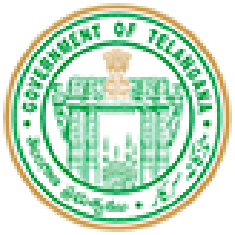


IT, Electronics &
Communications Department

Document Classifier

Contributors:

Ieshaan Saxena

K Venkat Anoop

Lakshya Kwatra

D Vasishta

T Yashwanth Reddy

Meghana Rao

Table Of Contents

User Manual: Document Classifier

1 Installation

1.0 Configuration

- Please make sure you have git and pip3 in your path.
- Also, note that this project has used python 3.6.7
- Also, the code has only been tested on 18.04 Ubuntu (Linux Distribution)

1.1 Command Line

Run the command given below in the Command Line -

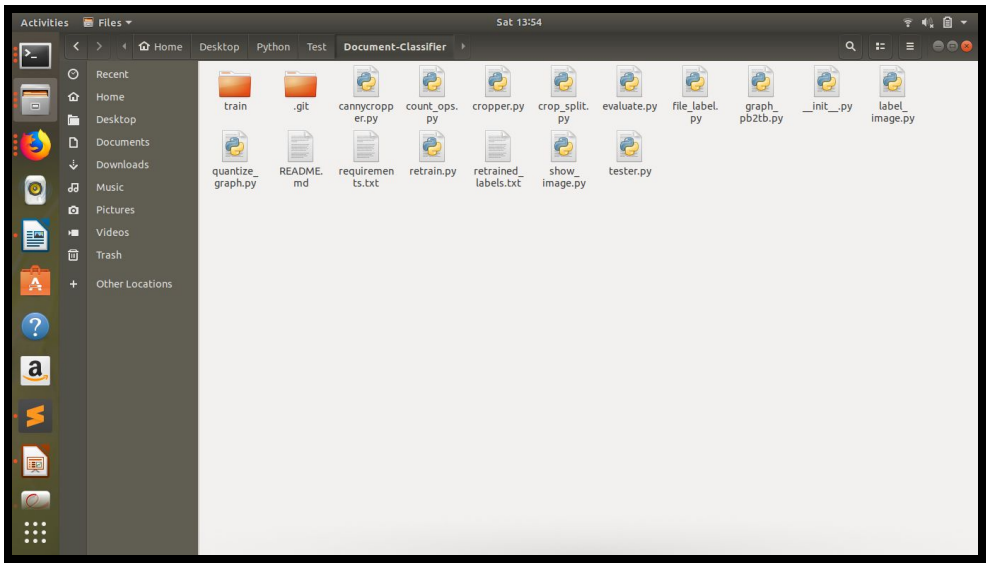
git clone https://github.com/Meghana-rao/Document-Classifier

Files that will be installed are as given below -

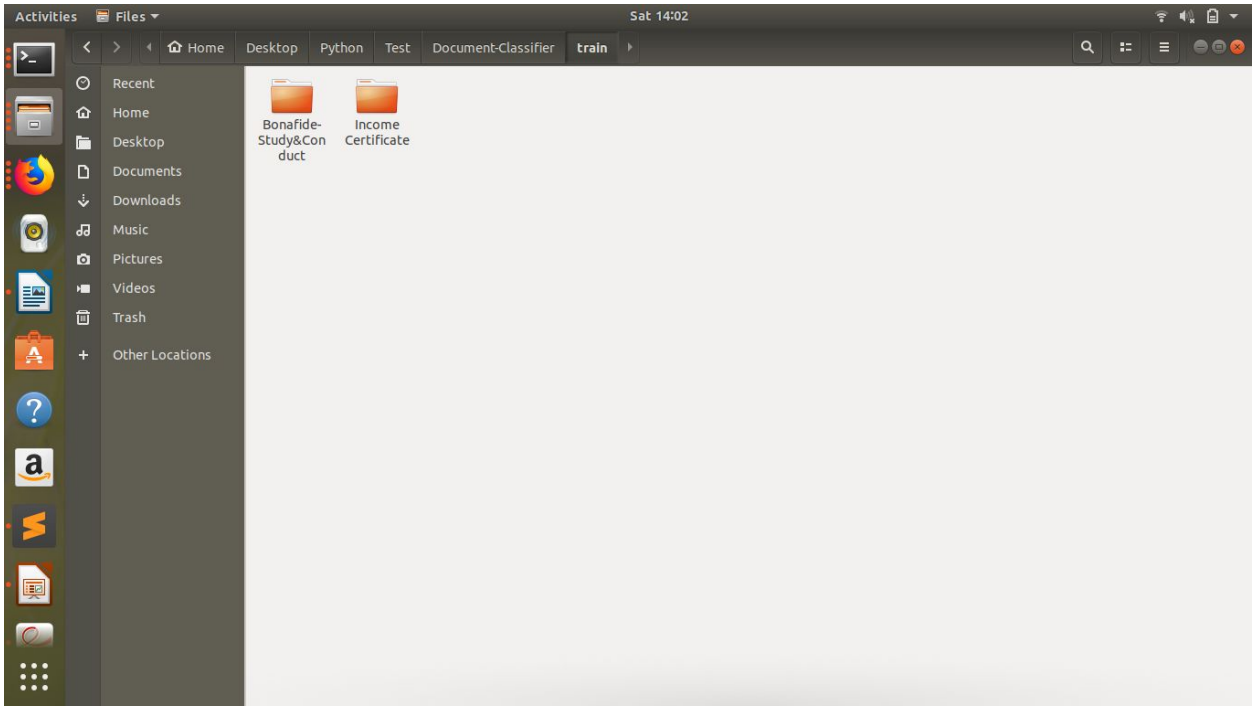
cannycropper.py. cropper.py, evaluate.py, graph_pb2tb.py, label_image.py, README.md, retrained_labels.txt, show_image.py, count_ops.py, crop_split.py, file_label.py, __init__.py, quantize_graph.py, requirements.txt, retrain.py, tester.py

1.2 Setting up training data

Create a folder named “train”.



Inside “train”, Folder, add your folderized data according to whatever name your class is supposed to be.



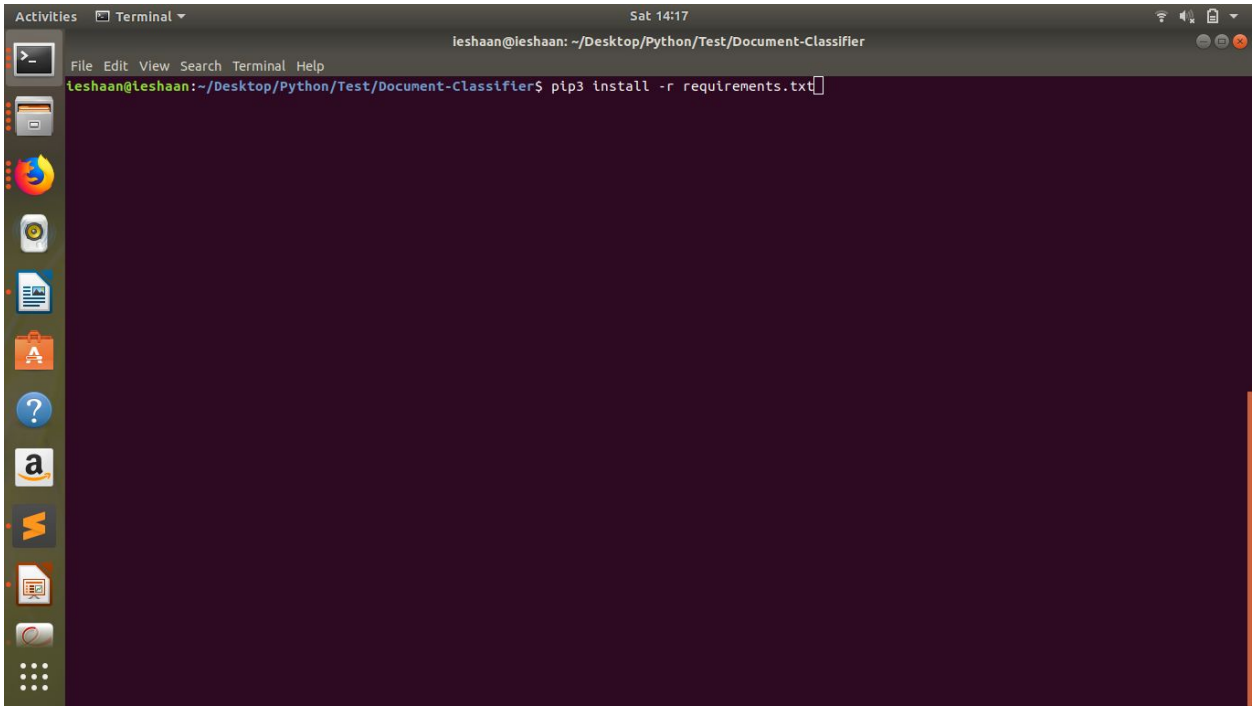
In this example, the class folders are “Income Certificate” and “Bonafide-Study&Conduct”.

