Object Detection based on Two-stage and One-stage Convolutional Neural Network

Presented by:

Veronica Naosekpam
PhD Scholar
Computer Science & Engineering
Indian Institute of Information Technology Guwahati, Assam, India

Two-stage Object Detector : RCNN[1]

- Object detection system consists of three modules:
  - First generates category-independent region proposals(through selective search here).
  - Second module is a large convolutional neural network that extracts a fixed-length feature vector from each region
  - Third module is a set of class specific linear SVMs.

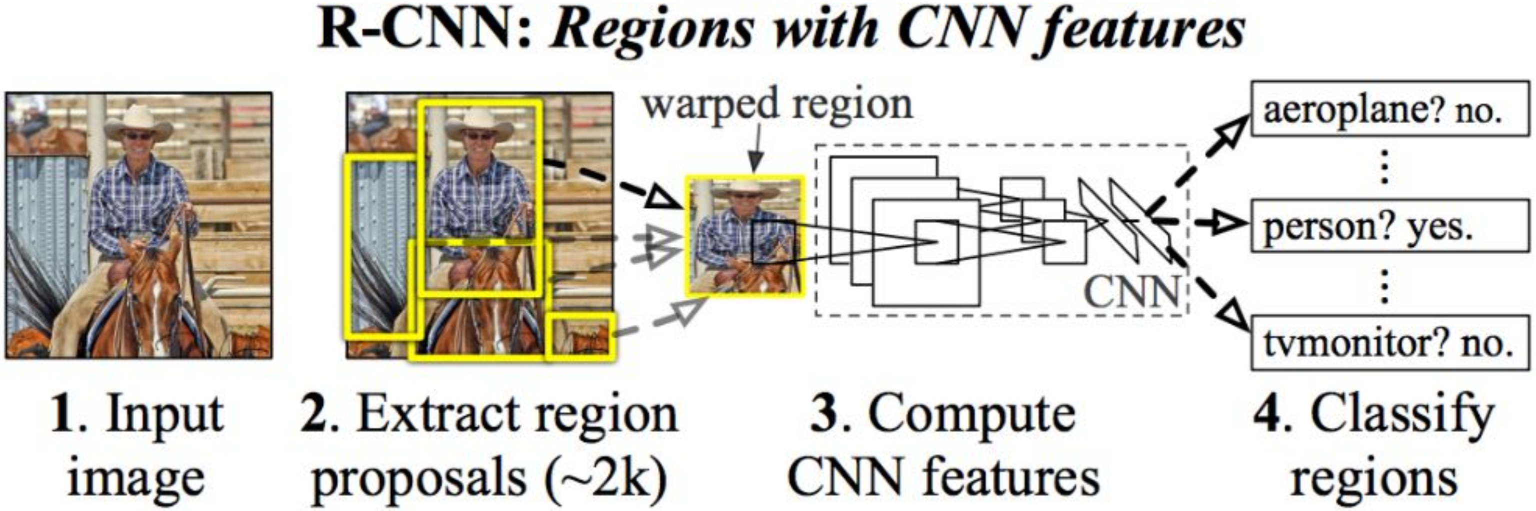
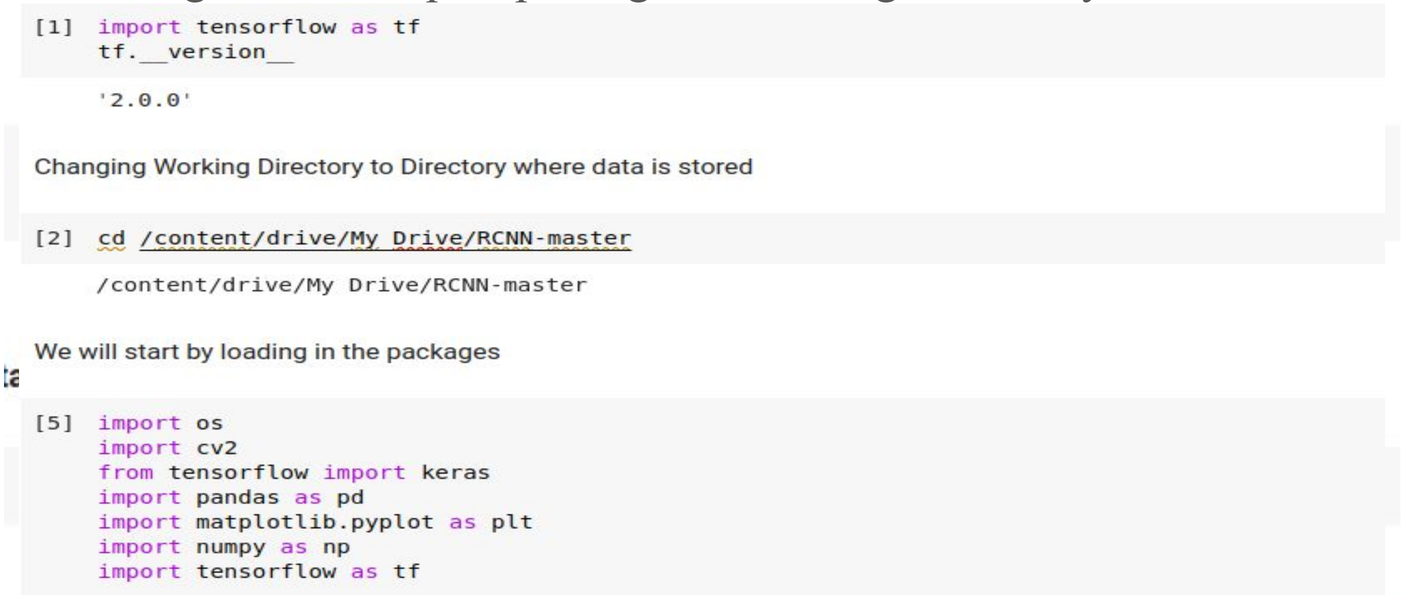


