|  |
| --- |
| D:\Users\Mohandas\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\O8ZW7CZ1\company_logo.png |
| SDD-Profield AI-v1 |
| SOFTWARE DESIGN DESCRIPTION |
|  |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Remarks |
| 1.0 | Blessy, Berlin , Amal | 03-06-2020 | Initial Draft |
| 1.0 | Blessy, Berlin , Amal | 15-07-2020 | Updated text extraction |
|  |  |  |  |

|  |
| --- |
| Software Design Document (High Level) for Profield ML |

Table of Contents

[**1. INTRODUCTION** 4](#_Toc46844095)

[1.1 PURPOSE 4](#_Toc46844096)

[1.2 SCOPE 4](#_Toc46844097)

[1.3 PRODUCT/SYSTEM OVERVIEW 4](#_Toc46844098)

[**2. FUNCTIONAL REQUIREMENTS** 5](#_Toc46844099)

[**2.1 OBJECT DETECTION** 5](#_Toc46844100)

[2.1.1 PRODUCT IMAGES 5](#_Toc46844101)

[2.1.2 IDENTIFY THE PRODUCT 5](#_Toc46844102)

[2.1.3 TRAINING MODEL 6](#_Toc46844103)

[**2.2 TEXT DETECTION** 6](#_Toc46844104)

[2.2.1 PRODUCT IMAGES 6](#_Toc46844105)

[2.2.2 IDENTIFY THE TEXT AND TABLE 6](#_Toc46844106)

[2.2.3 TRAINING MODEL 6](#_Toc46844107)

[2.2.4 TEXT EXTRACTION 6](#_Toc46844108)

[**3. DESIGN AND IMPLEMENTATION CONSTRAINTS** 7](#_Toc46844109)

[**3.1 OBJECT DETECTION** 7](#_Toc46844110)

[3.1.1 DATASET CREATION 7](#_Toc46844111)

[3.1.1.1 DATA AUGMENTATION 9](#_Toc46844112)

[3.1.1.2 CSV CREATION 11](#_Toc46844113)

[3.1.2 TRAINING 12](#_Toc46844114)

[3.1.2.1 CODE EXPLANATION 13](#_Toc46844115)

[3.1.3 TESTING 14](#_Toc46844116)

[3.1.3.1 TEST DATASET 15](#_Toc46844117)

[3.1.3.2 CODE EXPLANATION 15](#_Toc46844118)

[**3.2 TEXT EXTRACTION** 19](#_Toc46844119)

[3.2.1 DATASET CREATION 19](#_Toc46844120)

[FIG 3.2.1 FOLDER STRUCTURE WITH TEN PDFS 19](#_Toc46844121)

[3.2.1.1 DATA AUGMENTATION 20](#_Toc46844122)

[3.2.1.2 CSV CREATION 21](#_Toc46844123)

[3.2.2 TRAINING 23](#_Toc46844124)

[3.2.2.1 CODE EXPLANATION 24](#_Toc46844125)

[3.2.3 TESTING 25](#_Toc46844126)

[3.2.3.1 TEST DATASET 25](#_Toc46844127)

[3.2.3.2 CODE EXPLANATION 26](#_Toc46844128)

# **1. INTRODUCTION**

## 1.1 PURPOSE

The purpose of this document is to list a detailed requirements description of the Profield AI Application. Description includes the purpose and features of the system, the interfaces of the system, the operations in the system and the constraints under which it must operate. This document is intended for both customers and developers of the system and shall be reviewed and approved by the customer, before proceeding with development of the system.

## 1.2 SCOPE

This system will be an AI Application System for Profield Corporation in Japan. The AI system will allow users to provide the scanned images of the catalog and identify the images of the individual product and its details such as product name, description, price. The users will be able to view the details of every image and verify the items with its name and its image and the other details. If any discrepancy is identified, he/she should be able to modify the desired product name and color and then save it to the database.

## 1.3 PRODUCT/SYSTEM OVERVIEW

The overview of the image prediction system consists of two parts

1. First is the training the image detection model based on the metadata CSV file prepared by the ML developer on the existing product images.
2. Then based on the trained model, a product is identified from a new input image.

## 

FIG 1.3 SYSTEM OVERVIEW

# **2. FUNCTIONAL REQUIREMENTS**

The functionality of the AI application is to identify the products and its details in the catalog page image. This will be done in two phases. The object detection is done in the first phase and the text around the identified objects will be read and extracted in the second phase.

## **2.1 OBJECT DETECTION**

### 2.1.1 PRODUCT IMAGES

All individual images of a product and its related images from a catalog page will be given as input data to train and build the image detection model.

### 2.1.2 IDENTIFY THE PRODUCT

When a user gives a catalog page as an input to the image detection application, it should identify the objects by the Name of the product along with its color, its location in the page and its size.

### 2.1.3 TRAINING MODEL

In order to get a more accurate model from the training model, there should be at least 50 to 60 varieties of images required. If a user is unable to supply sufficient numbers of images for training, with the existing images, we will generate the required 50 to 60 from the existing images using augmentation techniques.

## **2.2 TEXT DETECTION**

### 2.2.1 PRODUCT IMAGES

All the PDFs are converted to jpg images and then given as input to build the image detection model.