These folders will contain already labelled image files according to the class.

1.3 Setting up Python Modules

Please run the following commands from the terminal inside the Document-Classifier folder:

```
pip3 install -r requirements.txt
```



This will install all the relevant modules, if you don't want to do this change in your system, please create a local environment. The link for the same is provided below - https://www.tutorialspoint.com/python/python_environment.htm

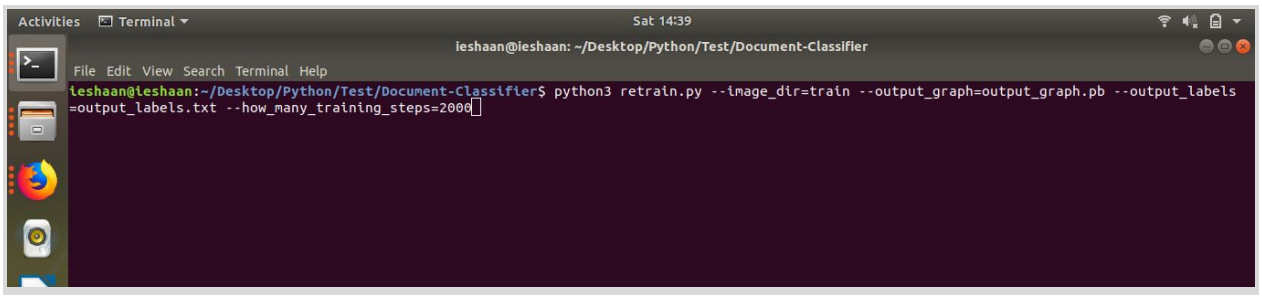
Note: Please make sure to install only these versions of modules. The program may misbehave otherwise.

The versions for each module is mentioned in the requirements.txt file.

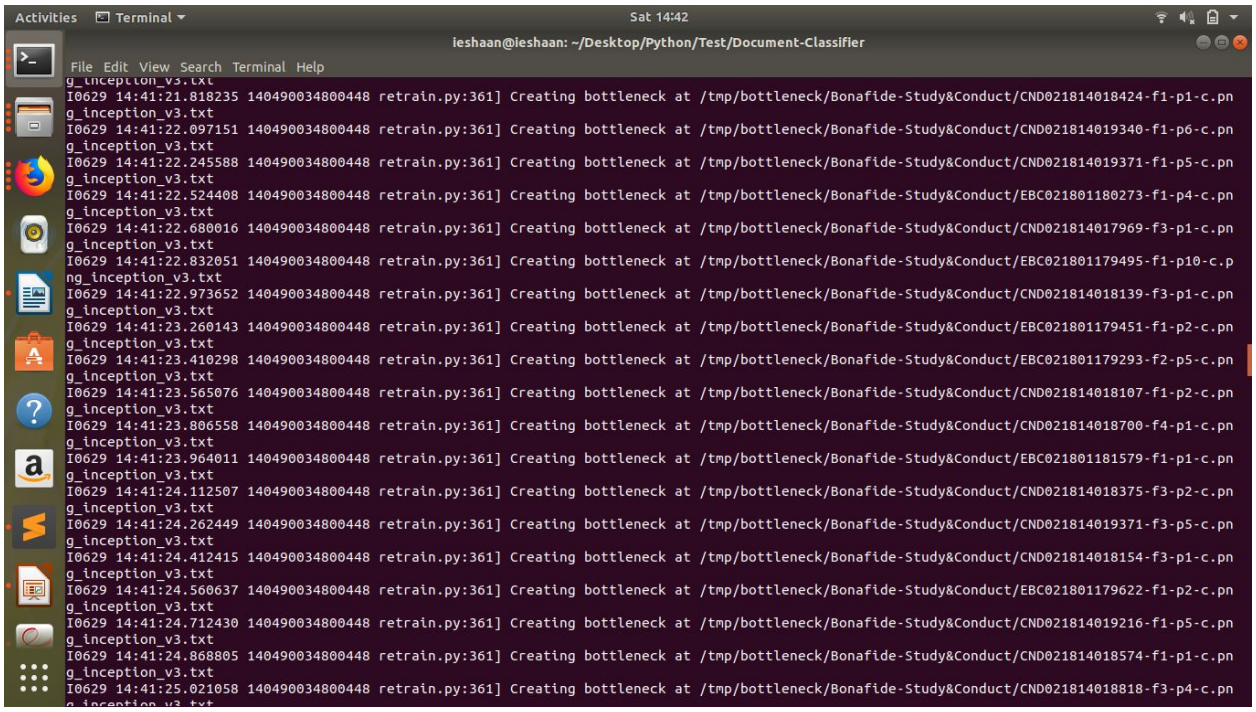
2 Making Model

2.1 Run python script retrain.py as follows

```
python3 retrain.py --image_dir=train --output_graph=output_graph.pb
--output_labels=output_labels.txt --how_many_training_steps=2000
```



After running the above command in the command line -



Inception model will start creating bottlenecks, soon the model will be trained. This can take different amounts of time depending on the training data, that is inside the train folder and the processing power of CPU.
There will be two files that will be created in your Document-Classifier folder that are-