Fig 1 : RCNN architecture

- Mount GoogleDrive, import packages and change directory



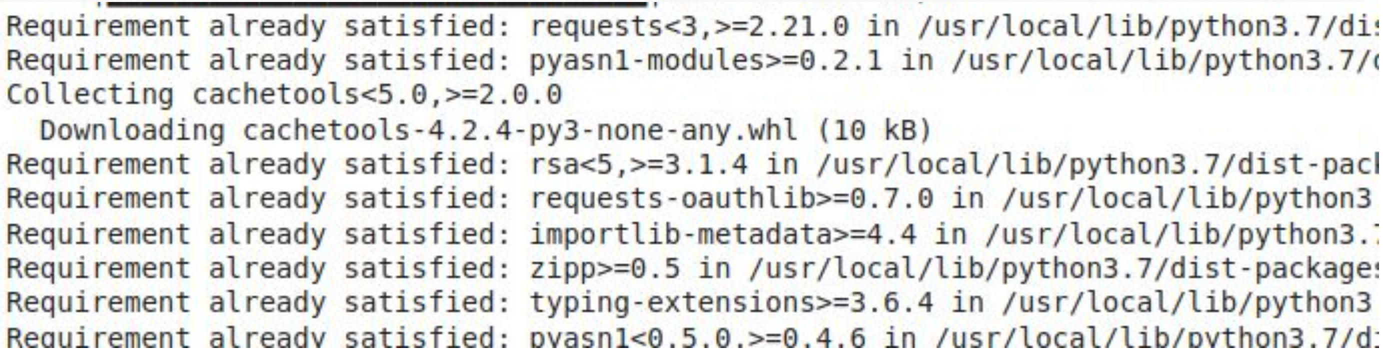
```
[1] import tensorflow as tf
    tf.__version__

    '2.0.0'
```

Changing Working Directory to Directory where data is stored

```
[2] cd /content/drive/My Drive/RCNN-master

    /content/drive/My Drive/RCNN-master
```

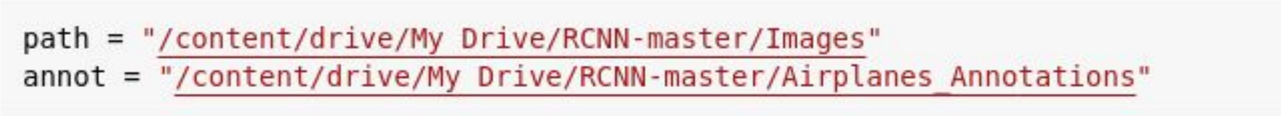
We will start by loading in the packages

```
[5] import os
    import cv2
    from tensorflow import keras
    import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np
    import tensorflow as tf
```

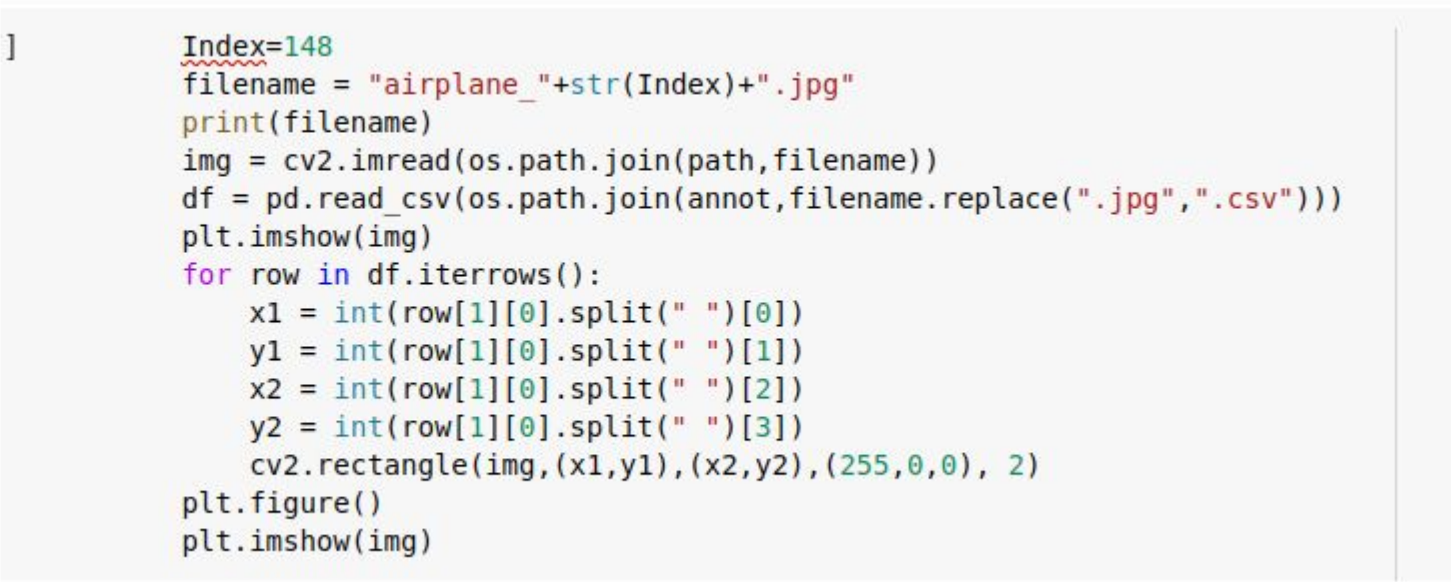


```
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/
Collecting cachetools<5.0,>=2.0.0
  Downloading cachetools-4.2.4-py3-none-any.whl (10 kB)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/d
```

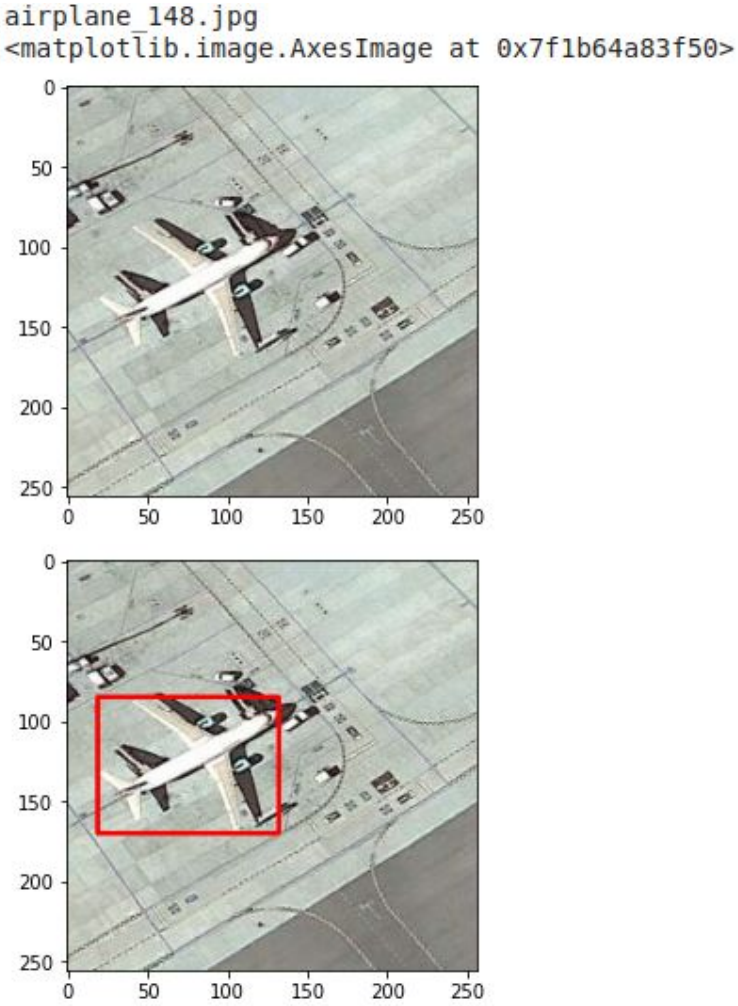
- Set the path and annot to the address of your image folder and Airplanes_Annotations folder.
- Index below and re-run to see different examples.



```
path = "/content/drive/My Drive/RCNN-master/Images"
annot = "/content/drive/My Drive/RCNN-master/Airplanes_Annotations"
```

