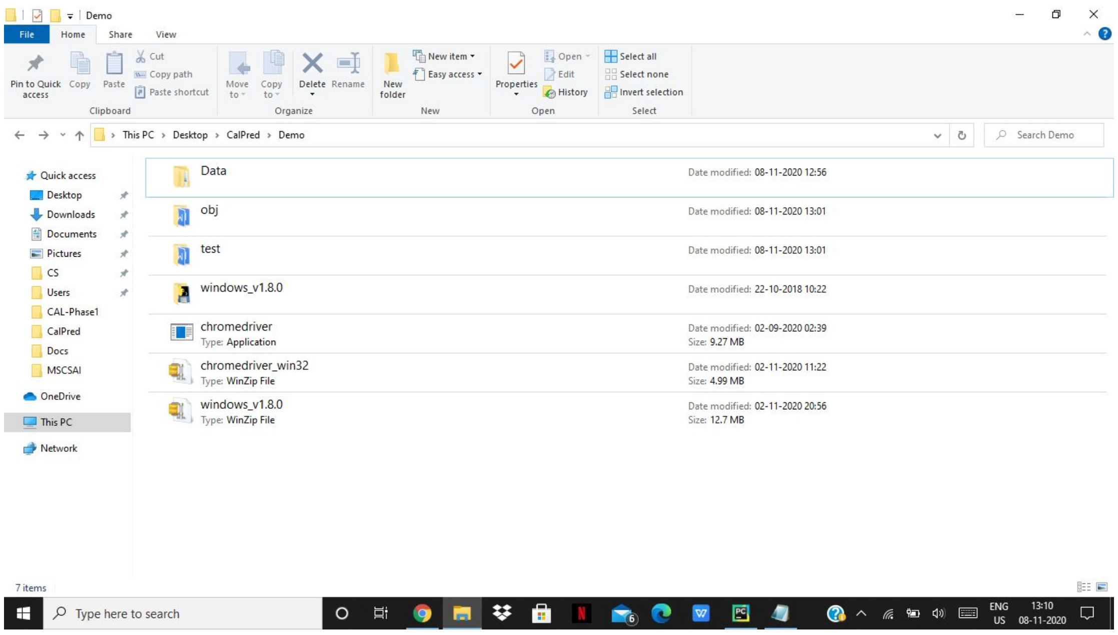
This PC › Desktop › CalPred › Demo › Data

Aloo_Sabzi	Date modified: 17-10-2020 21:24
Coconut Chutney	Date modified: 17-10-2020 21:24
Cornflakes	Date modified: 17-10-2020 21:25
Dosa+Coconut_Chutneys+Red_Chutney+Sambar	Date modified: 17-10-2020 21:27
Egg_Bhurji	Date modified: 17-10-2020 21:32
Green_Coconut_Chutney	Date modified: 17-10-2020 21:34
Idli+MeduVada+WCoconutChutney+Sambar	Date modified: 18-10-2020 00:09
MeduVada+WCoconutChutney+Sambar+Idli	Date modified: 17-10-2020 21:37
Rasam	Date modified: 17-10-2020 21:38
Rava_Dosa	Date modified: 17-10-2020 21:39
Red_Chutney	Date modified: 17-10-2020 21:39

18 items 1 item selected

  train_dir = Create a folder called ‘**obj**’ in the parent directory and give its path. This is where the training images and their corresponding annotation files will be stored.

  test_dir = Create a folder called ‘**test**’ in the parent directory and five its path. This is where the testing images and their corresponding annotation files will be stored.

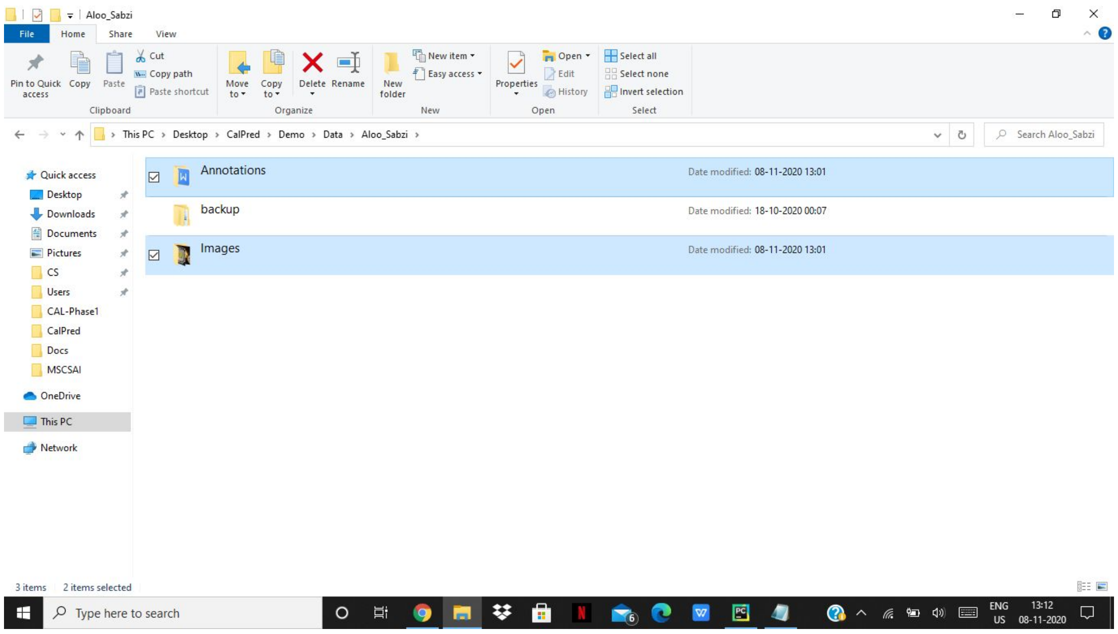


This PC › Desktop › CalPred › Demo

Data	Date modified: 08-11-2020 12:56
obj	Date modified: 08-11-2020 13:01
test	Date modified: 08-11-2020 13:01
windows_v1.8.0	Date modified: 22-10-2018 10:22
chromedriver Type: Application	Date modified: 02-09-2020 02:39 Size: 9.27 MB
chromedriver_win32 Type: WinZip File	Date modified: 02-11-2020 11:22 Size: 4.99 MB
windows_v1.8.0 Type: WinZip File	Date modified: 02-11-2020 20:56 Size: 12.7 MB

7 items

Note - Make Sure that your **directory path’s** data folder has one folder for each class inside which there are 2 folders called **“Images” and “Annotations”.**



This PC › Desktop › CalPred › Demo › Data › Aloo_Sabzi ›

Search Aloo_Sabzi

Annotations Date modified: 08-11-2020 13:01
backup Date modified: 18-10-2020 00:07
Images Date modified: 08-11-2020 13:01