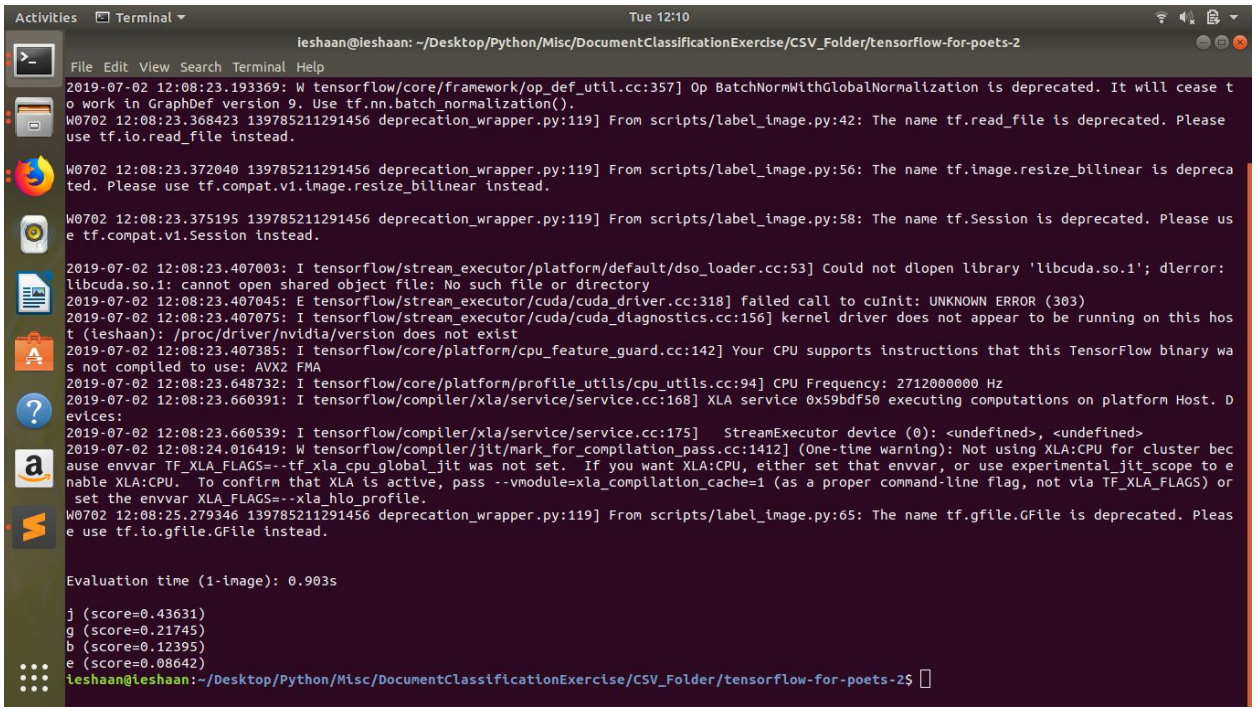
`output_graph.pb`
`output_labels.txt`

The `output_graph.pb` will help in the prediction of unlabelled data as it stores all the weights and the biases of the data. Also, the `output_labels.txt` will store the labels which will be the same as the name of your folders.

2.2 Labelling unlabelled images(single)

`python3 label_image.py --graph=output_graph.pb --labels=output_labels.txt --image=test.png`

The label_image.py file labels the test.png depending on the labels that have been created in `output_labels.txt` and the graph that is saved as `output_graph.pb`.



The script prints the evaluation time along with probable classes for different confidence levels.

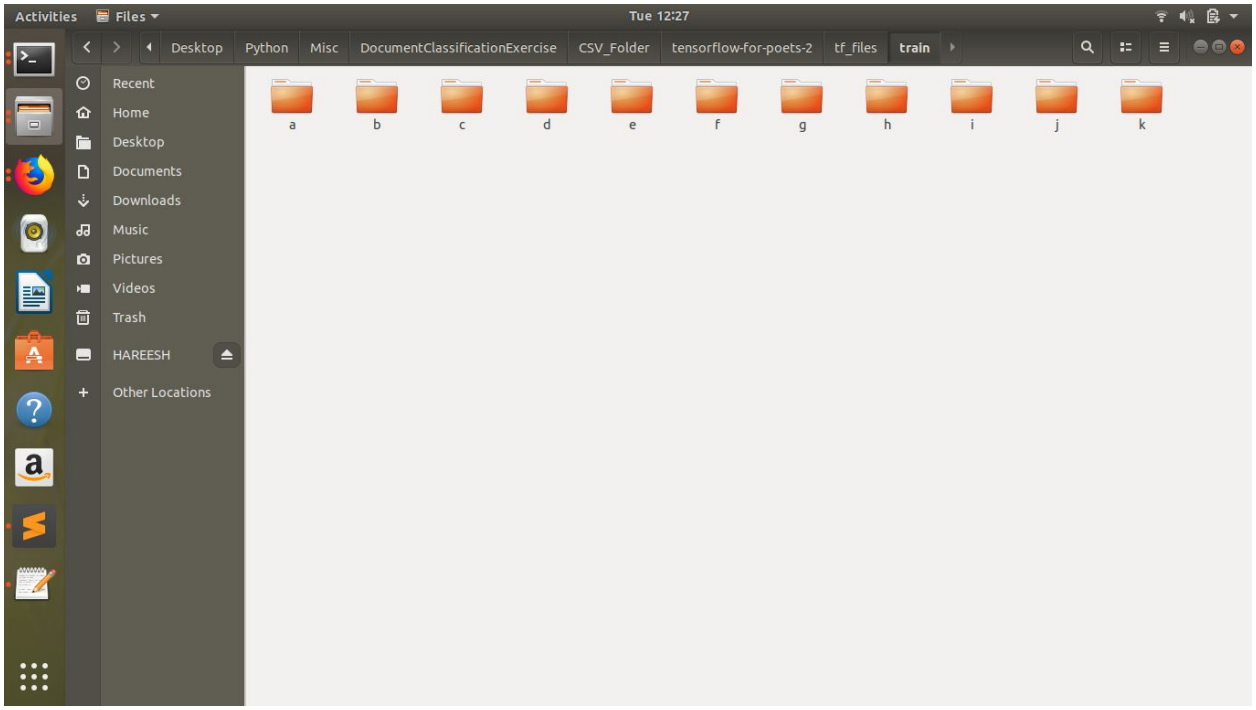
2.2 Labelling unlabelled images(all image files inside a folder)

`python3 tester.py`

For example here, the folders in our training data were as follows -

`Encoder = {`

"a":"Aadhar","b":"Community and Birth Certificate","c":"Marksheet","d":"Bonafide/Study and Conduct", "e":"Income Certificate","f":"National Food Security Card","g":"Others","h":"Caste Certificate", "i":"Blank Document","j":"EBC Certificate","k":"EBC application"}



The folders were stored based on an encoding as given above, the user can change the folders or increase the number depending on their needs for documents and classification, above we have used 11 classes. The folder names shouldn't have any spaces and be in lower case.

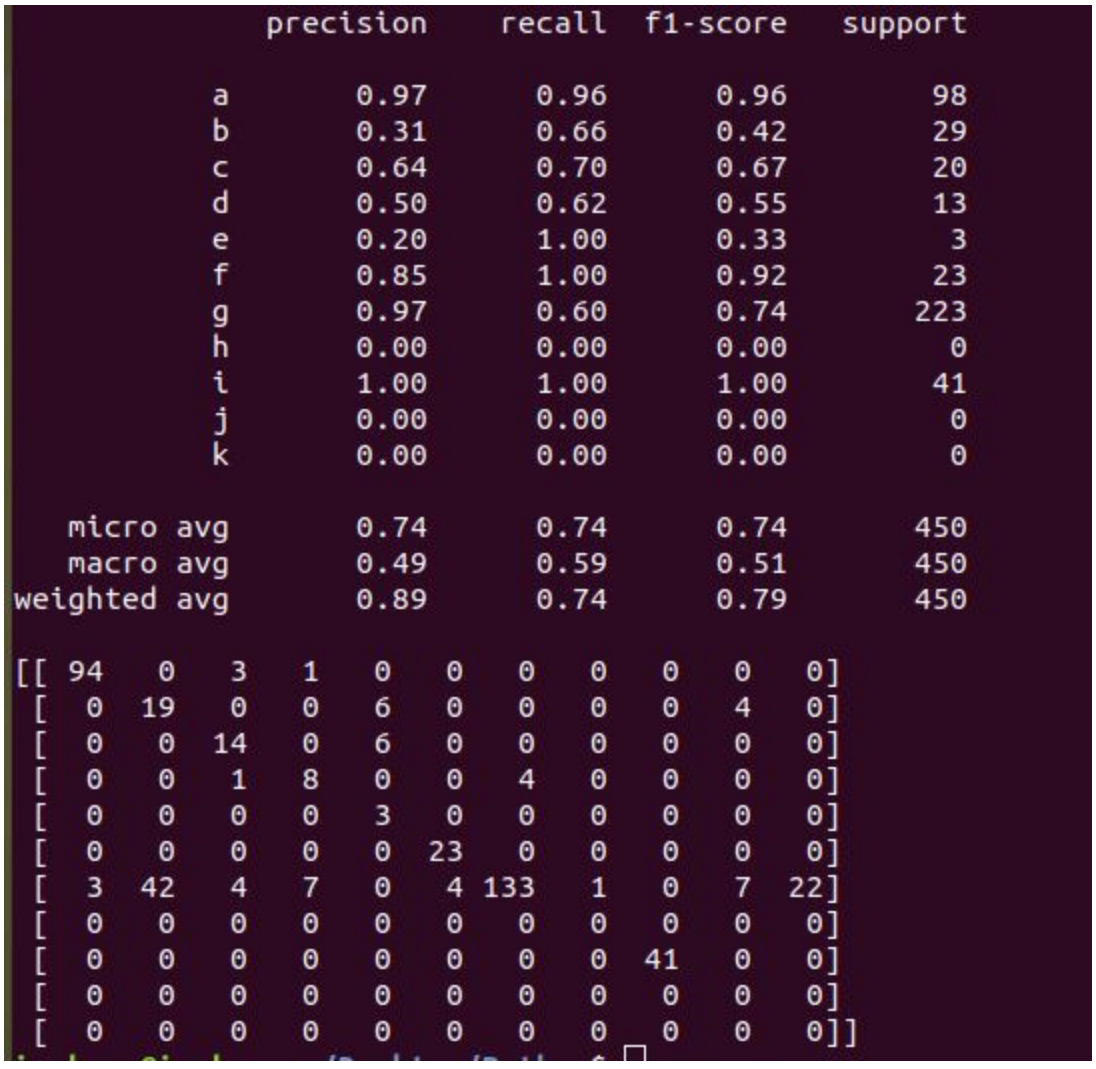
The csv_file, src_label, src_graph, folder_name need to be readjusted according to the location of the user.