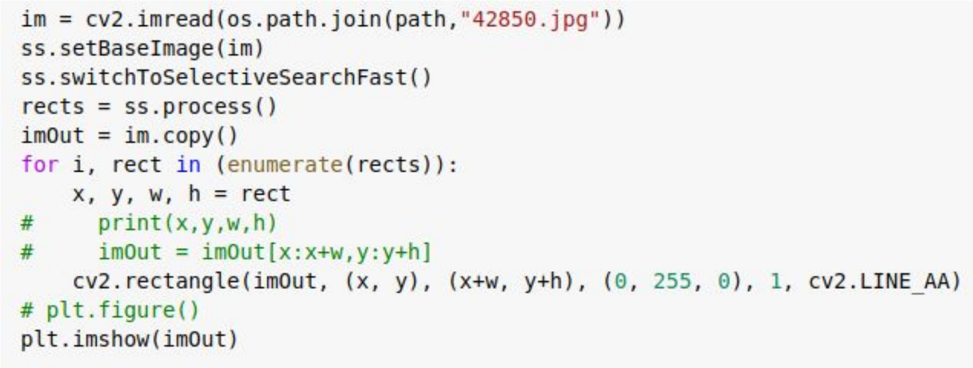


```
Index=148
filename = "airplane_"+str(Index)+".jpg"
print(filename)
img = cv2.imread(os.path.join(path,filename))
df = pd.read_csv(os.path.join(annot,filename.replace(".jpg",".csv")))
plt.imshow(img)
for row in df.iterrows():
    x1 = int(row[1][0].split(" ")[0])
    y1 = int(row[1][0].split(" ")[1])
    x2 = int(row[1][0].split(" ")[2])
    y2 = int(row[1][0].split(" ")[3])
    cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,0), 2)
plt.figure()
plt.imshow(img)
```



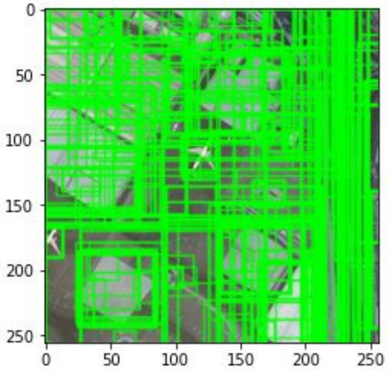
```
airplane_148.jpg
<matplotlib.image.AxesImage at 0x7f1b64a83f50>
```

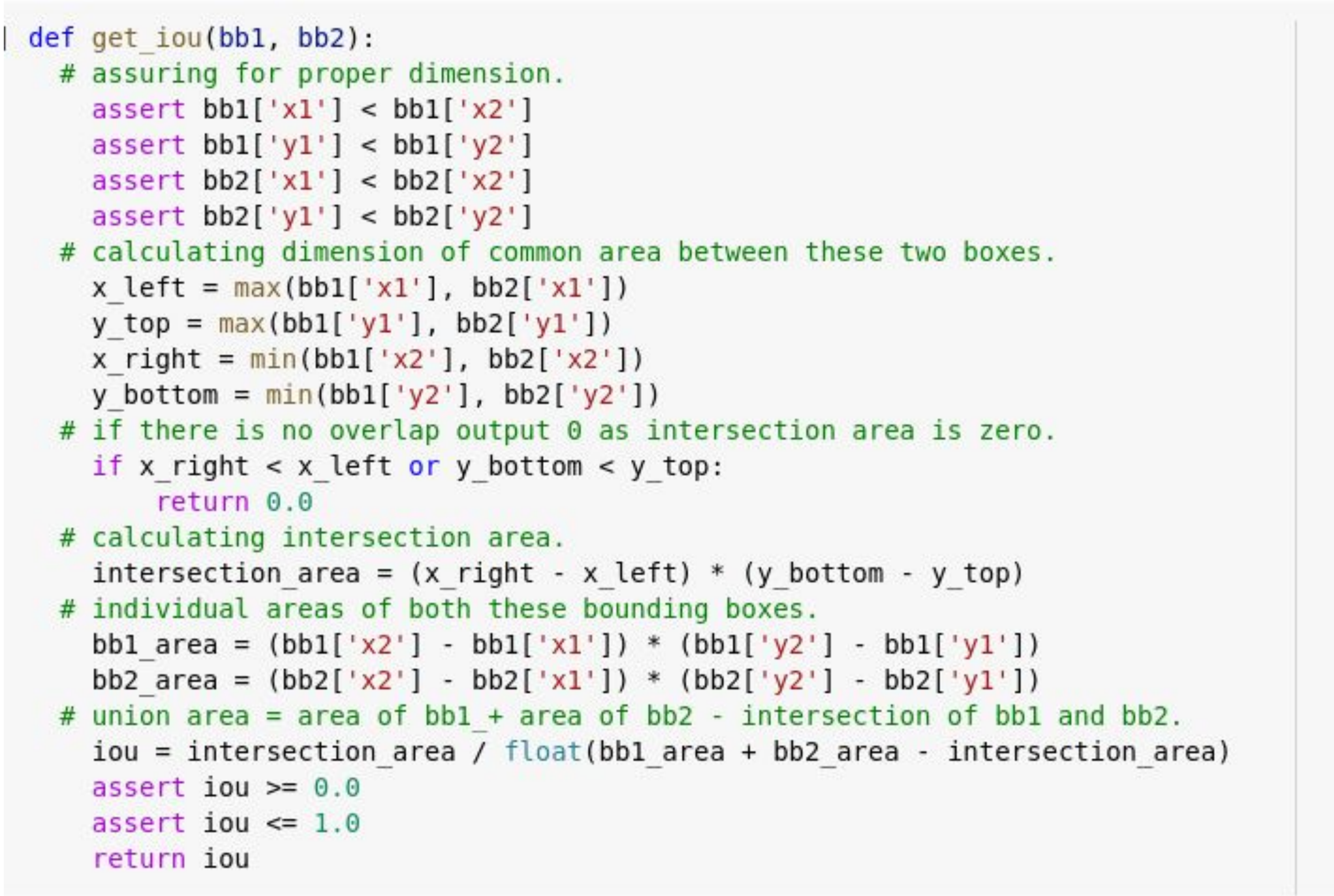
- Selective search we use cv2 library



```
im = cv2.imread(os.path.join(path,"42850.jpg"))
ss.setBaseImage(im)
ss.switchToSelectiveSearchFast()
rects = ss.process()
imOut = im.copy()
for i, rect in (enumerate(rects)):
    x, y, w, h = rect
#     print(x,y,w,h)
#     imOut = imOut[x:x+w,y:y+h]
    cv2.rectangle(imOut, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
# plt.figure()
plt.imshow(imOut)
```