### 2.2.2 IDENTIFY THE TEXT AND TABLE

When a user gives a catalog page as an input to the image detection application, it should identify the texts and tables present in the page and its location in the page.

### 2.2.3 TRAINING MODEL

In order to get a more accurate model from the training model, there should be at least 20 varieties of images required. If a user is unable to supply sufficient numbers of images for training, with the existing images, we will generate the required 20 from the existing images using augmentation techniques.

### 2.2.4 TEXT EXTRACTION

* Get the text and table around the image and store them to a list.
* The text extraction range is determined based on the upper left block in which the identified object is located to the bottom right corner of that block.
* The detected text will be further extracted from the bounding box and saved to a text file with the location.
* Along with the extracted text, the other details such as location, font type and font size may be needed. Based on the estimate for getting details such as location of text, size and font of the text this will be approved by the customer.

# **3. DESIGN AND IMPLEMENTATION CONSTRAINTS**

# **3.1 OBJECT DETECTION**

## 3.1.1 DATASET CREATION

Training data for Machine Learning is the key input algorithm that comprehends from such data and memorizes the information from for future prediction. Training data is a backbone of the entire AI and ML project without that it is not possible to train a machine that learns from humans and predicts for humans.

In the beginning, we received ten folders from the client. Each folder had a PDF and some images of chairs in it. The images correspond to that within the PDF.Some images were the part of a chair like cushion or seat. Each PDF had two pages. A PDF consisted of more than three types of chairs within which a chair had different colors and other specifications.

Sometimes an image in PDF consists of multiple chairs. Our first task was to compare the images and PDF within a folder. So we mapped the images to that within the PDF. We successfully grouped the images to different chairs and classified them on the basis of color.

In case of an image with multiple chairs, we mapped such images to all chair groups present in that image. In this way we successfully created the dataset and classified them to specific groups. We have a total of 60 types of chairs and they make a total of 164 classes. Each class had very few images in the beginning. So our dataset was very small and with this small dataset, if we do training, we will not get a good output. So we have to increase the size of the dataset.

## 

FIG 3.1.1 FOLDER STRUCTURE

The above figure shows the folder structure with PDF and images in it.



FIG 3.1.1 PDF STRUCTURE

The fig 2 shows the pdf structure inside a folder. One pdf consists of two pages as shown in figure. The images inside the folder get mapped to the image in pdf as shown in figure.

### 3.1.1.1 DATA AUGMENTATION

When there is only limited data available for training we can use Data Augmentation techniques through which we can generate additional images based on the original image. From these data collection we may need to carefully select a right mixture of sample data for building the model. This technique also helps in a simple way to prevent the model from over fitting.

Techniques include:

1. Flip  
   Flip images horizontally and vertically. In our scenario, we might use only horizontal flipping.
2. Scaling  
   Image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size, the Zoom In effect is produced.
3. Crop  
   Just randomly sample a section in the boundary based on the center, from the original image. We then resize this section to the original image size. This method is popularly known as random cropping
4. Translation/Shifting  
   Translation is moving the image along the X or Y direction (or both).
5. Noise   
   Adding the right amount of noise can enhance the prediction rate. The right amount of noise will help the model to detect objects, even if there is not enough quality as we provided in the training data.

6. Blurring  
Add a random variance of blurring of the image, which will help to increase the accuracy parameter of the model we are building.

Example:

The below are the samples generated from the original single Image. Each image is generated by one more technique in random with the parameter value variation randomly applied.

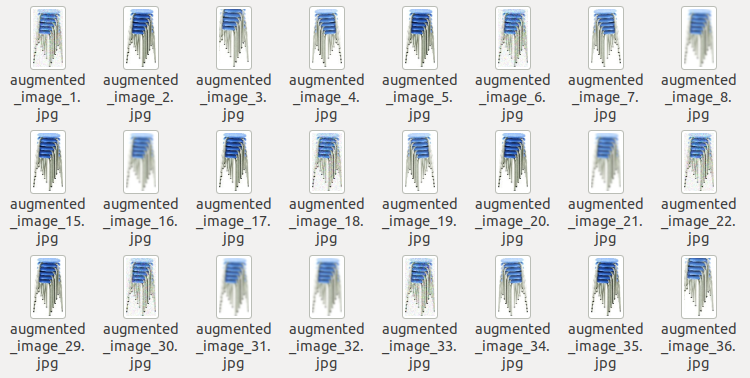


FIG 3.1.1.1 AUGMENTED IMAGES

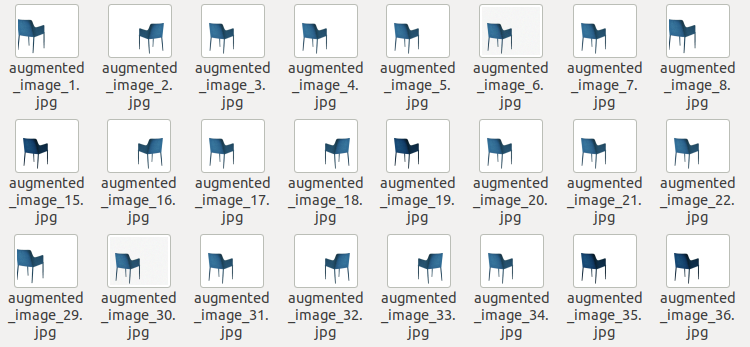


FIG 3.1.1.1 AUGMENTED IMAGES

After data augmentation, we got fifty samples of each class which made the images more than 7500. In addition to the fifty samples, we also added the multiple chair images to it. Finally, the dataset had more than 8300 images. Now the dataset size is big enough to do training. Before starting training, we have to plot the x-y coordinates of the images manually and then use this plot to create the csv file.