csv_file - Location of test data's actual labels(data.csv)
src_label - Location of label_image.py
src_graph - Location of output_graph.pb
folder_name - Location of folder containing image files

Please note that the program will run a subprocess for each file and create a csv_file with the name pred.csv inside the folder folder_name. This will store all the predicted labels for each file.

After running the files it will display all the predictions (as predicted by the model) and the true values (as stored in the data.csv.

The program then generates a classification report and a confusion matrix based on the predictions and true values.



	precision	recall	f1-score	support
a	0.97	0.96	0.96	98
b	0.31	0.66	0.42	29
c	0.64	0.70	0.67	20
d	0.50	0.62	0.55	13
e	0.20	1.00	0.33	3
f	0.85	1.00	0.92	23
g	0.97	0.60	0.74	223
h	0.00	0.00	0.00	0
i	1.00	1.00	1.00	41
j	0.00	0.00	0.00	0
k	0.00	0.00	0.00	0
micro avg	0.74	0.74	0.74	450
macro avg	0.49	0.59	0.51	450
weighted avg	0.89	0.74	0.79	450

```
[[ 94   0   3   1   0   0   0   0   0   0   0]
 [  0  19   0   0   6   0   0   0   0   4   0]
 [  0   0  14   0   6   0   0   0   0   0   0]
 [  0   0   1   8   0   0   4   0   0   0   0]
 [  0   0   0   0   3   0   0   0   0   0   0]
 [  0   0   0   0   0  23   0   0   0   0   0]
 [  3  42   4   7   0   4 133   1   0   7  22]
 [  0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  41   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0]]
```

An example of the classification report and confusion matrix above. As it shows, the test data was 450 images. Time taken to run it on an i5 processor (7th Gen) was 20-25 minutes. The train data for the model taken above was 6000 images. The model took about 50-60 minutes to train on the same processor.

3 Setting Up Crop_split.py

3.1 Requirements

The required modules and any additional requirements have been specified in the Requirements.txt document that has been attached.

3.2 Getting the files Ready

The script requires the Files to be arranged in the following manner:

- A super folder containing all the application folders.
- Each application folder contains a variable number of documents.
- The documents may contain a variable number of pages.

Allowed configurational changes have been detailed in section 2.2.

4 Running the code

4.1 The Following imports have to be performed for the code to work

```
from PyPDF2 import PdfFileWriter, PdfFileReader
from pdf2image import convert_from_path
import os,csv
import time.
```

4.2 Allowed configurations

The following snippet from the code will keep track of all the applications being accessed, and the time taken for the script to execute

```
start_time=time.time()
TOTAL_FILES = 0
folder=[].
```

You can configure the following directories in the code:

- The super folder directory

  ```
  cwd="/home/ieshaan/Desktop/Python/Misc/DocumentClassificationExercise/income/".
  ```

  It's important that the super folder follows the structure mentioned in section 1.2

- The destination for the split documents can be configured here
  `dest="/home/ieshaan/Desktop/Python/Misc/DocumentClassificationExercise/income_sorted/"`
- The destination for the final images can be configured here
  `dest_jpg="/home/ieshaan/Desktop/Python/Misc/DocumentClassificationExercise/income_png/"`

The image count and encountered errors are tracked here

```
errors = []
png_count = 0
```

The iterable which will be used in the body of the code is initiated here

```
for i in os.listdir(cwd):
 folder.append(i)
```

4.3 Body of the code

This snippet here will access the super folder first, then the applications within the super folder, and then the documents within the application.

```
for j in range(len(folder)):
        count=0
        print(folder[j])
        for t in os.listdir(cwd+folder[j]+"/"):
        if t.endswith('.pdf') or t.endswith('.PDF'):
        count += 1
        TOTAL_FILES += 1
        print('\t',t)
        try:
                inputpdf = PdfFileReader(open(cwd+folder[j]+"/"+t,
"rb"),strict=False)
                for k in range(inputpdf.numPages):
                output = PdfFileWriter()
                output.addPage(inputpdf.getPage(k))
                f_name=str(folder[j])+"-f"+str(count)+"-p"+str(k+1)+".pdf"
                with open(dest+f_name, "wb") as outputStream:
```

```
                output.write(outputStream)
            page=convert_from_path(dest + f_name,dpi=100);
            fname=f_name[:-4]+ ".png";
            page[0].save(dest_jpg+fname,'PNG');
            png_count +=1
        except Exception as e:
            print(e)
            temp = []
            temp.append(folder[j]+"/"+t)
            temp.append(str(e))
            errors.append(temp)
```

The iterator `j` runs through the application folders, and the iterator `t` runs through the documents in the folder.

`Inputpdf` is the object pertaining to the full document, note that this may contain multiple pages. The `numPages` attribute of this object returns in integer format the page count of the current document.