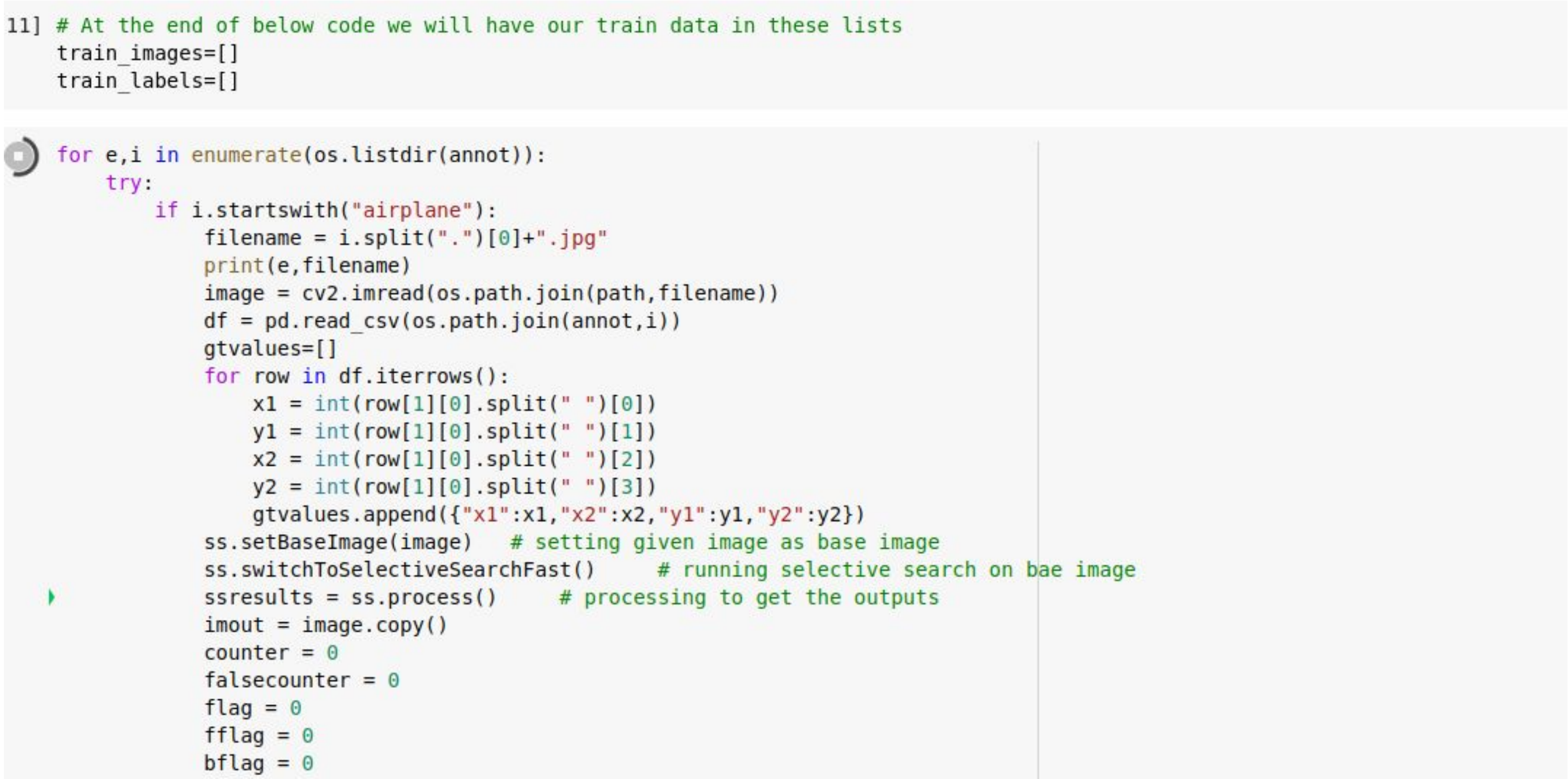
```
<matplotlib.image.AxesImage at 0x7f1b6459f850>
```