### 3.1.1.2 CSV CREATION

After creating the dataset, we manually plotted the x-y coordinates of each image using a tool. In this way we created the text files of each sample(166 text files). We cannot do training with the text file. So we created comma seperated files(csv files) from text files.

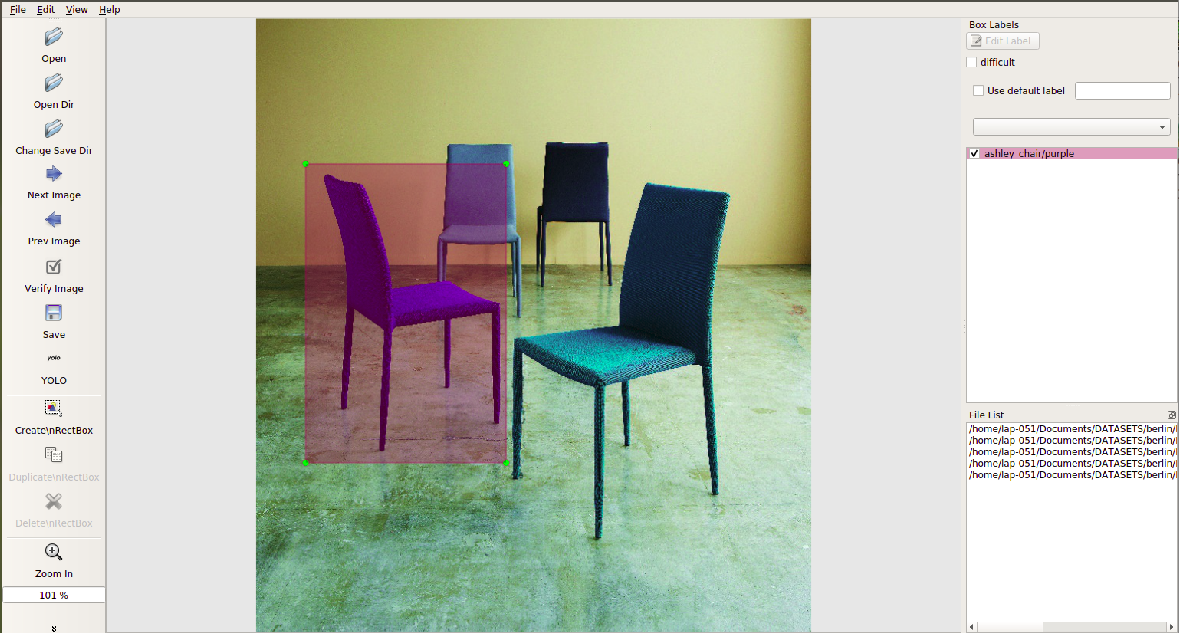


FIG 3.1.1.2 COORDINATES PLOTTING OF ASHLEY CHAIR

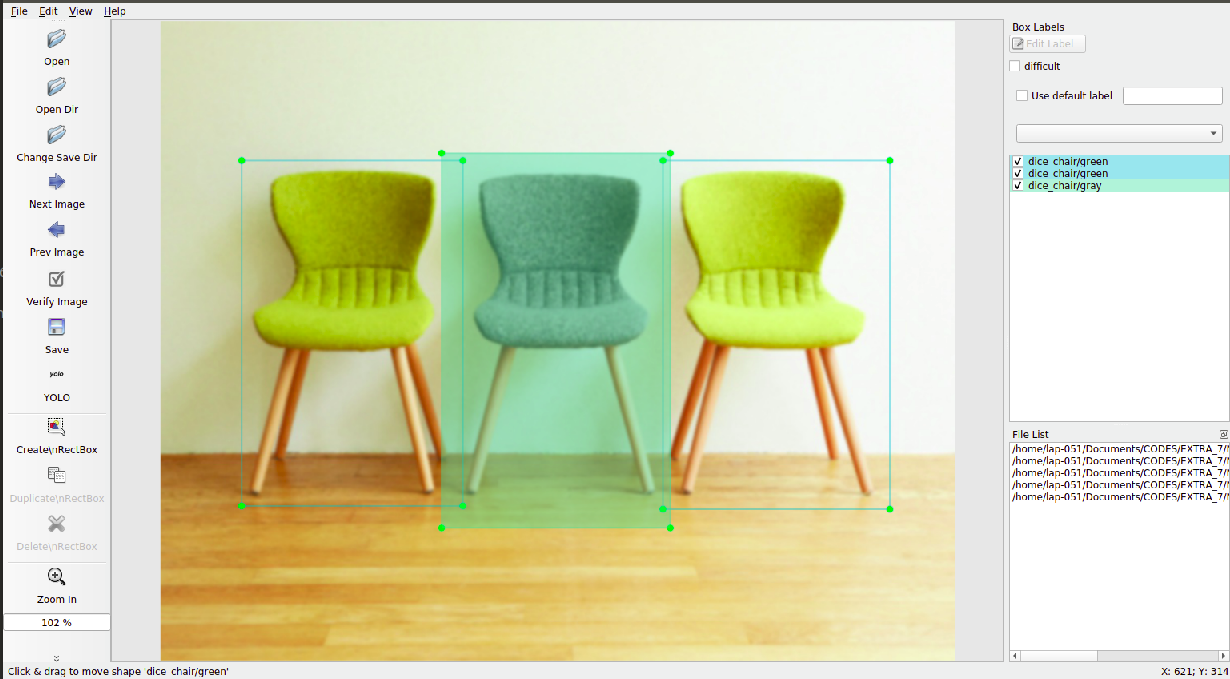


FIG 3.1.1.2 COORDINATES PLOTTING OF DICE CHAIR

The above two figures show how the coordinates are plotted using the software tool.

## 3.1.2 TRAINING

It's not possible to do the training in Jupyter Notebook as we are dealing with a huge dataset, we need enough memory. So we chose google colab as the running space. Upload the dataset, csv file and all other necessary input files to the drive and then run the python code. The libraries and its versions are mentioned below:

1. Keras - 2.1.5
2. Tensorflow - 1.6.0

After importing the following versions of keras and Tensorflow, we use a code to convert the darknet yolo model to keras. The python code is saved as ‘convert.py’. To run the code, we imported the following libraries:

* Numpy
* Keras.backend
* Keras.layers
* Keras.models
* Keras.optimizers
* Keras.callbacks

### 3.1.2.1 CODE EXPLANATION

**train.py** -The training process occurs in this file. After importing the necessary libraries, open the inputs (annotations.txt, yolo\_anchors\_path.txt and classes.txt) using the path.

**annotations.txt** - The csv file of dataset

**classes.txt** - The list of 168 classes

**yolo\_anchors.txt** - We have this file inside ‘model\_data’ in which there are some predefined dimensions. When we do prediction, the objects are detected with a bounding box, but before that the bounding box gets converted to some dimension within the yolo\_anchors.txt file and then gets displayed. So the real bounding box that we are seeing is not the actual one but the converted one from yolo\_anchors.txt.

The input shape is regularized as (416,416) which is defined by ‘input\_shape’. Model means the yolo model which gets created during training. Moreover, a checkpoint is used to save the checkpoints after every three epochs. Reduce learning is used when model learning does not improve and thereby learning rate get reduced. Again it tries to improve little by little and if it's completely not possible, the training gets ‘early stopped’. The two variables ‘num\_validation’ and ‘num\_train’ are used to split the training data in such a way that 80% for training and 20% for validation.