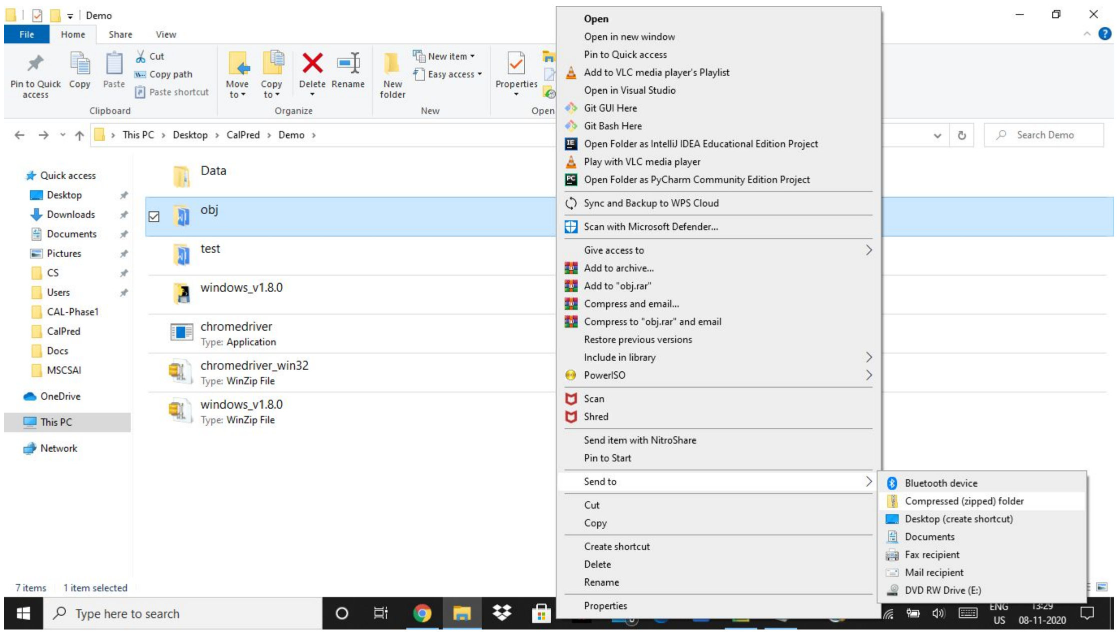
3 items 2 items selected

```
import shutil
import os

directory_path = r'C:\Users\cvpra\OneDrive\Desktop\CalPred\Demo\Data'
train_dir = r'C:\Users\cvpra\OneDrive\Desktop\CalPred\Demo\obj'
test_dir = r'C:\Users\cvpra\OneDrive\Desktop\CalPred\Demo\test'



def check_matching(dir_path):
   images_list = []
   for file in os.listdir(r'{}'.format(dir_path) + r'\Images'):
       if file.split('.')[1] == 'jpg':
           images_list.append(file)
   print(len(images_list), 'images')
   annotations_list = []
   for file in os.listdir(r'{}'.format(dir_path) + r'\Annotations'):
       if file.split('.')[1] == 'txt' and file.split('.')[1] != 'classes':
           annotations_list.append(file)
   print(len(annotations_list), 'annotations')
   matching_images = []
   matching_annotations = []
   missing_images = []
   missing_annotations = []
   if len(images_list) > len(annotations_list):
       for i in images_list:
           a = (i.split('.')[0] + '.txt')
           if a in annotations_list:
               matching_images.append(i)
               matching_annotations.append(a)
           else:
               missing_annotations.append(a)
       print('Images > Annotations', len(missing_annotations), ' annotation.txt
files are missing !', '\n', missing_annotations)
```

```
    elif len(annotations_list) > len(images_list):
        for a in annotations_list:
            i = (a.split('.')[0] + '.jpg')
            if i in images_list:
                matching_images.append(i)
                matching_annotations.append(a)
            else:
                missing_images.append(i)
        print('Annotations > Images', len(missing_images), ' images.jpg files are
missing !', missing_images)
    elif len(images_list) == len(annotations_list):
        for i in images_list:
            a = (i.split('.')[0] + '.txt')
            if a in annotations_list:
                matching_images.append(i)
                matching_annotations.append(a)
            else:
                missing_annotations.append(a)
        print('Images == Annotations', len(missing_annotations), '  annotation.txt
files are missing !', '\n', missing_annotations)
    return matching_images, matching_annotations, missing_images,
missing_annotations


def tt_split(dir_path, train_percent, train_dir, test_dir ):
    mi,ma,misi,misa = check_matching(r'{}'.format(dir_path))
    num_imgs = len(mi)
    train_images = (train_percent * num_imgs) / 100
    count = 0
    for i,a in zip(mi, ma):
        if count < train_images:
            try:
                shutil.move(r'{}'.format(dir_path)+r'\Images'+ r'\{}'.format(i),
r'{}'.format(train_dir)+r'\{}'.format(i))
            except: print(i,Exception)
            try:
                shutil.move(r'{}'.format(dir_path) + r'\Annotations' +
r'\{}'.format(a), r'{}'.format(train_dir)+ r'\{}'.format(a))
            except:
                print(a, Exception)

        else:
            try:
                shutil.move(r'{}'.format(dir_path) + r'\Images' +
r'\{}'.format(i),
                            r'{}'.format(test_dir)+ r'\{}'.format(i))
            except:
                print(i, Exception)
            try:
                shutil.move(r'{}'.format(dir_path) + r'\Annotations' +
r'\{}'.format(a),
                            r'{}'.format(test_dir)+ r'\{}'.format(a))
            except:
                print(a, Exception)
        count += 1
```

```
for dp in [directory_path + r'\{}'.format(i) for i in
os.listdir(directory_path)]:
    tt_split(dp,80,train_dir, test_dir)
```

- Zip the obj and train folders so that we can easily upload them to our drive. Before uploading the obj.zip and test.zip files to your drive (or a shared folder), rename them to have a unique name in the shared folder while making sure that they start with ‘obj’ and ‘test’. For ex - if your ‘obj.zip’ file has the data for apple pie and banana rename it as ‘obj-apple_pie-banana.zip’ .



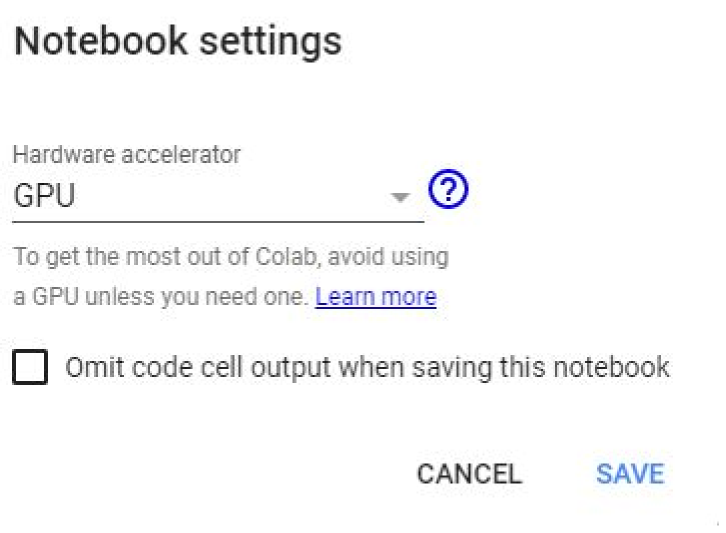
Step 2: Open and Setup Colab

Google Colaboratory or **“Colab”** is a google product which can be used to write, share and execute python code along with images, graphs etc through the browser, while providing free access to powerful computational resources like Graphical Processing Units (GPUs).

- Go to the colab notebook HERE

- Change the runtime environment to GPU

In the **edit menu** at top left > select **notebook settings** > turn the hardware accelerator to **GPU**. This will help us to utilize a GPU in colab and make the training faster.



Notebook settings

Hardware accelerator
GPU

To get the most out of Colab, avoid using a GPU unless you need one. Learn more

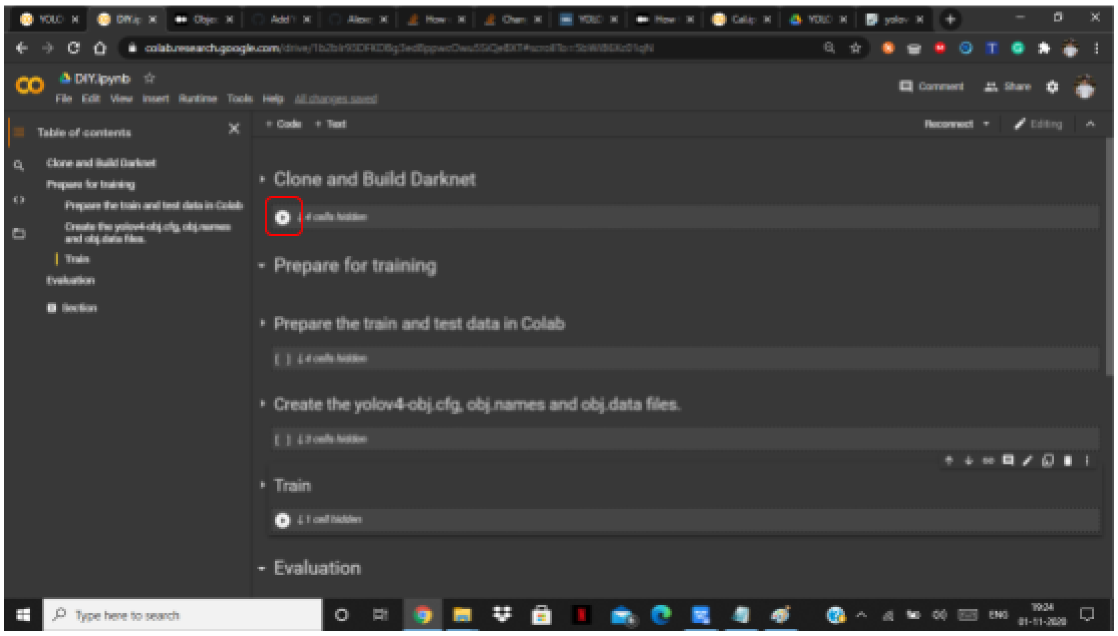
☐ Omit code cell output when saving this notebook

CANCEL SAVE

Step 3: Clone and Build Darknet

The algorithm we are going to use for object detection is known as YOLOv4 (You only look once). Darknet is a framework using which yolo is implemented.