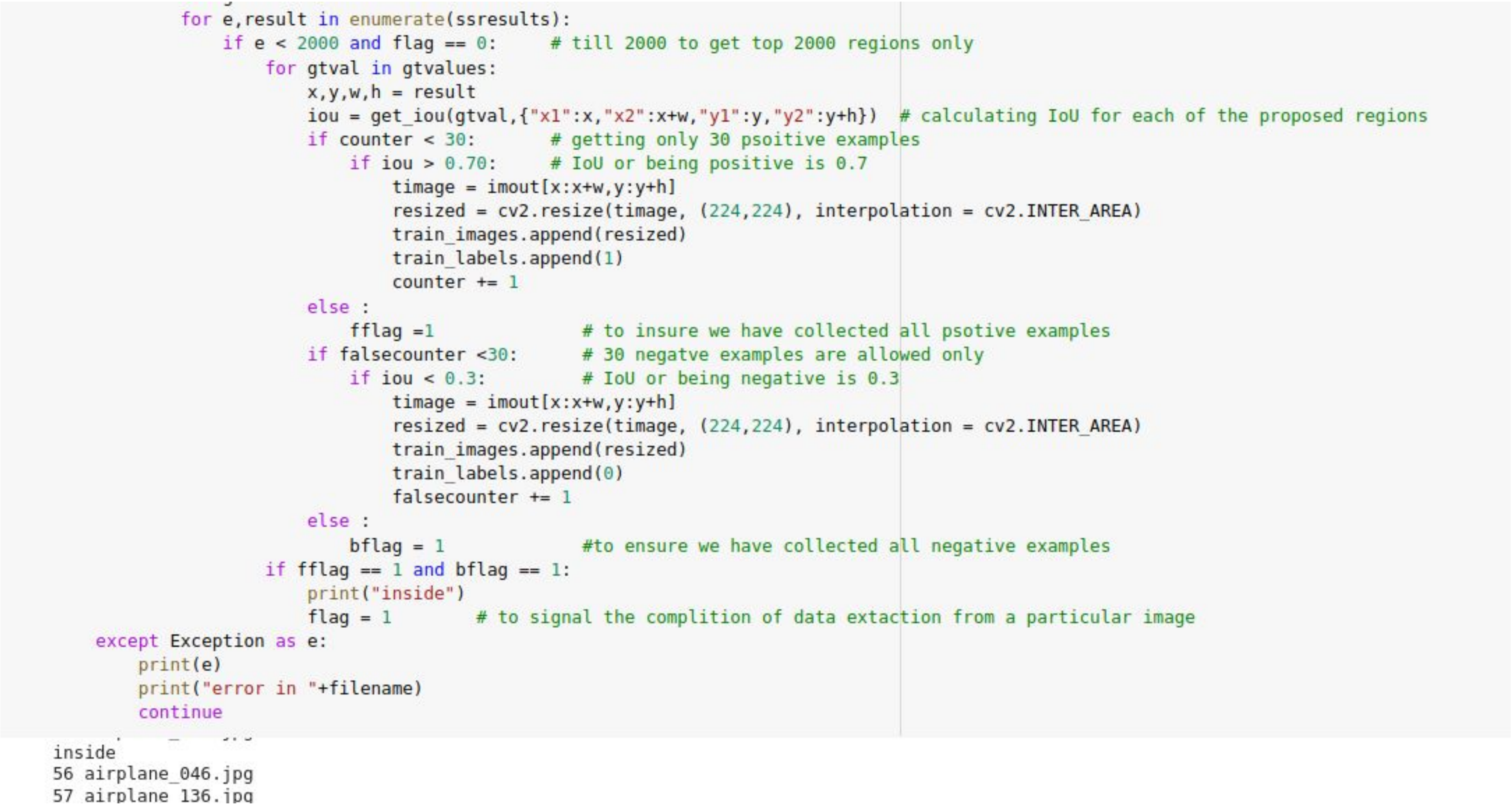
```python
def get_iou(bb1, bb2):
 # assuring for proper dimension.
  assert bb1['x1'] < bb1['x2']
  assert bb1['y1'] < bb1['y2']
  assert bb2['x1'] < bb2['x2']
  assert bb2['y1'] < bb2['y2']
 # calculating dimension of common area between these two boxes.
  x_left = max(bb1['x1'], bb2['x1'])
  y_top = max(bb1['y1'], bb2['y1'])
  x_right = min(bb1['x2'], bb2['x2'])
  y_bottom = min(bb1['y2'], bb2['y2'])
 # if there is no overlap output 0 as intersection area is zero.
  if x_right < x_left or y_bottom < y_top:
    return 0.0
 # calculating intersection area.
  intersection_area = (x_right - x_left) * (y_bottom - y_top)
 # individual areas of both these bounding boxes.
  bb1_area = (bb1['x2'] - bb1['x1']) * (bb1['y2'] - bb1['y1'])
  bb2_area = (bb2['x2'] - bb2['x1']) * (bb2['y2'] - bb2['y1'])
 # union area = area of bb1_+ area of bb2 - intersection of bb1 and bb2.
  iou = intersection_area / float(bb1_area + bb2_area - intersection_area)
  assert iou >= 0.0
  assert iou <= 1.0
  return iou
```

- Boxes which have an IoU greater than 0.7 (original paper it's 0.5) are considered as a positive example.
- Boxes with relative low IoU 0.3 are taken to be negative examples.
- Number of regions taken here is 30 positive and 30 negative.