Using the result from above, the following snippet gets an object representing a single page in the document.

```
    for k in range(inputpdf.numPages):
            output = PdfFileWriter()
            output.addPage(inputpdf.getPage(k))
```

Then the object is used to rename and save the extracted page in the specified destination folder.

```
    f_name=str(folder[j])+"-f"+str(count)+"-p"+str(k+1)+".pdf"
            with open(dest+f_name, "wb") as outputStream:
                output.write(outputStream)
```

The object is also used to produce an image of the current page, which will be saved in the `dest_jpg` Folder.

```
    page=convert_from_path(dest + f_name,dpi=100);
```

```
                fname=f_name[:-4]+ ".png";
                page[0].save(dest_jpg+fname,'PNG');
                png_count +=1
```

The errors are recorded and will be both outputted to the terminal and saved to a csv for future reference.

```
except Exception as e:
                print(e)
                temp = []
                temp.append(folder[j]+"/"+t)
                temp.append(str(e))
                errors.append(temp)
```

The path for the csv files can be specified here

```
Path =
"/home/ieshaan/Desktop/Python/Misc/DocumentClassificationExercise/S
tats_Errors/"
```

The errors are logged as shown.

```
with open(path+'data.csv','w') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerows(errors)
csvFile.close()
```

The stats recorded during the execution are also logged as shown

```
timer = ["Time"]
timer.append(str(time_taken))
files = ["TOTAL_FILES(PDF)"]
files.append(str(TOTAL_FILES))
no_of_png = ["PNG COUNT"]
no_of_png.append(str(png_count))
```

```
stats = []
stats.append(timer)
stats.append(files)
stats.append(no_of_png)
```

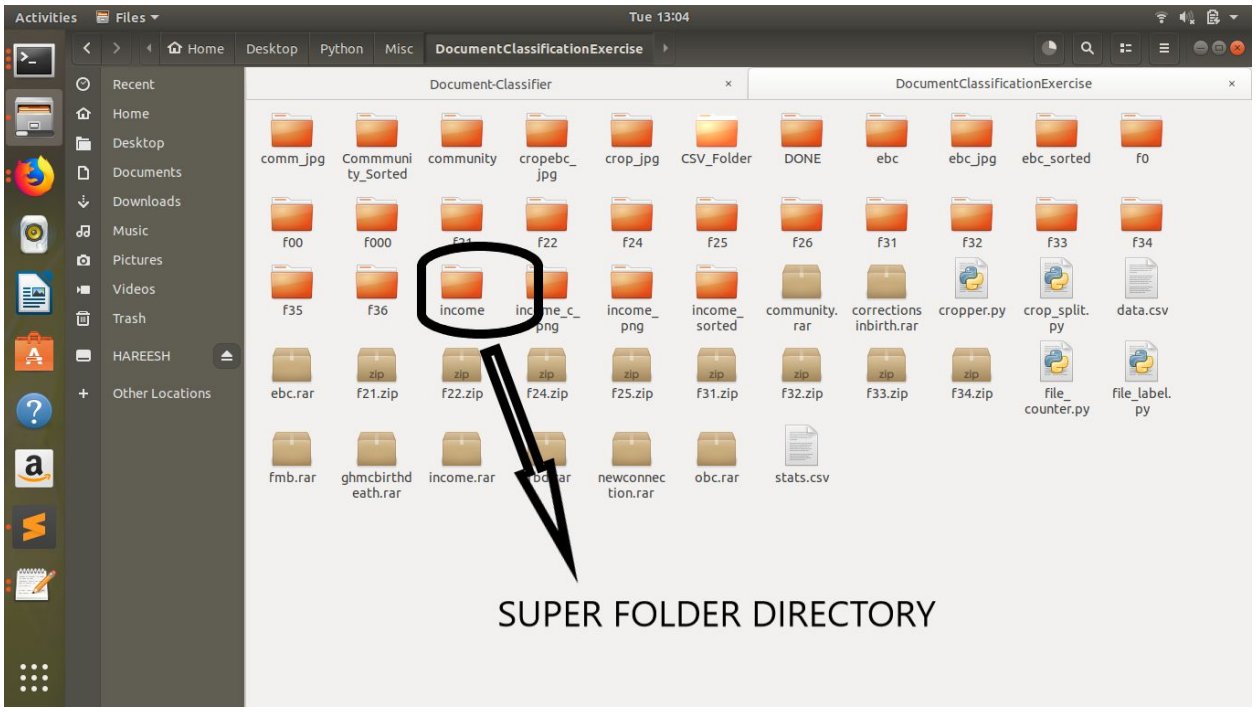
They are saved in the same location as the errors file

```
with open(path+'stats.csv','w') as csvFile:
    writer = csv.writer(csvFile)
    writer.writerows(stats)
csvFile.close()
```

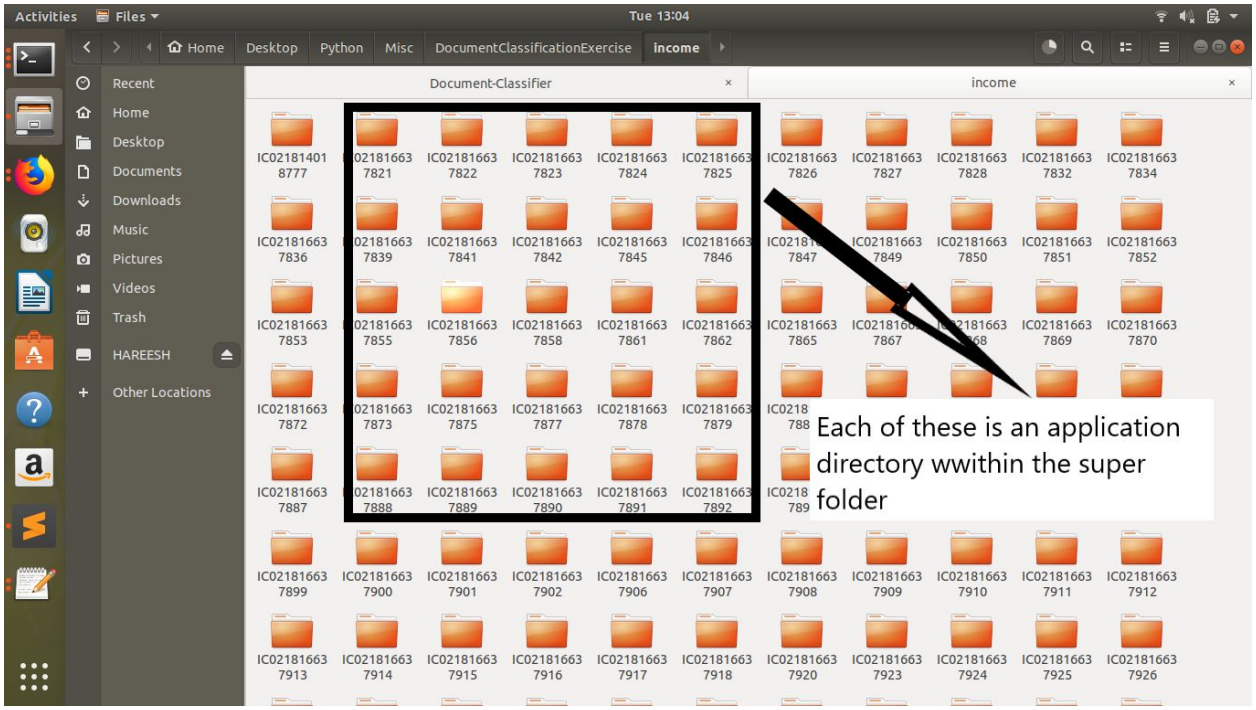
5 Code Demo

5.1 The folder structure needs to follow the example detailed here.

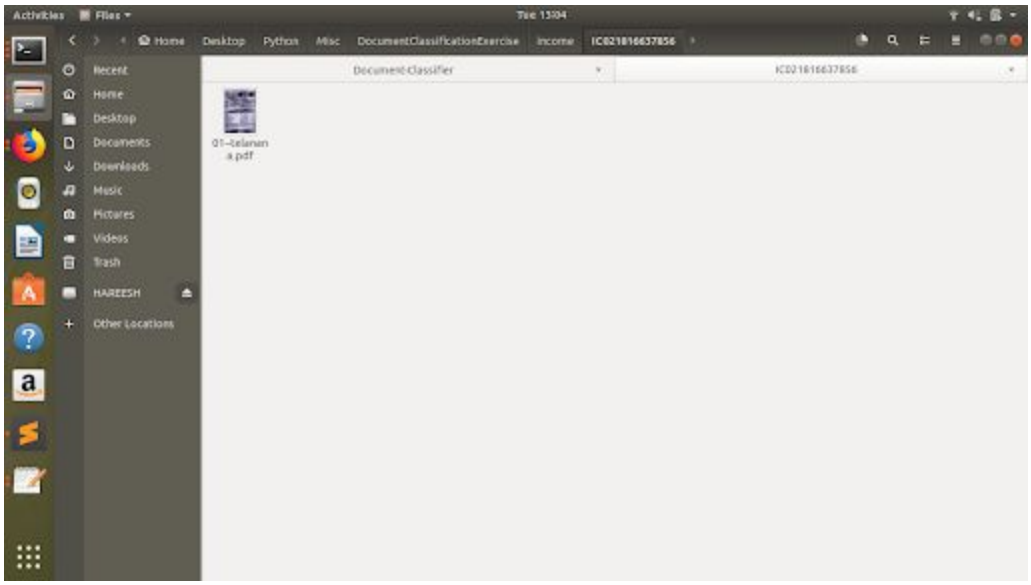
The highest level is the super folder directory