Initially, the first few layers are given as frozen to maintain a stable loss, because first layers have the general features and deep learning occurs in the last layers. An augmentation

process repeats within the code so that it creates more samples of images required for training. Some parameters which we can change during the training are:

EPOCH - It means the number of iterations of training which we can change accordingly. The more the number, the better will be accuracy.

BATCH SIZE - It decides how many samples should be taken in one iteration for training.

After that, we save a weight and using this weight we unfreeze the frozen layers for fine-tuning. For fine-tuning, we do the second training. In this way, the training proceeds and trainied\_weight gets saved in the location provided in drive. When training gets completed, take the ‘trained\_weight\_final.h5’ which is the final trained\_weight for testing.

The important files used here are as follows:

**model.py** -The entire architecture of yolov3 is written in model.py. It is initially defined with some darknet layers. The changes in output is made here, which means that we can change output to a required format through this file. Deeper inside, it processes the convolution layers.

**convert.py** - Python Code to convert the darknet yolo model to keras.

**yolov3.cfg** - The configurations (settings) of full code is done here.

**yolo\_anchors.txt -**This is a text file consisting of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in your training datasets. After doing some clustering studies on ground truth labels, it turns out that most bounding boxes have certain height-width ratios. So instead of directly predicting a bounding box, YOLOv3 predicts off-sets from a predetermined set of boxes with particular height-width ratios. Those predetermined sets of boxes are the anchor boxes.

**classes.txt -**This file contains all the chair names.

**yolo.h5 -**It consists of the yolo weights.

**logs/000** -A folder used to save the checkpoints and the final trained weight.

## 3.1.3 TESTING

The training data is used to make sure the machine recognizes patterns in the data, the cross-validation data is used to ensure better accuracy and efficiency of the algorithm used to train the machine, and the test data is used to see how well the machine can predict new answers based on its training. In machine learning we usually split our data into two subsets: training data and testing data.

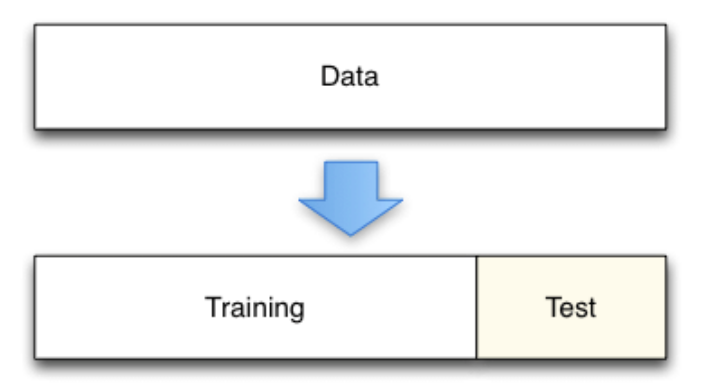


FIG 3.1.3 DATASET SPLITTING

### 3.1.3.1 TEST DATASET

When 80% of the dataset is used for training, the rest 20% is used for testing.