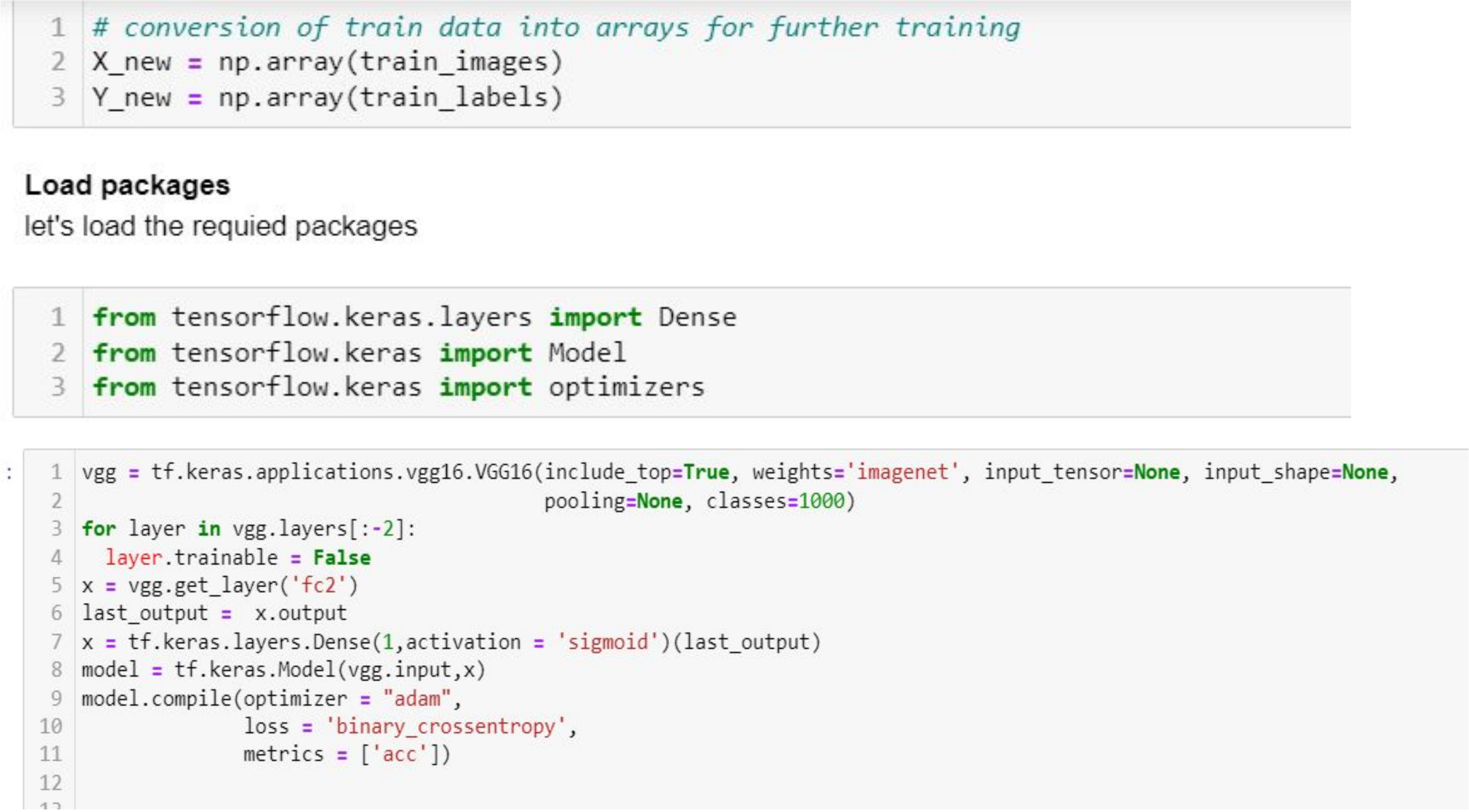
```
11] # At the end of below code we will have our train data in these lists
    train_images=[]
    train_labels=[]
```

```
for e,i in enumerate(os.listdir(annot)):
    try:
        if i.startswith("airplane"):
            filename = i.split(".")[0]+".jpg"
            print(e,filename)
            image = cv2.imread(os.path.join(path,filename))
            df = pd.read_csv(os.path.join(annot,i))
            gtvalues=[]
            for row in df.iterrows():
                x1 = int(row[1][0].split(" ")[0])
                y1 = int(row[1][0].split(" ")[1])
                x2 = int(row[1][0].split(" ")[2])
                y2 = int(row[1][0].split(" ")[3])
                gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})
            ss.setBaseImage(image)   # setting given image as base image
            ss.switchToSelectiveSearchFast()     # running selective search on bae image
            ssresults = ss.process()     # processing to get the outputs
            imout = image.copy()
            counter = 0
            falsecounter = 0
            flag = 0
            fflag = 0
            bflag = 0
```



```
                for e,result in enumerate(ssresults):
                    if e < 2000 and flag == 0:     # till 2000 to get top 2000 regions only
                        for gtval in gtvalues:
                            x,y,w,h = result
                            iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})  # calculating IoU for each of the proposed regions
                            if counter < 30:       # getting only 30 psoitive examples
                                if iou > 0.70:     # IoU or being positive is 0.7
                                    timage = imout[x:x+w,y:y+h]
                                    resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                    train_images.append(resized)
                                    train_labels.append(1)
                                    counter += 1
                            else :
                                fflag =1              # to insure we have collected all psotive examples
                            if falsecounter <30:      # 30 negatve examples are allowed only
                                if iou < 0.3:         # IoU or being negative is 0.3
                                    timage = imout[x:x+w,y:y+h]
                                    resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                    train_images.append(resized)
                                    train_labels.append(0)
                                    falsecounter += 1
                            else :
                                bflag = 1             #to ensure we have collected all negative examples
                        if fflag == 1 and bflag == 1:
                            print("inside")
                            flag = 1        # to signal the complition of data extaction from a particular image
        except Exception as e:
            print(e)
            print("error in "+filename)
            continue
```

```
inside
56 airplane_046.jpg
57 airplane_136.jpg
```



```python
# conversion of train data into arrays for further training
X_new = np.array(train_images)
Y_new = np.array(train_labels)
```

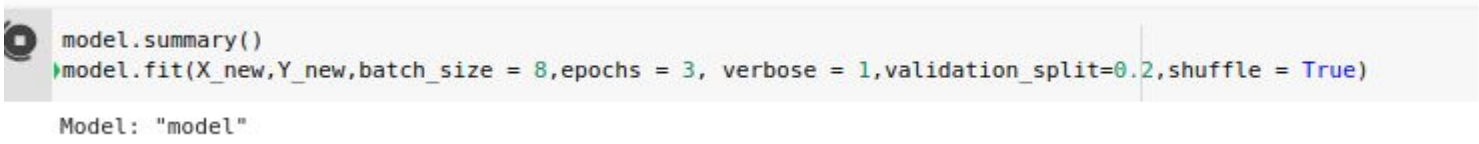
Load packages

let's load the requied packages

```python
from tensorflow.keras.layers import Dense
from tensorflow.keras import Model
from tensorflow.keras import optimizers
```

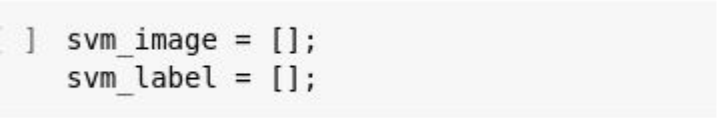
```python
vgg = tf.keras.applications.vgg16.VGG16(include_top=True, weights='imagenet', input_tensor=None, input_shape=None,
                                        pooling=None, classes=1000)
for layer in vgg.layers[:-2]:
  layer.trainable = False
x = vgg.get_layer('fc2')
last_output =  x.output
x = tf.keras.layers.Dense(1,activation = 'sigmoid')(last_output)
model = tf.keras.Model(vgg.input,x)
model.compile(optimizer = "adam",
              loss = 'binary_crossentropy',
              metrics = ['acc'])

```

- Summary of the model and training
- Making New Network with svm
- First create the dataset for the svm.