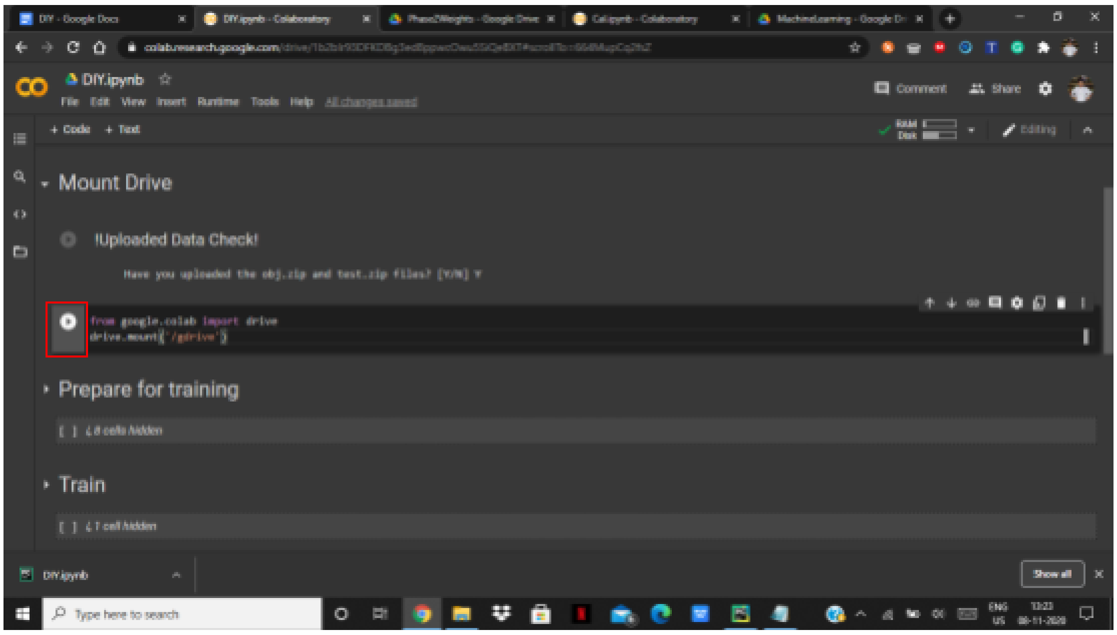
- Clone and Build Darknet by running the cells under that section so that we can train the model.


DIY.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Table of contents
Clone and Build Darknet
Prepare for training
Prepare the train and test data in Colab
Create the yolov4-obj.cfg, obj.names and obj.data files.
Train
Evaluation
Clone and Build Darknet
4 cells hidden
Prepare for training
Prepare the train and test data in Colab
4 cells hidden
Create the yolov4-obj.cfg, obj.names and obj.data files.
3 cells hidden
Train
1 cell hidden
Evaluation
Type here to search

Step 4: Mount Google Drive in Colab

- Upload the training and testing data, i.e. the **obj.zip** and **test.zip** files to your google drive.

- Now mount your drive to Colab running the following cell. This will generate a link, open that and copy the confirmation code back to the cell.


Mount Drive

!Uploaded Data Check!

```
Have you uploaded the obj.zip and test.zip files? [Y/N] Y
```

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

Prepare for training

[ ] ↳ 8 cells hidden

Train

[ ] ↳ 1 cell hidden

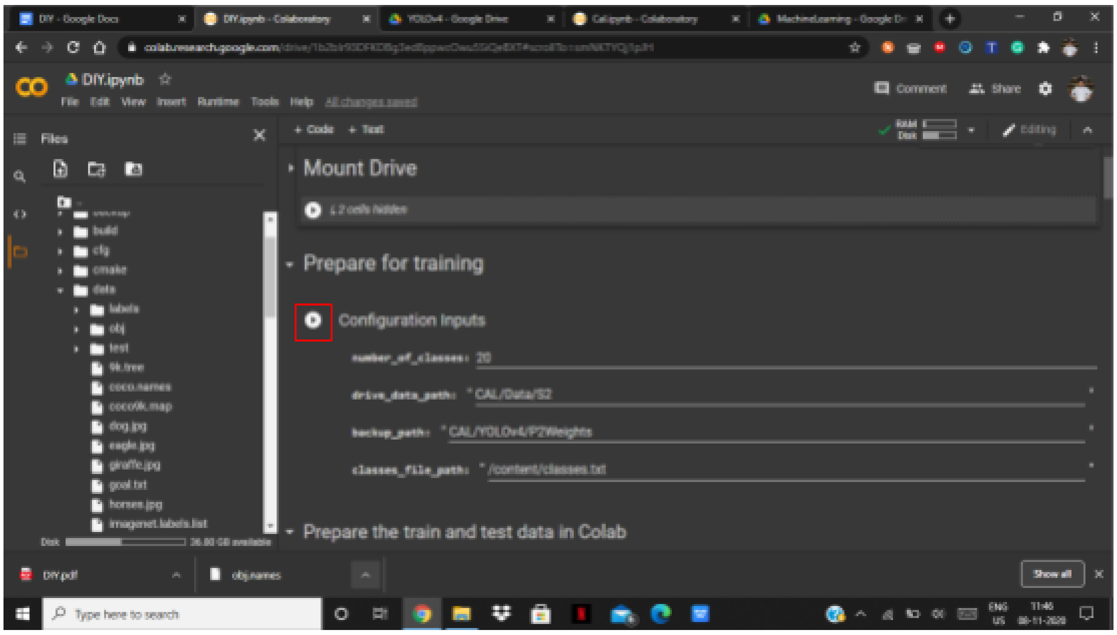
Step 5: Prepare data for training

Once we have the training and testing data ready, we need to create 2 files called **train.txt and test.txt.** These files should contain the paths to the images which we are going to train and test.

For example, if train.txt contains the paths shown below, the model would train on img1, img2. img3 and img4 in the obj folder.

```
data/obj/img1.jpg
data/obj/img2.jpg
data/obj/img3.jpg
data/obj/img4.jpg
```

- Under the Prepare for Training cell, Enter the configuration inputs in the form and click the play button beside it.