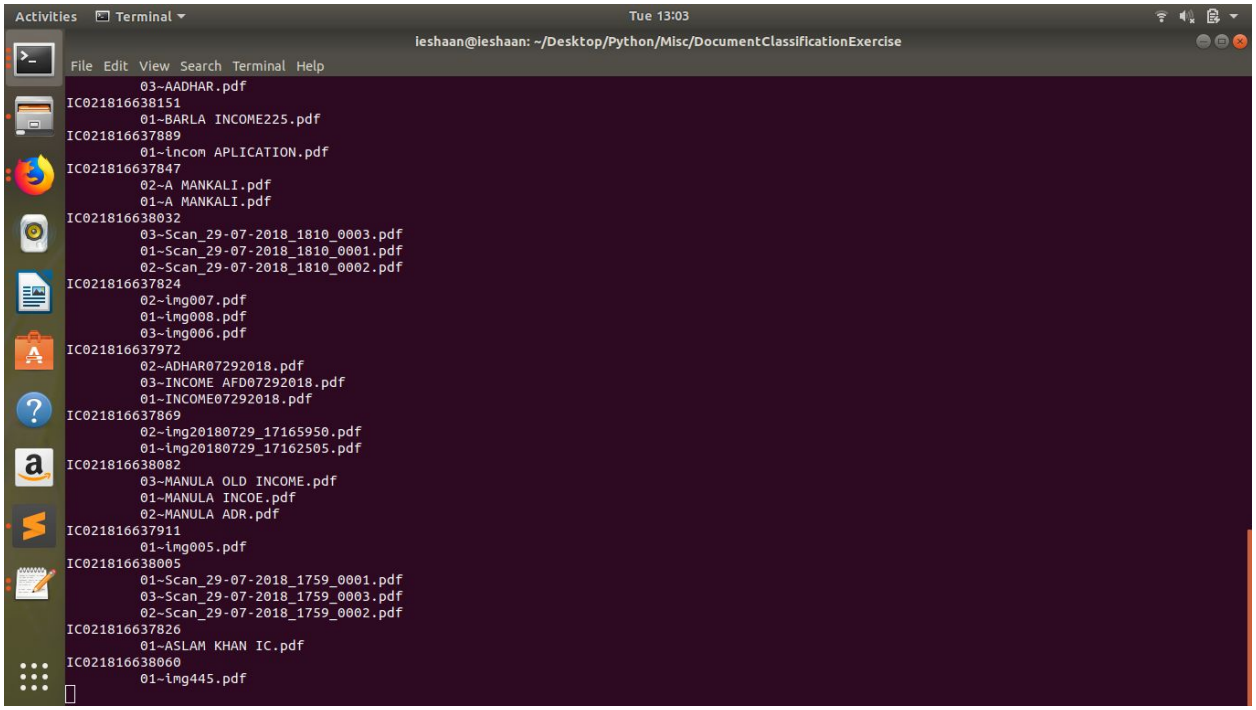
Inside the superfolder directory are the application directories



The application folders contain the documents



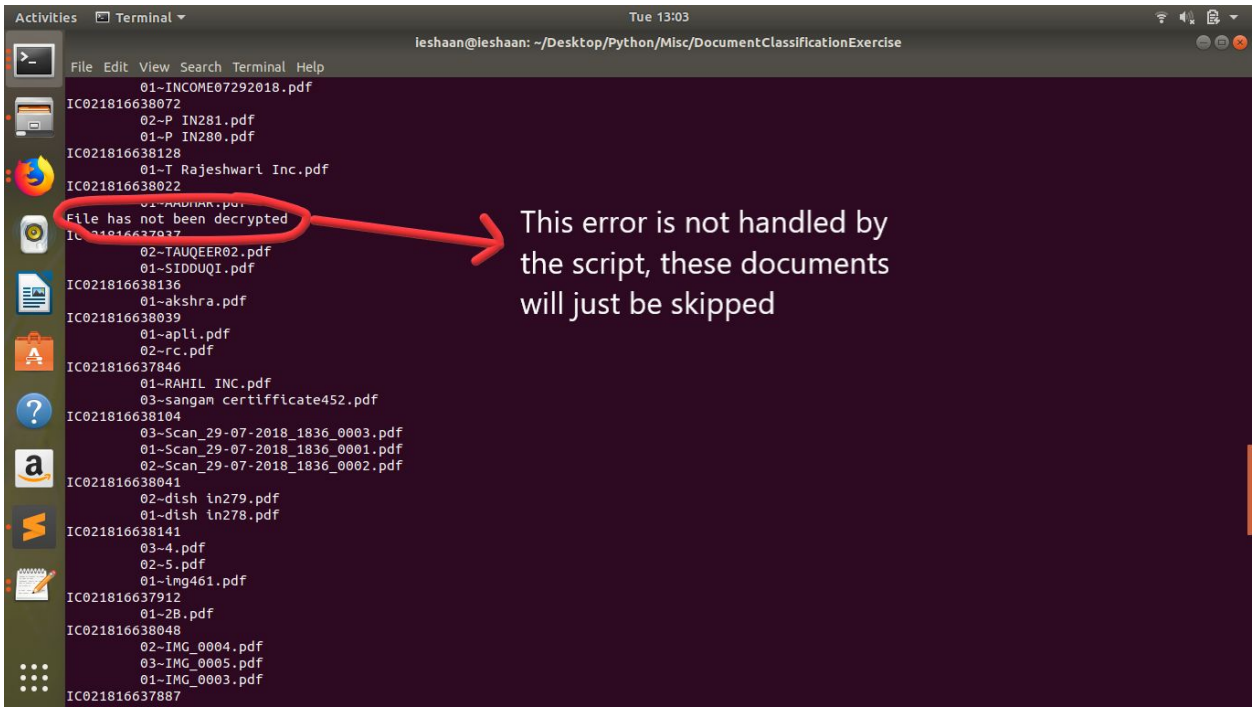
5.2 The output will closely resemble the following