### 3.1.3.2 CODE EXPLANATION

After completing training, we get three parameters:

* Trained\_weight
* Class
* Image

The important files used in testing are given below:

**yolo\_image.py** - This file acts as an interface. It reads the input file given to it and passes it to utils.py and yolo.py for processing. After processing, the output file gets saved to a particular folder from this file.

**yolo.py -**The actual testing(image detection) occurs here.The trained output is read through this file. The functions for image slicing, contouring are defined in utils.py but called in this file. During prediction, to differentiate one class from another, each class will be predicted in different colors. This function is also defined here.

**utils.py** - The image slicing techniques are defined in specific ratios to detect image. Moreover, contouring of the image is also defined here. The extra functions required for image prediction are included in this file. Correct bounding box are drawn with the function defined in it and moreover it discards invalid boxes.

First, the image gets passed to the loop and if it can open the image, it is passed to the function ‘yolo.detect\_image(image)’. The default model\_image size taken is (416,416). The image is passed to the function ‘crop\_image’. In this function, we do image slicing. Image is sliced to 12 parts and then the top x-y and bottom x-y are calculated and form a list just like an array and then this function is returned.

Image is converted to array since processing in opencv can be done only if the image is in the form of an array. Then the image is converted to gray-scale and then further to binary-scale. On the basis of contours, a bounding box is created and we store the values ‘top x-y’ and ‘bottom x-y’. Now we have two lists(array) one in function ‘cropped\_box’ and another in function ‘bound\_box’. So, all the dimensions are combined together in function ‘combined\_images’.

To get a better accuracy in output, opencv is better than pillow and therefore the functionalities we process in pillow should be done in opencv. Considering this, we do image processing in the form of an array. Within ‘combined\_images’, we can see so many dimensions which after processing, converts back to a pillow.

In the function, ‘letter\_box’ the image size is converted to an optimum size accordingly. The parameters image\_width, image\_height, model\_width, model\_height are calculated. These parameters are used to scale the image and resize the image.

The output parameters we acquire are:

* Output box
* Score
* Class

During testing, according to the size of the image we gave as input, thickness is decided and processed. Then during processing we have so many output classes and processing occurs one by one. Image position is calculated, rectangular box is drawn, label is decided for every image and all the processes continue for every class. To control repeated prediction over the same class, noise is added in sections and output image is processed. After processing the loops for drawing rectangle, writing the class name and all, it gets mapped to the new image. Options are given to choose the color of the rectangular box and thickness of text.

An example of input image given for testing is shown below:



FIG 3.1.3 INPUT IMAGE

Output image is shown below:



FIG 3.1.3 OUTPUT IMAGE

# **3.2 TEXT DETECTION**

## 3.2.1 DATASET CREATION

In Text\_detection, when we provide the input image, all the text within the image will be detected along with its location and gets saved to a text file

In the beginning, we received ten folders from the client. Each folder had a PDF and some images of chairs in it. For text extraction, we need the ten pdfs. So, we selected the ten PDFs to a folder for further processing.

## 

## FIG 3.2.1 FOLDER STRUCTURE WITH TEN PDFS

The above figure shows the folder structure with ten PDFs. Each pdf has two pages which makes a total of 20 pages. Each page has to be converted to JPEG image format before augmentation. With the code for converting pdf to jpg, all the 20 pages are converted to respective images. The figure below shows the 20 images.

FIG 3.2.1 TOTAL IMAGES

### 3.2.1.1 DATA AUGMENTATION

When there is only limited data available for training we can use Data Augmentation techniques through which we can generate additional images based on the original image. From these data collection we may need to carefully select a right mixture of sample data for building the model. This technique also helps in a simple way to prevent the model from over fitting.

Techniques include:

1. Warp shift  
   Image warping is the process of digitally manipulating an image such that any shapes portrayed in the image have been significantly distorted.
2. Gamma  
   Gamma is a nonlinear operation used to encode and decode luminance or tristimulus values in image.

Example:

The below are the samples generated from the original single Image. Each image is generated by one more technique in random with the parameter value variation randomly applied.