Mount Drive

Prepare for training

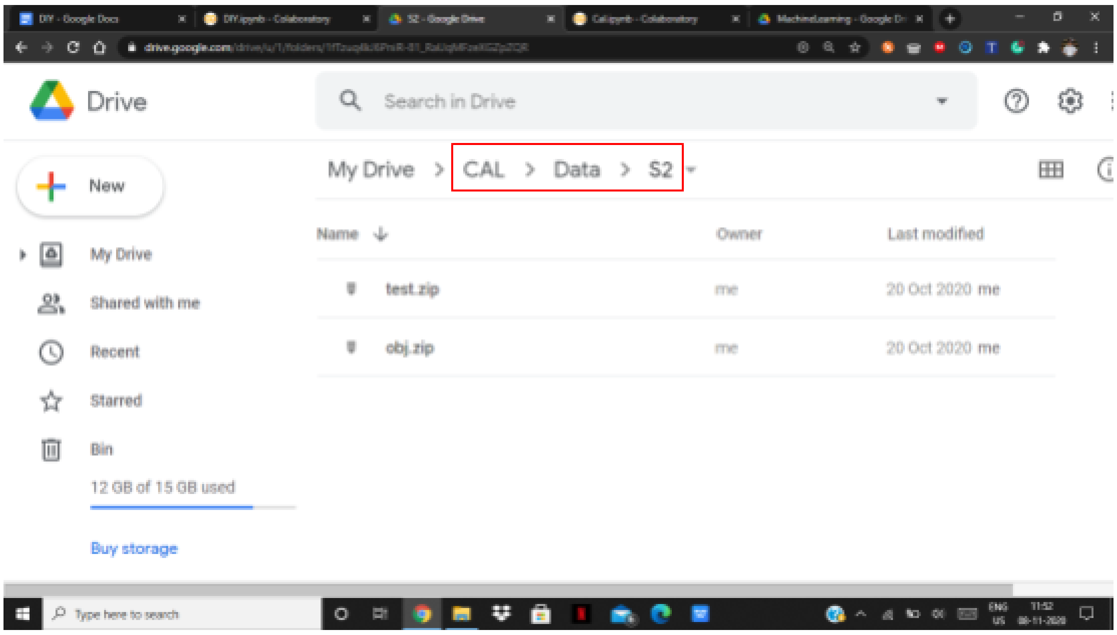
Configuration Inputs

```
number_of_classes: 20
drive_data_path: "CAL/Data/S2"
backup_path: "CAL/YOLOv4/P2Weights"
classes_file_path: "/content/classes.txt"
```

Prepare the train and test data in Colab

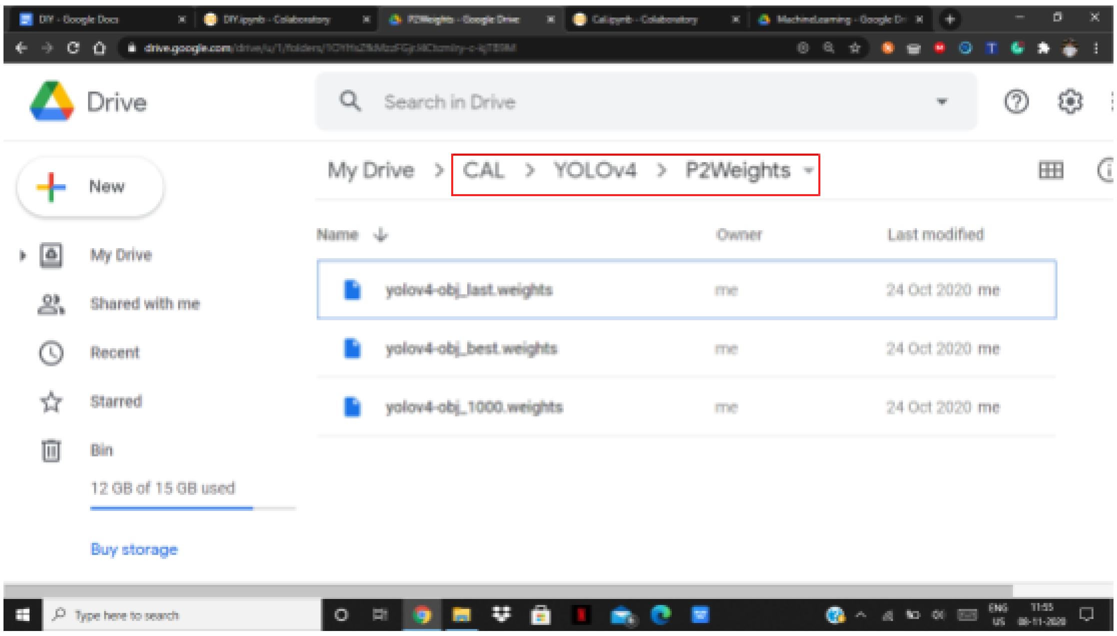
number_of_classes = The number of items you are training your model on. For example, if you are training your model to recognize apple and banana, number_of_classes = 2.

drive_data_path = The path of the folder in the My Drive section of your google drive, where the training and testing data folders, i.e. obj.zip and test.zip are stored. For example, for the below image drive_data_path = “CAL/DATA/S2”. If your data is in a shared folder i.e. the “Shared with me” section, create a shortcut to it in your My Drive section and then give its path.


My Drive > CAL > Data > S2

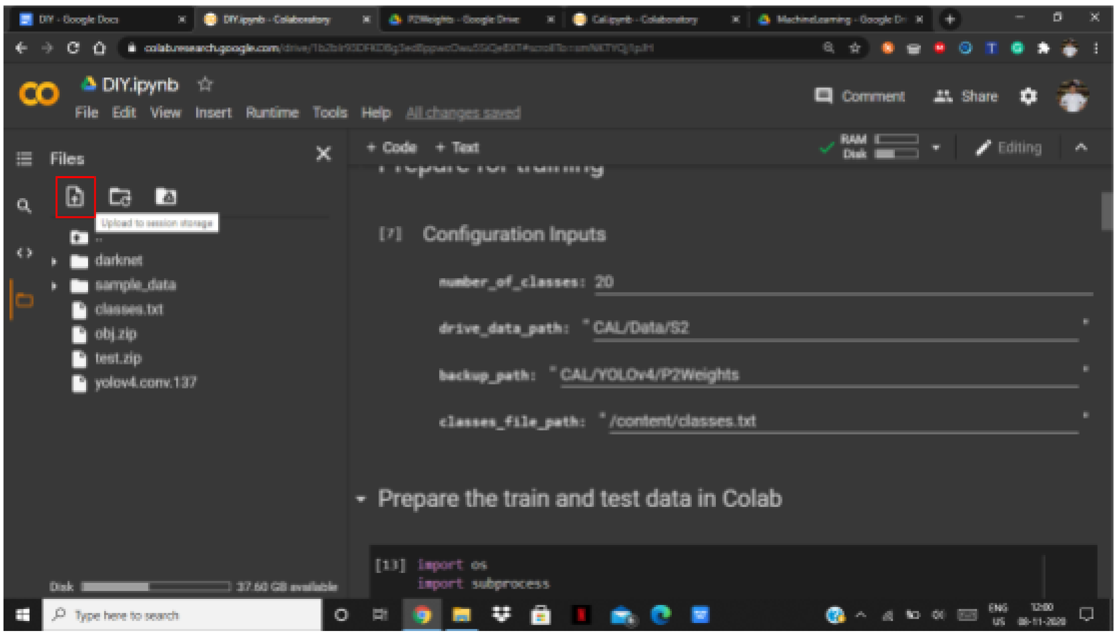
Name	Owner	Last modified
test.zip	me	20 Oct 2020 me
obj.zip	me	20 Oct 2020 me

backup_path = The location to the folder in My Drive where the results of our training, i.e. the model weights will be saved. For example, if you want to save them in a folder called P2Weights as show below, backup_path = “CAL/YOLOv4/P2Weights”



Name	Owner	Last modified
yolov4-obj_last.weights	me	24 Oct 2020 me
yolov4-obj_best.weights	me	24 Oct 2020 me
yolov4-obj_1000.weights	me	24 Oct 2020 me

classes_file_path = Upload the text file containing the names of all your classes/objects that you created while annotating your images. Then copy its path and paste it. For example, classes_file_path = "/content/classes.txt".