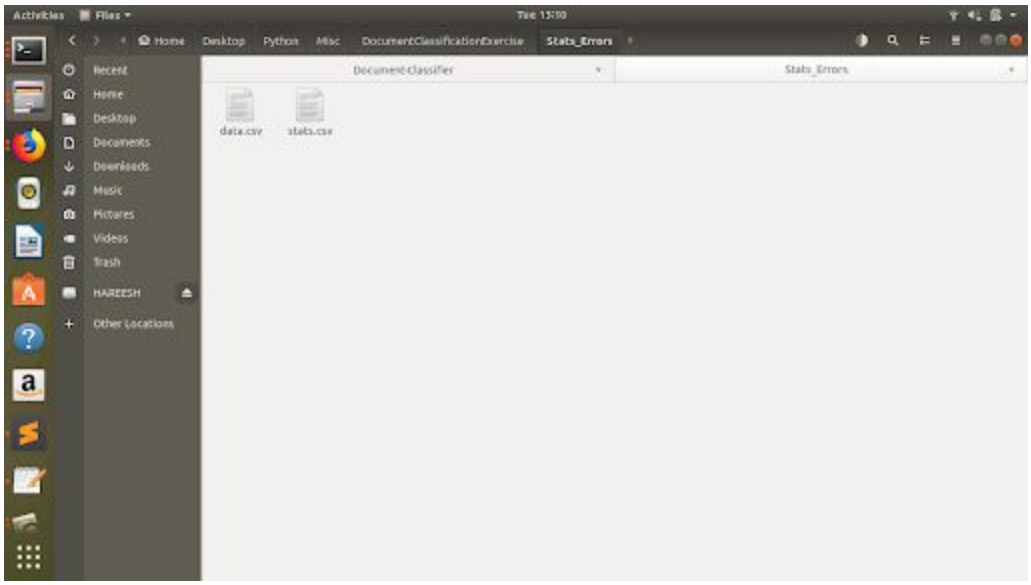
As we can infer from above, the script runs through the application folders first, and then the documents. The renamed final documents and images will contain the name of the application, the index of the file in the application folder, and the page number in that file.

6 Errors and Statistics

The most common run-in is when the file has been encrypted. These files are not handled by the code and are simply skipped.



The errors and statistics are logged in the specified `path` as csv files



7 Setting Up Cropper.py

7.1 Requirements

The required modules and any additional requirements have been specified in the requirements.txt document that has been attached.

7.2 Getting the files Ready

The script requires the files to be arranged in the following manner:

- The source file contains files in .png format which are generated after the PDFs get split by running crop_split.py.
- The cropped files will be stored in the destination folder after we run this code.
- A super folder containing all the application folders.

Allowed configurational changes have been detailed in section 2.2.

8 Running the code

8.1 Imports

The following imports have to be performed for the code to work:

```
import cv2
import os
import numpy as np
```

8.2 Allowed configurations

The following directories in the code can be configured:

- The source folder from which our program will read is `src_jpg = "/home/ieshaan/Desktop/Python/Misc/Document Classification Exercise/comm_jpg/"`
- The cropped files will be saved in the destination folder `cropped_jpg = "/home/ieshaan/Desktop/Python/Misc/Document Classification Exercise/crop_jpg/"`

It's important that the super folder follows the structure mentioned in section 1.2.

8.3 Body of the code

```
for i in os.listdir(src_jpg):
        img = cv2.imread(src_jpg + str(i))
        gray_seg = cv2.Canny(cv2.cvtColor(img,cv2.COLOR_BGR2GRAY),0,25)
        pts = np.argwhere(gray_seg>0)
        try:
                y1,x1 = pts.min(axis = 0)
                y2,x2 = pts.max(axis = 0)
                cropped = img[y1:y2, x1:x2]
                fname = cropped_jpg + i[:-4] + '-c.png'

                cv2.imwrite(fname,cropped,
[int(cv2.IMWRITE_PNG_COMPRESSION),9])

        except ValueError:
                fname = cropped_jpg + i[:-4] + '-nc.png'
                cv2.imwrite(fname,img,[int(cv2.IMWRITE_PNG_COMPRESSION),9])
```

The given snippet iterates through all the files present in the source folder whose address we have initialized as src_jpg:

```
for i in os.listdir(src_jpg):
```

This line reads the .png files present in src_jpg and stores it as a 3 dimensional numpy array in the variable img.

```
        img = cv2.imread(src_jpg + str(i))
```

This line converts the image stored in img to grayscaled 2 Dimensional numpy array containing the color value of each pixel. The array is 2-Dimensional as it contains the pixel values as rows x columns of the image. The color value in grayscale ranges from 0(Black) to 255(White) and all the shades of gray lie between this range. The function cv2.Canny() then applies canny edge detection on the grayscaled numpy array and stores the resultant 2D array in gray_seg. Now gray_seg only contains either 0(Black) or 255(White) as the pixel values because Canny Edge Detection makes “light-shaded” areas as black and “darker” areas as white. The threshold value has been experimentally optimized and can be tweaked as per the requirement. The set range for the threshold value is [0 to 25].

```
        gray_seg = cv2.Canny(cv2.cvtColor(img,cv2.COLOR_BGR2GRAY),0,25)
```

This line stores the indices where the value of the elements of the gray_seg array is greater than 0 i.e. the value of the elements is equal to 255(White).

```
pts = np.argwhere(gray_seg>0)
        try:
                y1,x1 = pts.min(axis = 0)
                y2,x2 = pts.max(axis = 0)
                cropped = img[y1:y2, x1:x2]
                fname = cropped_jpg + i[:-4] + '-c.png'
cv2.imwrite(fname,cropped,[int(cv2.IMWRITE_PNG_COMPRESSION),9])

        except ValueError:
                fname = cropped_jpg + i[:-4] + '-nc.png'
                cv2.imwrite(fname,img,[int(cv2.IMWRITE_PNG_COMPRESSION),9])
```

NOTE: ValueError, is produced for images that are homogeneously of one color in every pixel which results in no edge detection.

NOTE: Please note that Image Compression Mode 9 has been used which reduces the size of the image, but it takes time to process, Image Compression Mode can be changed to 3, if space is not a constraint.

9 Document Classifier Interface

The modules required and their version are mentioned in the requirements.txt file.

9.1 Models.py

To use the models, we need to import the models module from Django.

```
from django.db import models
```

The model 'Document' is created. Document consists of two fields `pdf` and `Application_ID`.
Each field is represented by an instance of a Field class. `Pdf` consists of a FileField which is a file-upload field and `Application_ID` consists of a CharField for character fields.

It's important to add `__str__()` methods to your models because objects' representations are used throughout Django's automatically-generated admin.Here we created a method `__str__()` and it returns the `Application_ID`

9.2 Forms.py

To use the Form class, we need to import the forms module from Django.

```
from django import forms
from up_conv.models import Document
```

We create a `PostForm` class, which inherits from Django's `forms.ModelForm` class. A subclass `Meta` is created and a variety of fields such as `Application_ID` and `Pdf` are specified.

9.3 Views.py

A view is a place where we put the "logic" of our application. It will request information from the model you created before and pass it to a template.

A dictionary `Encoder` is created which consists of all the categories(classes)and their respective codes.In future if you want to add another category,it can be mentioned here.

Adjust the `src_label` and `src_graph` file locations.

A method `post_create` is defined which takes the form from the PostForm created in the Forms.py. It checks if the form is valid,if valid it saves the instance of that file and appends the filename to the folder.

`PdfFileReader` is used to open and read the file

- For the range k (no. of classes) it counts the number of pngs (pages)

9.3.1 Cropping the image

Saving the first image with 'PNG'

A function `crop` is defined which consists of four arguments.`img`, `fname` (filename), `count` (page count), `dest`.

Converts the `img` to grayscale and gives error only when `gray_seg` >0 (i.e when the img is a blank page).

`fn` is the filename of the png along with count.

The compression mode is set to 9. This mode reduces the size of the png without data loss, but it is slow. If space is not a priority, change the compression mode to 3.

9.3.2 Applying the model

```
command = "python3 {} --graph={}
--image={}".format(src_label,src_graph,fn)
test = subprocess.Popen(command,shell =
True,stdout=subprocess.PIPE)
```

The `subprocess.Popen` takes the command, shell is set to 'True' and then `stdout=subprocess.PIPE` means that subprocess' `stdout` is redirected to a pipe that you should read

`test.communicate()` gives the output and the errors. The output is encoded using 'utf-8' and split to newlines.