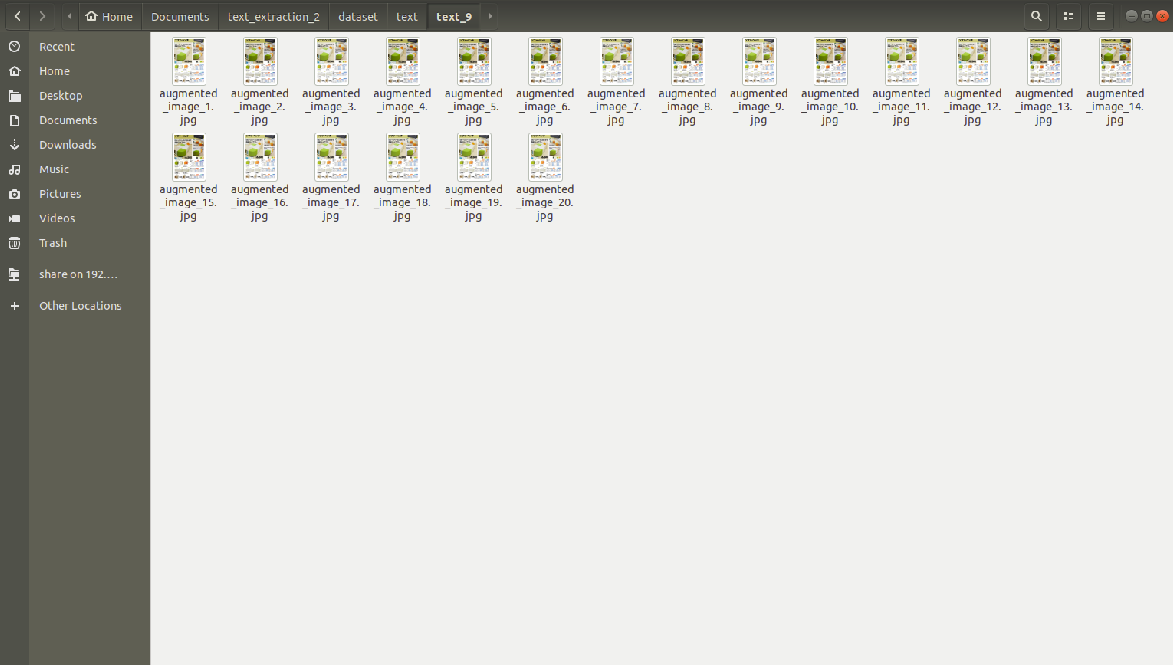


FIG 3.2.1.1 AUGMENTED IMAGES

After data augmentation, we got twenty samples of each class which made the images to a total of 400 images.Before starting training, we have to plot the x-y coordinates of all the text and tables within an image manually and then use this plot to create the csv file.

### 3.2.1.2 CSV CREATION

After creating the dataset, we manually plotted the x-y coordinates of each text and table using a tool. In this way we created the text files of each PDF to JPG images. We cannot do training with the text file. So we created comma seperated files(csv files) from text files.

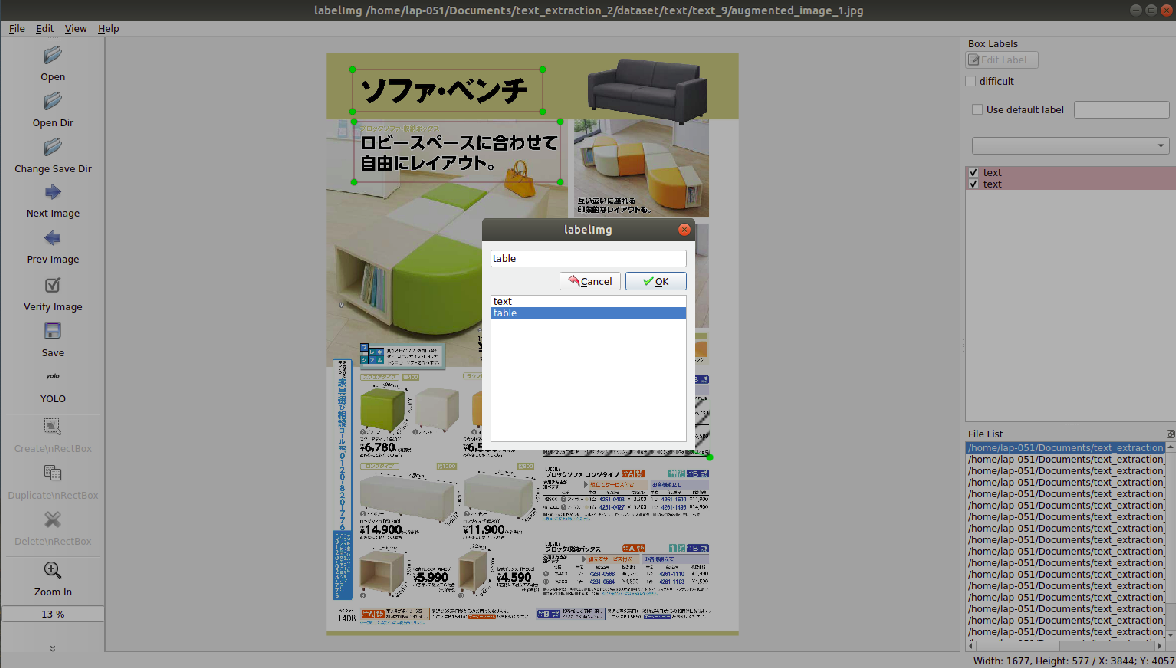


FIG 3.2.1.2 CLASS SELECTION



FIG 3.2.1.2 COORDINATES PLOTTING OF TEXT AND TABLE

The figure 1.2.1 and figure 1.2.2 shows how the coordinates are plotted using the software tool.When we plot the coordinates using this tool and save it, it automatically creates the text\_files within the folder. This text files are further used to create the ‘annotations’. The figure below shows the way text files are saved. It shows the twenty text files of the corresponding twenty augmented images of a single pdf to jpg image.