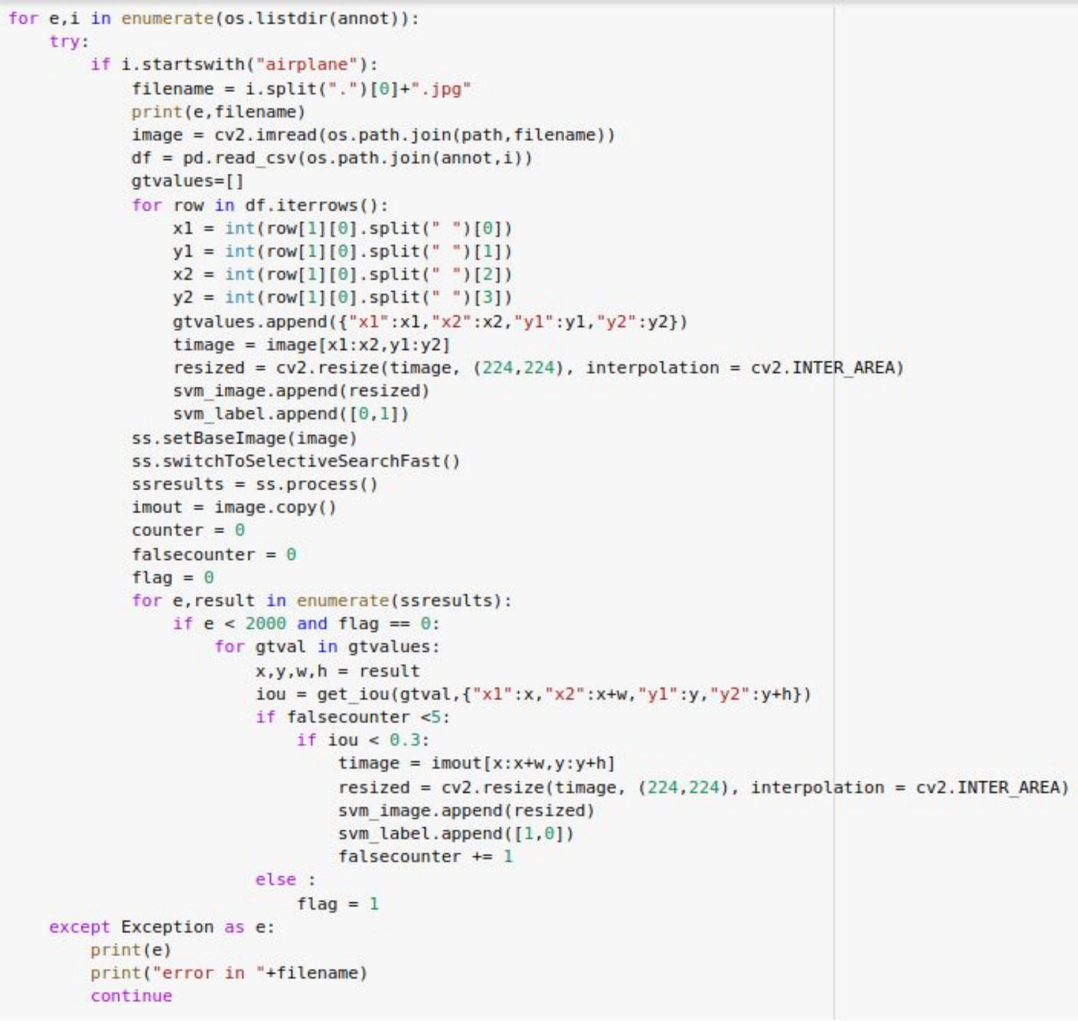


```
model.summary()
model.fit(X_new,Y_new,batch_size = 8,epochs = 3, verbose = 1,validation_split=0.2,shuffle = True)
```

```
Model: "model"
```

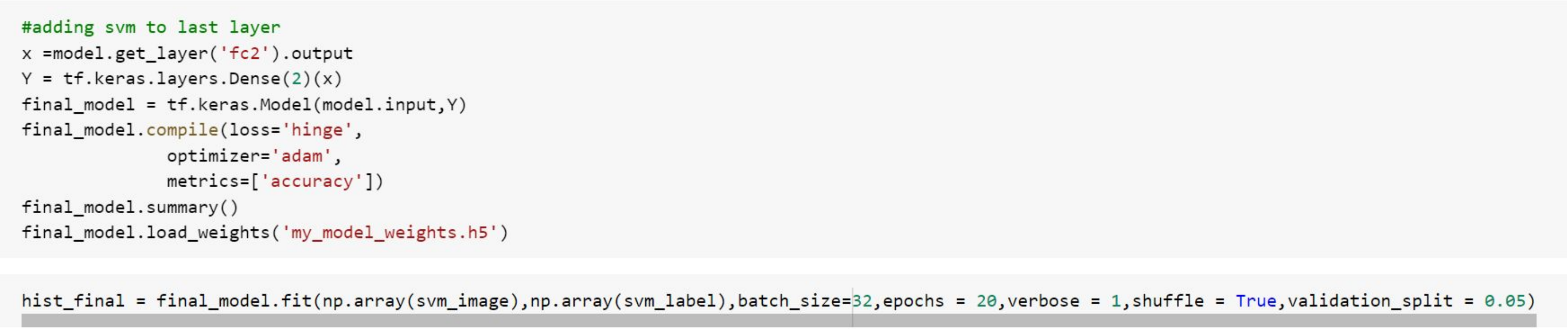


```
svm_image = [];
svm_label = [];
```



```
for e,i in enumerate(os.listdir(annot)):
    try:
        if i.startswith("airplane"):
            filename = i.split(".")[0]+".jpg"
            print(e,filename)
            image = cv2.imread(os.path.join(path,filename))
            df = pd.read_csv(os.path.join(annot,i))
            gtvalues=[]
            for row in df.iterrows():
                x1 = int(row[1][0].split(" ")[0])
                y1 = int(row[1][0].split(" ")[1])
                x2 = int(row[1][0].split(" ")[2])
                y2 = int(row[1][0].split(" ")[3])
                gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})
                timage = image[x1:x2,y1:y2]
                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                svm_image.append(resized)
                svm_label.append([0,1])
            ss.setBaseImage(image)
            ss.switchToSelectiveSearchFast()
            ssresults = ss.process()
            imout = image.copy()
            counter = 0
            falsecounter = 0
            flag = 0
            for e,result in enumerate(ssresults):
                if e < 2000 and flag == 0:
                    for gtval in gtvalues:
                        x,y,w,h = result
                        iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})
                        if falsecounter <5:
                            if iou < 0.3:
                                timage = imout[x:x+w,y:y+h]
                                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                svm_image.append(resized)
                                svm_label.append([1,0])
                                falsecounter += 1
                        else :
                            flag = 1
    except Exception as e:
        print(e)
        print("error in "+filename)
        continue
```

- For svm dataset we considered all ground truth bounding boxes as positive examples and those which were having an IOU less than 0.3 as false examples to increase the preciseness.



```
#adding svm to last layer
x =model.get_layer('fc2').output
Y = tf.keras.layers.Dense(2)(x)
final_model = tf.keras.Model(model.input,Y)
final_model.compile(loss='hinge',
              optimizer='adam',
              metrics=['accuracy'])
final_model.summary()
final_model.load_weights('my_model_weights.h5')
```

```
hist_final = final_model.fit(np.array(svm_image),np.array(svm_label),batch_size=32,epochs = 20,verbose = 1,shuffle = True,validation_split = 0.05)
```

One-stage Object Detector : RetinaNet [2]

- Facebook AI Research.
- Work well with dense and small scale objects.
  Handles imbalances and inconsistencies of the single-shot object detectors like YOLO and SSD while dealing with extreme foreground-background classes.