The `fn` (filename) is split using '\' and appended to the `img_list`.

```
s = output[3]
# s gives the whole line
ex: (A (score = 0.3638)
```

The output is given in newlines.

```
c = s[0]
temp.append(c)
 #c is the class
```

Then `classes` is appended to temp.

```
f = s[s.find("(")+1:s.find(")")]
num = f.split('=')[1]
```

It finds from "(" to ")" and splits it based on the "=". It converts the `num` to float and checks if `(num>0.9)` and appends valid and validation required accordingly.

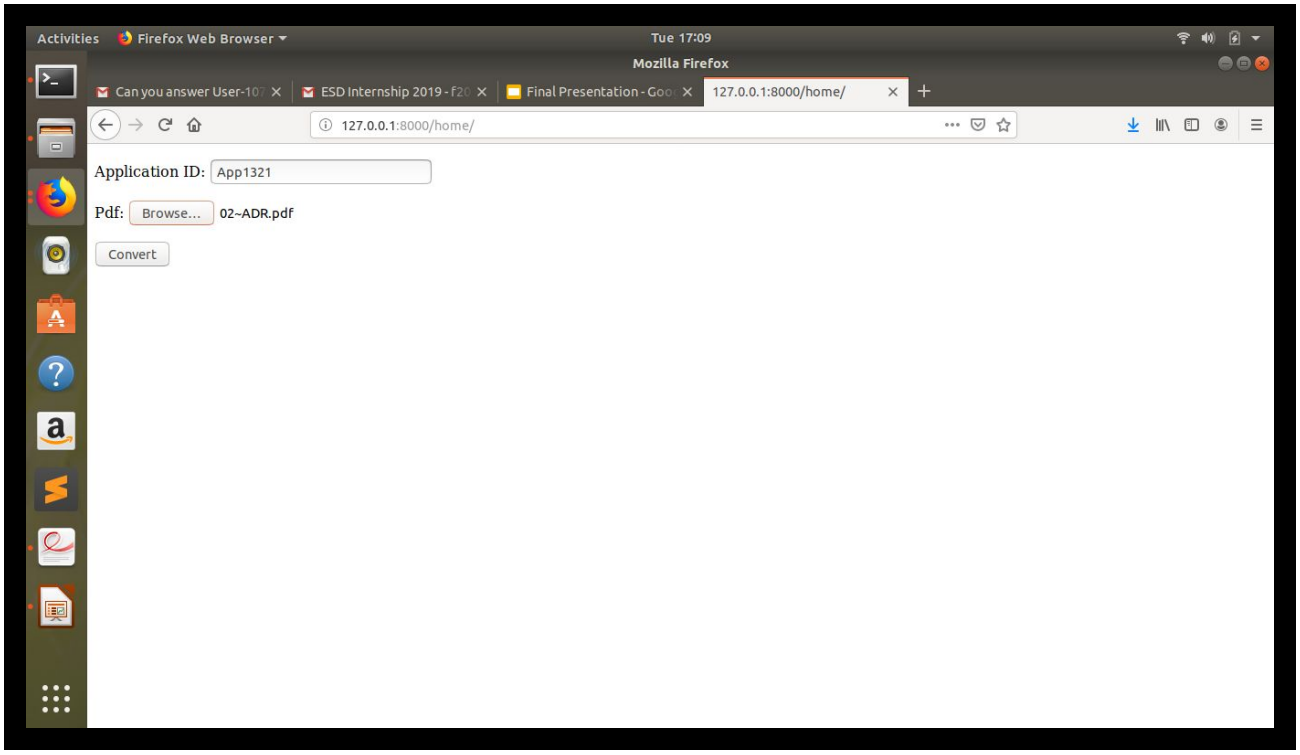
`temp` is appended to `main_csv` and `Encoder[c]` to `classes`.

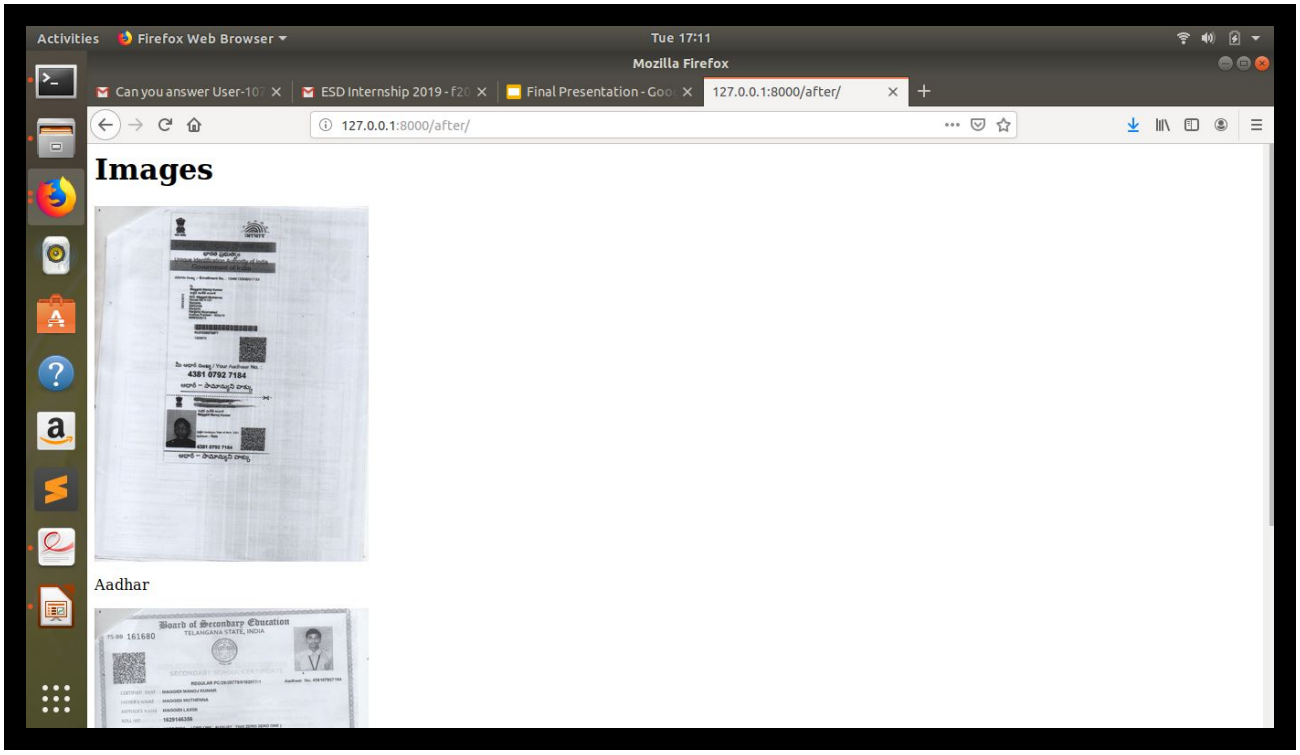
After all the pngs are stored in the `cropped_folder` unlink (remove) all the files from `pdfs_norm`, `dest_pdf`, `dest_png`.

A data.csv file is created in the cropped_folder which consists of all the png files in the following format:
Filename,class,confidence level,(valid or validation required)

A method `gallery` is defined. It consists of the path of the `cropped_folder` and a `tagged_list` which had the `image_list` followed by the `classname`. It will render it to the `gallery.html` template.

The `MEDIA_ROOT` must contain the `cropped_folder,pdfs_norm,dest_pdf,dest_png` to store the cropped pngs.




Activities
Firefox Web Browser
Tue 17:11
Mozilla Firefox
Can you answer User-107
ESD Internship 2019 - F20
Final Presentation - Goo
127.0.0.1:8000/after/
127.0.0.1:8000/after/
Images
4381 0792 7184
Aadhar
Board of Secondary Education
TELANGANA STATE, INDIA