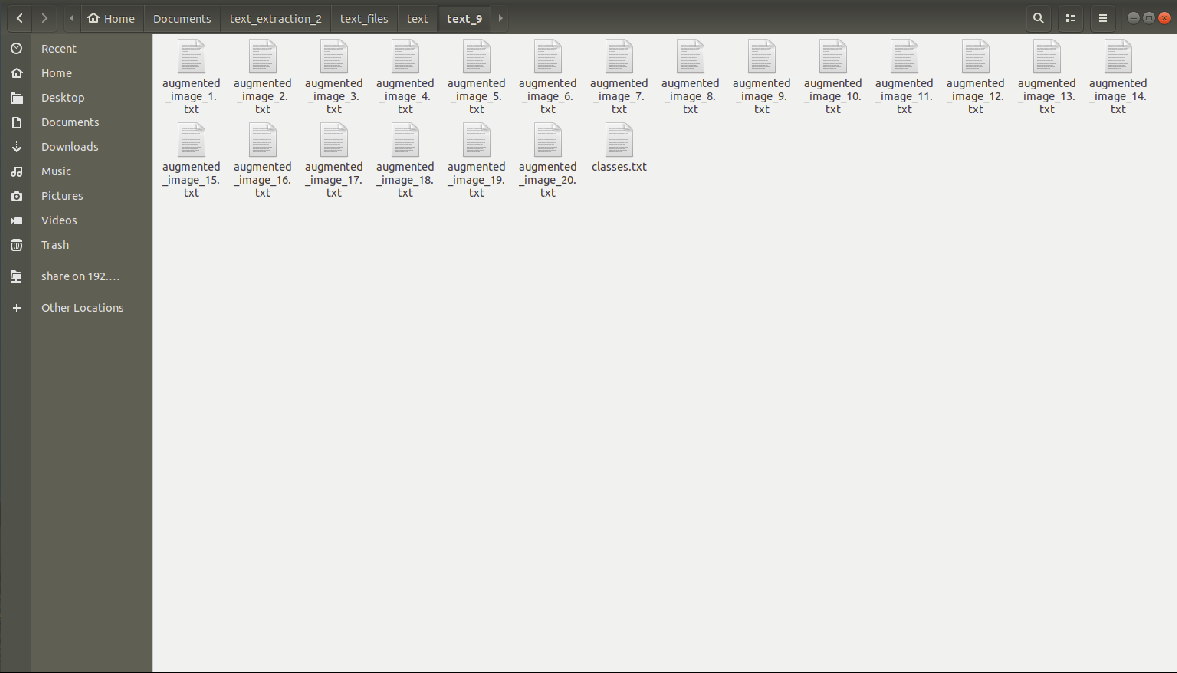


FIG 3.2.1.2 TEXT FILES

## 3.2.2 TRAINING

It's not possible to do the training in Jupyter Notebook as we are dealing with a huge dataset, we need enough memory. So we chose google colab as the running space. Upload the dataset, csv file and all other necessary input files to the drive and then run the python code. The libraries and its versions are mentioned below:

1. Keras - 2.1.5
2. Tensorflow - 1.6.0

After importing the following versions of keras and Tensorflow, we use a code to convert the darknet yolo model to keras. The python code is saved as ‘convert.py’. To run the code, we imported the following libraries:

* Numpy
* Keras.backend
* Keras.layers
* Keras.models
* Keras.optimizers
* Keras.callbacks

### 3.2.2.1 CODE EXPLANATION

After importing the necessary libraries, open the inputs (annotations.txt, yolo\_anchors\_path.txt and classes.txt) using the path.

**annotations.txt** - The csv file of dataset

**classes.txt** - The two classes are text and table

**yolo\_anchors.txt** - We have this file inside ‘model\_data’ in which there are some predefined dimensions. When we do prediction, the objects are detected with a bounding box, but before that the bounding box gets converted to some dimension within the yolo\_anchors.txt file and then gets displayed. So the the real bounding box that we are seeing is not the actual one but the converted one from yolo\_anchors.txt.

The input shape is regularized as (416,416) which is defined by ‘input\_shape’. Model means the yolo model which gets created during training. Moreover, a checkpoint is used to save the checkpoints after every three epochs. Reduce learning is used when model learning does not improve and thereby learning rate get reduced. Again it tries to improve little by little and if it's completely not possible, the training gets ‘early stopped’. The two variables ‘num\_validation’ and ‘num\_train’ are used to split the training data in such a way that 80% for training and 20% for validation.

Initially, the first few layers are given as frozen to maintain a stable loss, because first layers have the general features and deep learning occurs in the last layers. Some parameters which we can change during the training are:

EPOCH - It means the number of iterations of training which we can change accordingly. The more the number, the better will be accuracy.

BATCH SIZE - It decides how many samples should be taken in one iteration for training. After that, we save a weight and using this weight we unfreeze the frozen layers for fine-tuning. For fine-tuning, we do the second training. In this way, the training proceeds and trainied\_weight gets saved in the location provided in drive. When training gets completed, take the ‘trained\_weight\_final.h5’ which is the final trained\_weight for testing.