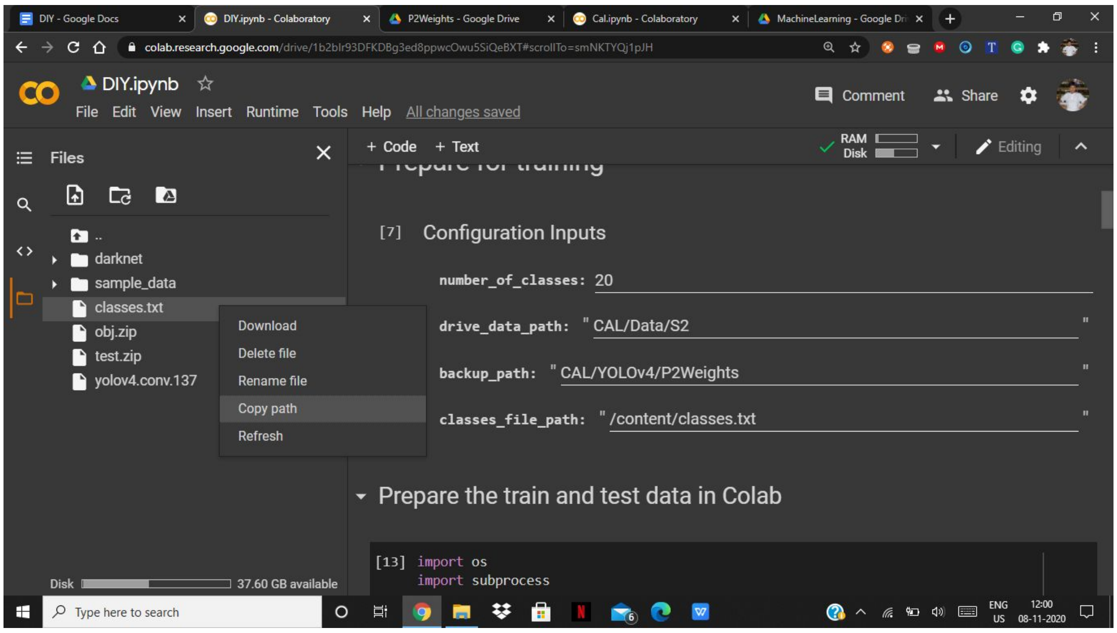


[7] Configuration Inputs

```
number_of_classes: 20
drive_data_path: "CAL/Data/S2
backup_path: "CAL/YOLOv4/P2Weights
classes_file_path: "/content/classes.txt
```

Prepare the train and test data in Colab

```
[13] import os
     import subprocess
```



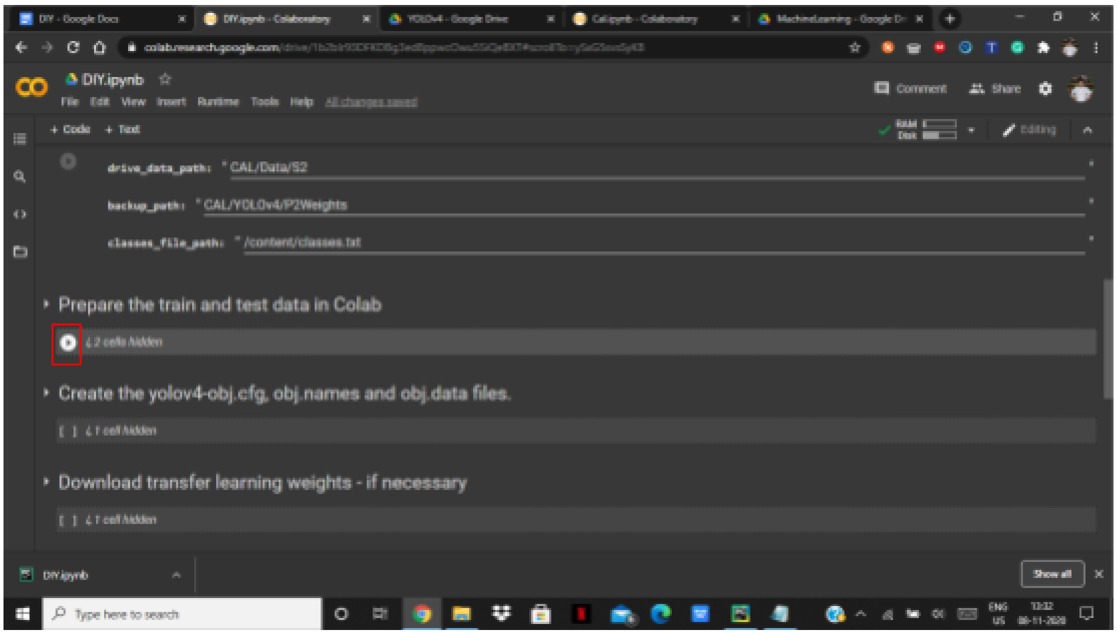
[7] Configuration Inputs

```
number_of_classes: 20
drive_data_path: "CAL/Data/S2"
backup_path: "CAL/YOLOv4/P2Weights"
classes_file_path: "/content/classes.txt"
```

Prepare the train and test data in Colab

```
[13] import os
     import subprocess
```

- Run the “Prepare the train and test data in Colab” cell.



```
drive_data_path: "CAL/Data/S2"
backup_path: "CAL/YOLOv4/P2Weights"
classes_file_path: "/content/classes.txt"
```

Prepare the train and test data in Colab

↳ 2 cells hidden

Create the yolov4-obj.cfg, obj.names and obj.data files.

↳ 1 cell hidden

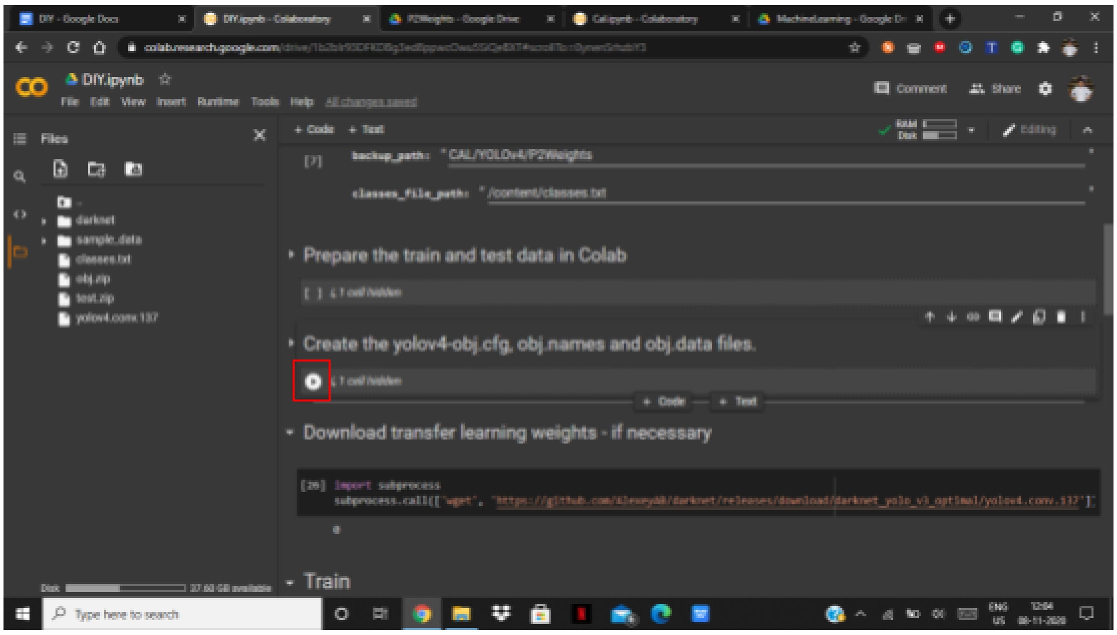
Download transfer learning weights - if necessary

↳ 1 cell hidden

Step 6: Configure Files for Training

Before training the model, we need to provide it with three files. These files should give it information such as the number of classes we are going to train it on, the names of those classes, the backup location, the location of the training and testing data and other details dependent on them. For example, the maximum number of iterations the model should train for, should be set to to the number of classes*2000. Running the below mentioned cells will take care of this for you.

- Configure the files for training by running the “Create the yolov4-obj.cfg, obj.names and obj.data files” cell.



```
backup_path: " CAL/YOLOv4/P2Weights
classes_file_path: " /content/classes.txt
```

Prepare the train and test data in Colab

Create the yolov4-obj.cfg, obj.names and obj.data files.

Download transfer learning weights - if necessary

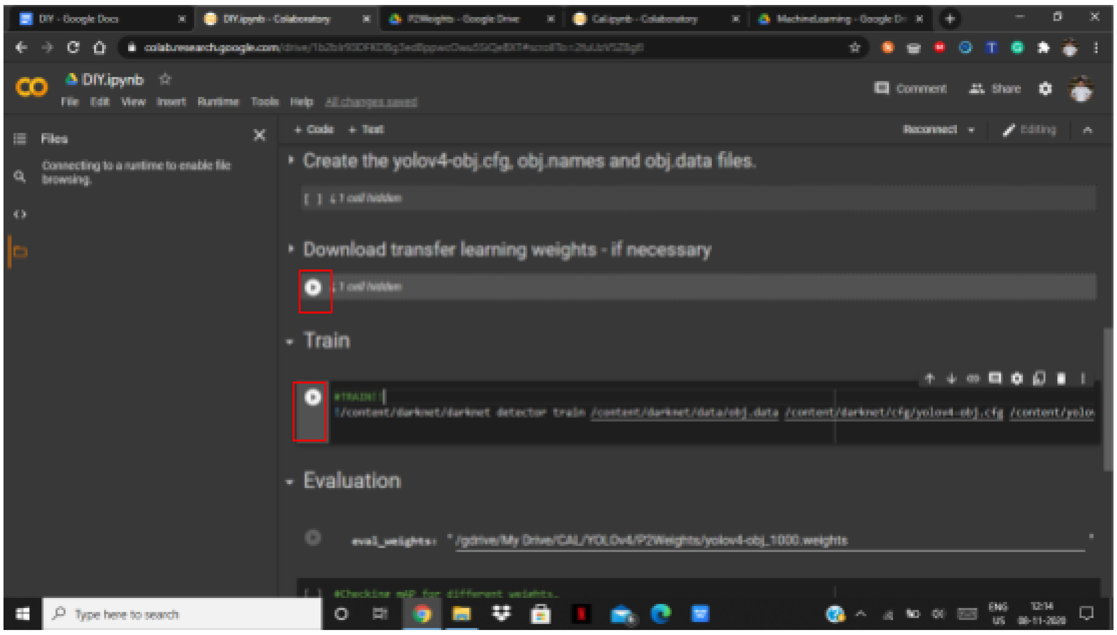
```
[20] import subprocess
     subprocess.call(['wget', 'https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137'])
0
```

Train

Step 7: Train!

- Run the next cell to download the transfer learning weights file to make training faster. This is the weights file which was obtained by training yolo on 80 classes including objects like apple, bicycle, car etc. By using this as a starting point for the training instead of starting from scratch, the model starts with some knowledge thereby helping it converge(figure out the problem) faster.

- Run the TRAIN cell to start the training.



Create the yolov4-obj.cfg, obj.names and obj.data files.

[ ] ↳ 1 cell hidden

Download transfer learning weights - if necessary

↳ 1 cell hidden

Train

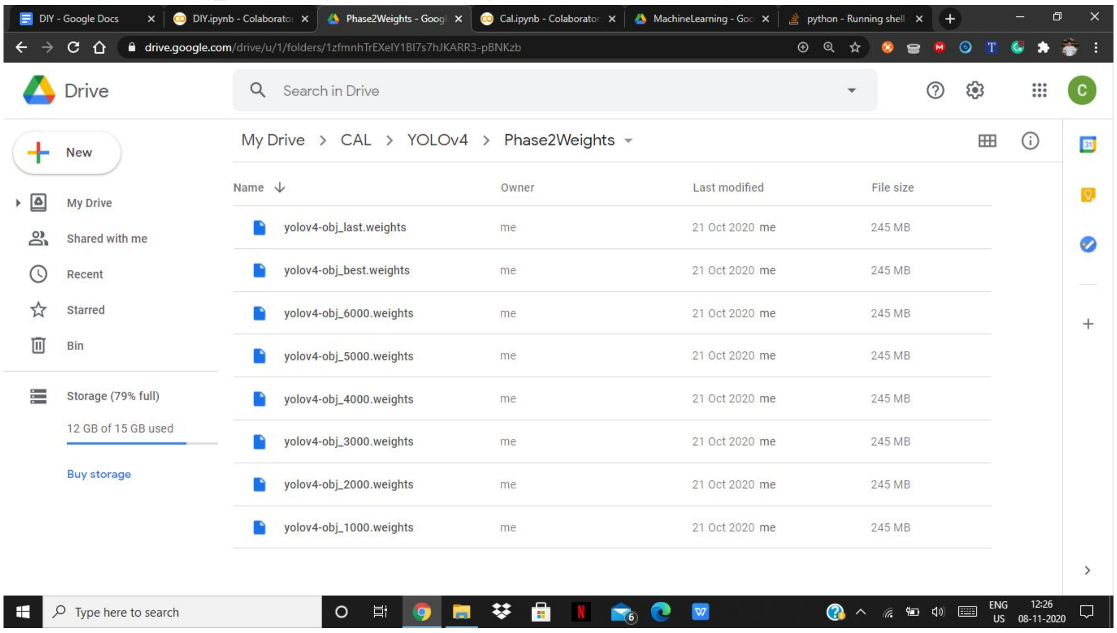
```
#TRAIN!!
!/content/darknet/darknet detector train /content/darknet/data/obj.data /content/darknet/cfg/yolov4-obj.cfg /content/yolo
```

Evaluation

eval_weights: " /gdrive/My Drive/CAL/YOLOv4/P2Weights/yolov4-obj_1000.weights

```
#Checking mAP for different weights..
```

- After training, you will find the weights saved during different stages of the training in the backup folder you specified. For example, **yolov4-obj_1000.weights** is the file saved after 1000 iterations. **yolov4-obj_last.weights** is the latest weights file and **yolov4-obj_best.weights** is the weights file with the best performance.



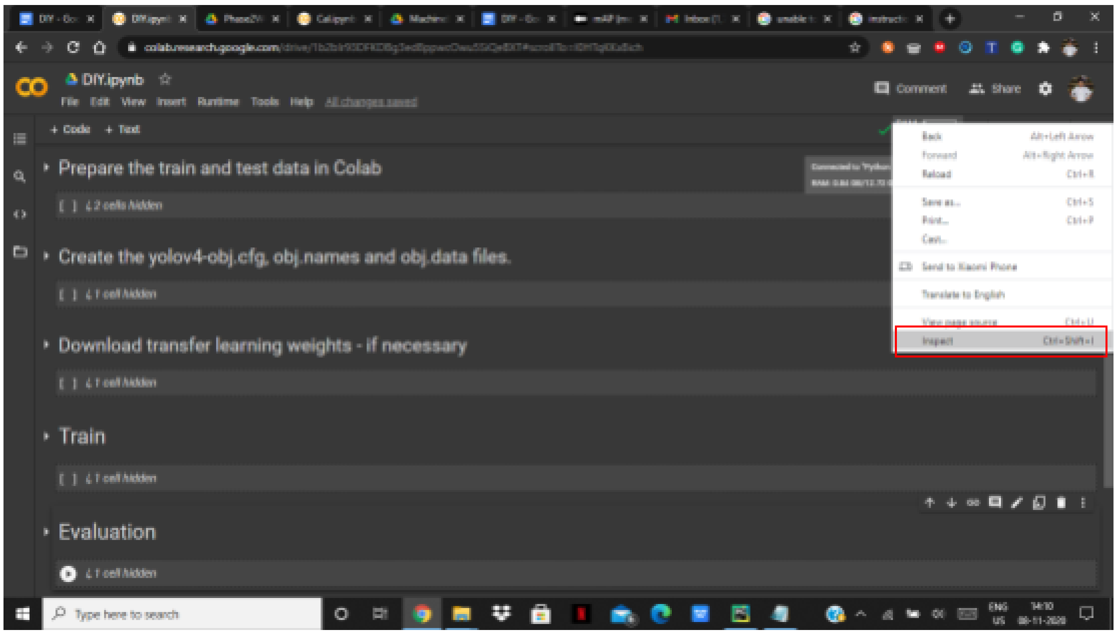
My Drive > CAL > YOLOv4 > Phase2Weights

Name	Owner	Last modified	File size
yolov4-obj_last.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_best.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_6000.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_5000.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_4000.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_3000.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_2000.weights	me	21 Oct 2020 me	245 MB
yolov4-obj_1000.weights	me	21 Oct 2020 me	245 MB

- To prevent colab from disconnecting during training, right click on the page > select the inspect option and paste the following code in the console.

```
function ClickConnect(){
console.log("Working");
document
  .querySelector('#top-toolbar > colab-connect-button')
  .shadowRoot.querySelector('#connect')
```

```
   .click()
}
setInterval(ClickConnect,60000)
```