## 3.2.3 TESTING

The training data is used to make sure the machine recognizes patterns in the data, the cross-validation data is used to ensure better accuracy and efficiency of the algorithm used to train the machine, and the test data is used to see how well the machine can predict new answers based on its training. In machine learning we usually split our data into two subsets: training data and testing data.

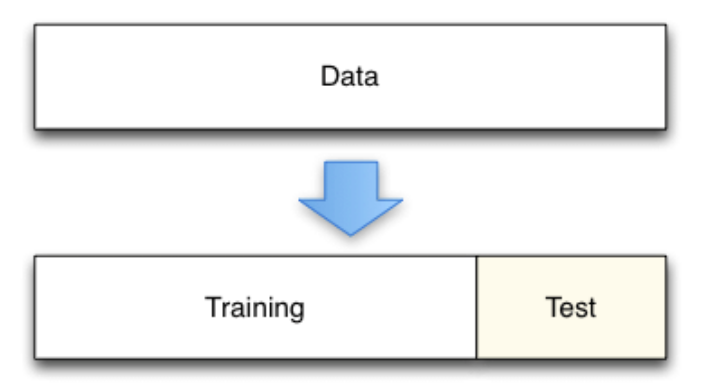


FIG 3.2.3 DATASET SPLITTING

### 3.2.3.1 TEST DATASET

When 80% of the dataset is used for training, the rest 20% is used for testing.

### 3.2.3.2 CODE EXPLANATION

After completing training, we get three parameters:

* Trained\_weight
* Class
* Image

First, the image gets passed to the loop and if it can open the image, it is passed to the function ‘yolo.detect\_image(image)’and the corresponding image is passed through the function ‘detect\_text’ in yolo.py.

In the function, ‘letter\_box’ the image size is converted to an optimum size accordingly. The parameters image\_width, image\_height, model\_width, model\_height are calculated. These parameters are used to scale the image and resize the image.

After detecting the object(text and table) the bounding box is plotted and each bounding box is used for text extraction.‘Pytesseract’ is used to convert an image to string. It detects the text within the boxes and converts to string. The corresponding texts are saved into a text file.

The output parameters we acquire are:

* Output box
* Score
* Class
* Extracted text

During testing, according to the size of the image we gave as input, thickness is decided and processed. Then during processing we have two output classes and processing occurs one by one. Image position is calculated, rectangular box is drawn, label is decided for every box and all the processes continue for every class. After processing the loops for drawing rectangle, writing the class name and all, it gets mapped to the new image. Options are given to choose the color of the rectangular box and thickness of text.

An example of input image given for testing is shown below:



FIG 3.2.3 INPUT IMAGE

Output image is shown below:



FIG 3.2.3 OUTPUT IMAGE

The output text file contains of the entire text and table details with its location as shown in figure:

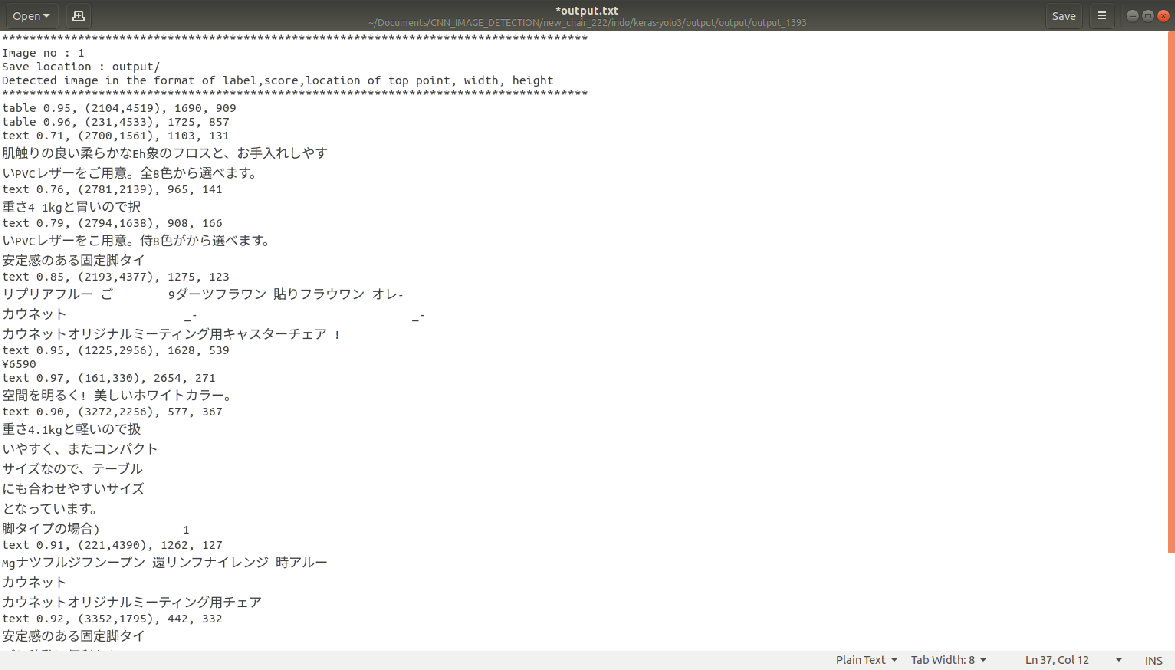


FIG 3.2.3 OUTPUT TEXT FILE