DIY.ipynb

File Edit View Insert Runtime Tools Help All changes saved

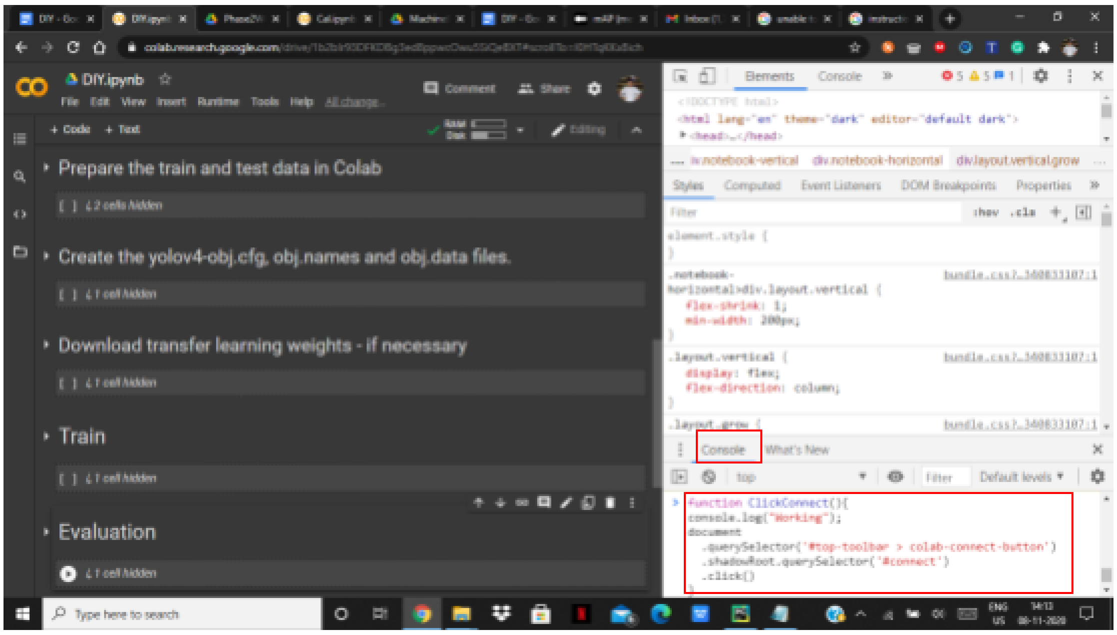
Prepare the train and test data in Colab

Create the yolov4-obj.cfg, obj.names and obj.data files.

Download transfer learning weights - if necessary

Train

Evaluation



DIY.ipynb

Prepare the train and test data in Colab

Create the yolov4-obj.cfg, obj.names and obj.data files.

Download transfer learning weights - if necessary

Train

Evaluation

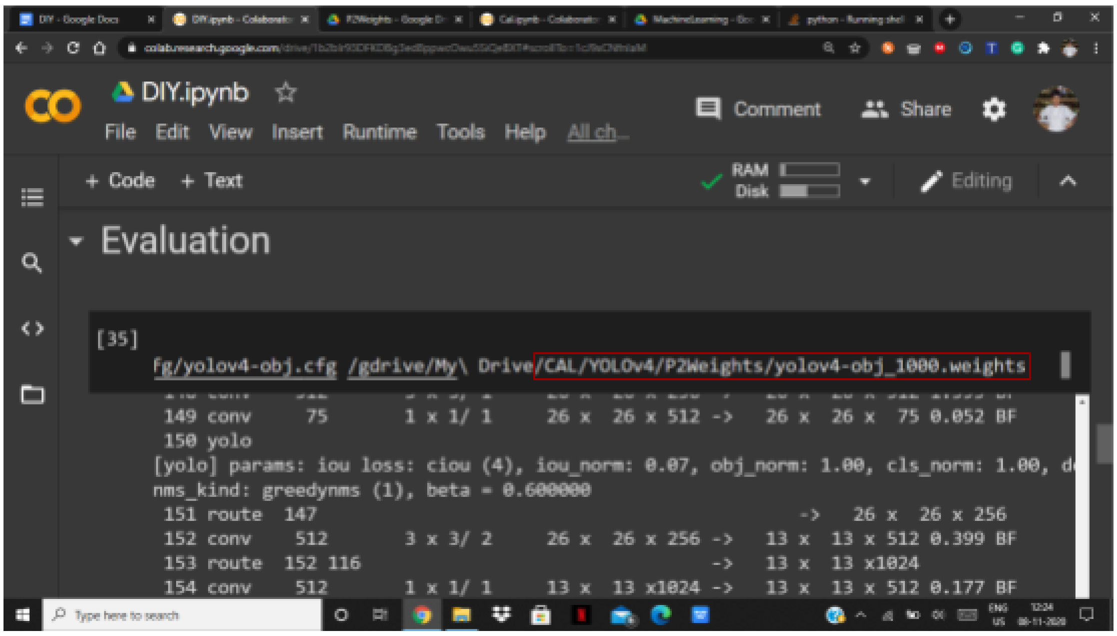
```
function ClickConnect(){
console.log("Working");
document
   .querySelector('#top-toolbar > colab-connect-button')
   .shadowRoot.querySelector('#connect')
   .click()
}
```

Doing this will automatically click the connect button on colab every 60 seconds, preventing the session from timing out during the training.

Model Evaluation

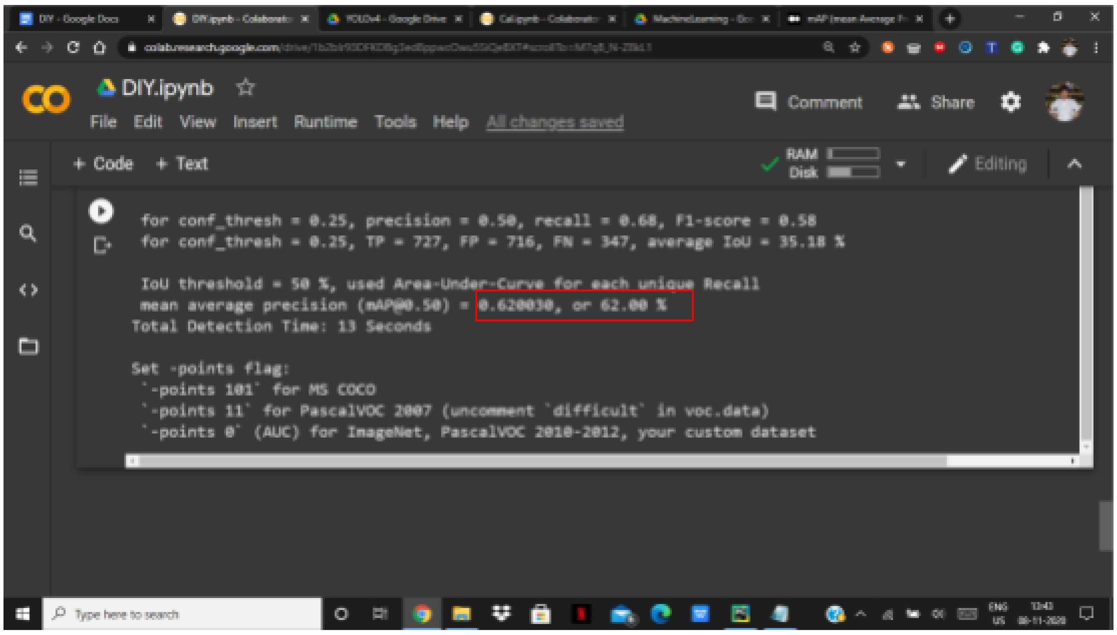
To evaluate our model, we use a metric called mean average precision (**mAP**), a metric that takes into consideration both the precision and recall of the model for all the classes it was trained on.

- After the training, evaluate the weights file by running this cell by giving the path to the weights file at the end.



```
[35]
fg/yolov4-obj.cfg /gdrive/My\ Drive/CAL/YOLOv4/P2Weights/yolov4-obj_1000.weights
 149 conv     75       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  75 0.052 BF
 150 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, d
nms_kind: greedynms (1), beta = 0.600000
 151 route  147                                    ->   26 x  26 x 256
 152 conv    512       3 x 3/ 2     26 x  26 x 256 ->   13 x  13 x 512 0.399 BF
 153 route  152 116                                ->   13 x  13 x1024
 154 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
```

- Select a weights file which gives an mAP of at least 80%, for your final model.



```
 for conf_thresh = 0.25, precision = 0.50, recall = 0.68, F1-score = 0.58
 for conf_thresh = 0.25, TP = 727, FP = 716, FN = 347, average IoU = 35.18 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.620030, or 62.00 %
Total Detection Time: 13 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

Adding More Classes

Assume that we have a model that is trained to recognize say apple and banana. Now , what if we want it to recognize orange as well? Should we start from scratch? Is there any way to make use of the training the model has gone through till now?

Consider the kid in the Big Picture example, assume that the kid while learning to recognize cats and dogs wrote down the patterns he observed to recognize cats and dogs, for example he might have written down that “these objects have 4 legs”, “these objects have 2 eyes”, “usually small in size” etc. Now, consider that you want to train him to recognize tigers and leopards. In this case the patterns he learnt in the previous task could be useful here as well as even tigers and lions have 2 eyes and 4 legs. So if you give him the paper he prepared during the first task, he can skip the basic patterns like “they have 4 legs” and work on figuring out the other patterns, and therefore learn quicker.

Similarly, in the case of the machine learning model, before starting the training of the model, we feed it the weights file (like the paper) created by the previous training process, giving it a jumpstart.

Therefore, the model is essentially applying the knowledge acquired in one task (ex recognizing apple and banana) in learning a different but related task (ex recognizing apple, banana and orange). This is known as **Transfer Learning.**

The process to do this iis more or less the same except a few changes.

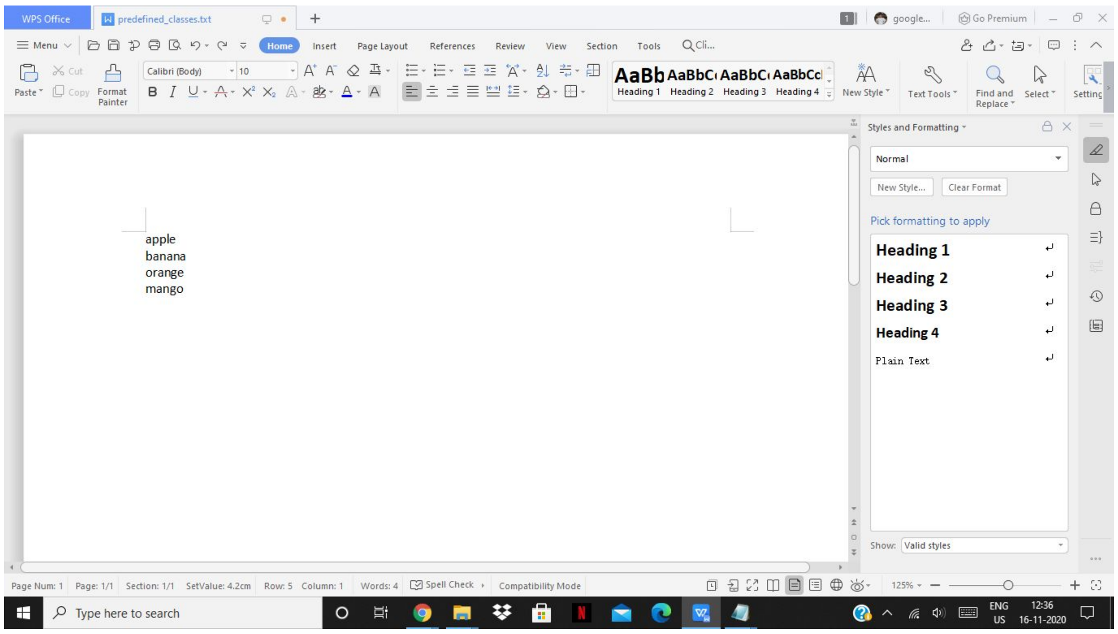
Data Collection

Step 1: Create a new folder called DataPhase2 in the Demo folder.

Step 2: Collect the images for the new classes by following Steps (3,4) in the Data Collection section previously mentioned. But this time, store the data for the new classes in the DataPhase2 folder.

Data Annotation

Step 1: Before you start annotating, edit the predefined_classes.txt to include the new class names at the end. For example, if you initially had three classes (apple, banana and orange) and now want to add mango to the list, your file should look as shown below.


apple
banana
orange
mango

Note: Do not change the previous order i.e, if your initial predefined_classes.txt file had apple first, banana first and orange third, your new file should also have them in the same order.

Step 2: Next, Continue Annotating them by following Steps (3,4) under the Data Annotation section previously mentioned.

Data Merging

The new data set must include images and annotations for both the old classes and the new classes. Therefore, we need to combine both these datasets.

Step 1: Copy the prev obj and test files to the phase 2 folder.

Step 1: Move the image and annotation files of the new classes into the obj and test files containing the previous classes data as well by following Step 1 under the Model Training section.

Model Training

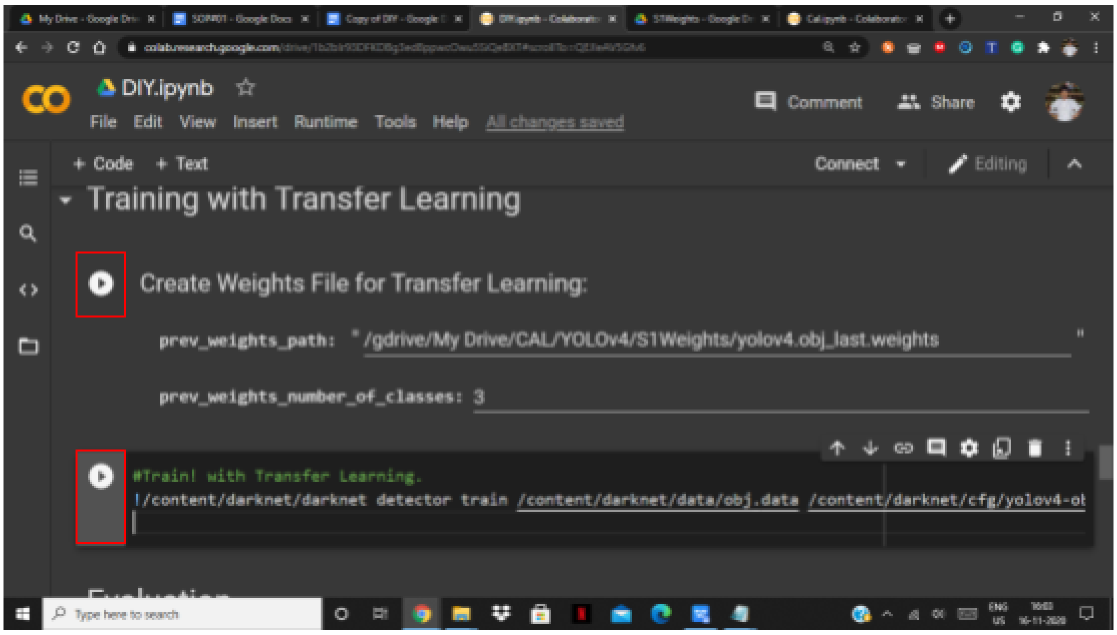
Step 1: Prepare for training by following Steps (2, 3, 4, 5, 6) under the Model Training Section previously mentioned.

Step 2: Follow Step 7 under the Model Training Section

(Or)

Prepare the weights file for transfer learning using the previously trained model. To do this give the path to the weights file and the number of classes included in the previous training in the following cell and run it.

Step 3: Run the next cell to start training using transfer learning!



Training with Transfer Learning

Create Weights File for Transfer Learning:

prev_weights_path: "/gdrive/My Drive/CAL/YOLOv4/S1Weights/yolov4.obj_last.weights"

prev_weights_number_of_classes: 3

```
#Train! with Transfer Learning.
!/content/darknet/darknet detector train /content/darknet/data/obj.data /content/darknet/cfg/yolov4-ol
```

Model Evaluation

There is no change in the manner we perform model evaluation. Follow the same instructions mentioned here.