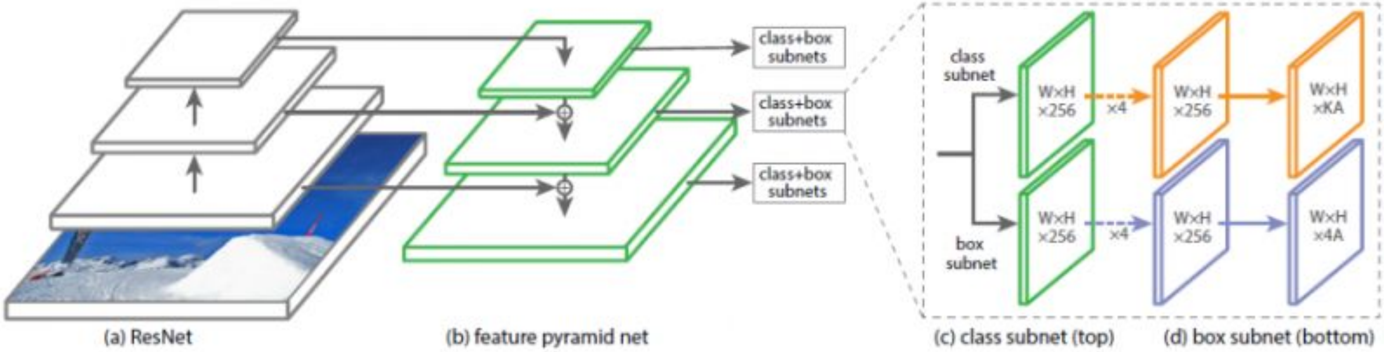
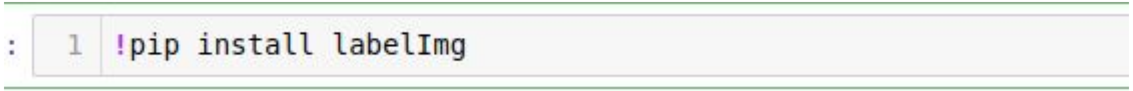


Fig 2 : RetinaNet architecture

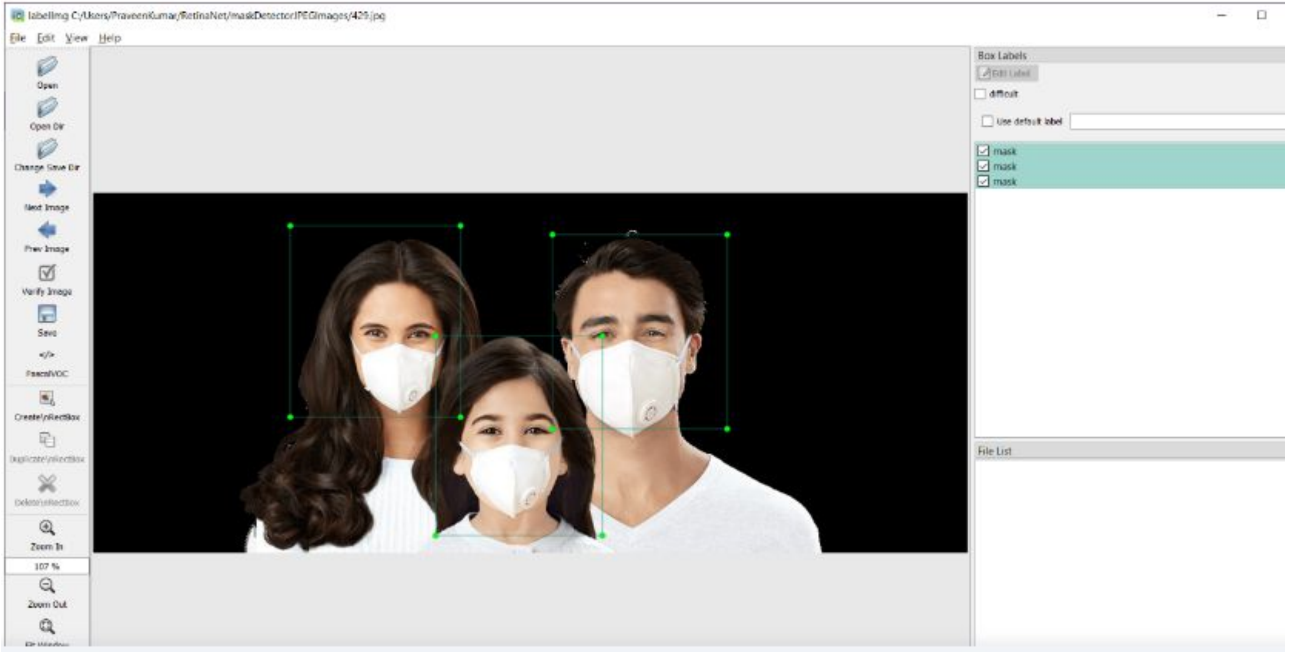
1. Backbone Network :
    a. Bottom up pathway : feature extraction.
    b. Top down pathway with lateral connections : upsamples the spatially coarser feature maps from higher pyramid levels,
        i. Lateral connections merge the top-down layers and the bottom-up layers with the same spatial size.
        ii. Higher-level feature maps tend to have small resolution
            1. Suitable for detecting larger objects.
        iii. Grid cells from lower-level feature maps have high resolution
            1. Better at detecting smaller objects
        iv. Combination of the top-down pathway and its lateral connections with bottom up the pathway, which do not require much extra computation,
    c. Scale Invariant.

2. Subnetwork for object Classification :
    a. A fully convolutional network (FCN) is attached to each FPN level for object classification.
    b. Incorporates 3 * 3 convolutional layers with 256 filters followed by another 3 * 3 convolutional layer with K * A filters.
    c. Output feature map size = W*H*KA.
    d. Sigmoid layer is used for object classification.
3. Subnetwork for object regression:
    a. Attached to each feature map of the FPN in parallel to the classification subnetwork.
    b. Design is identical to the classification subnet, except that the last convolutional layer is of size 3*3 with 4 filters resulting in an output feature map with the size of W*H*4A .
4. Focal Loss :
    a. Improved version of Cross-Entropy Loss (CE) .
    b. Ties to handle the class imbalance problem by assigning more weights to hard or easily misclassified examples and to down-weight easy examples.

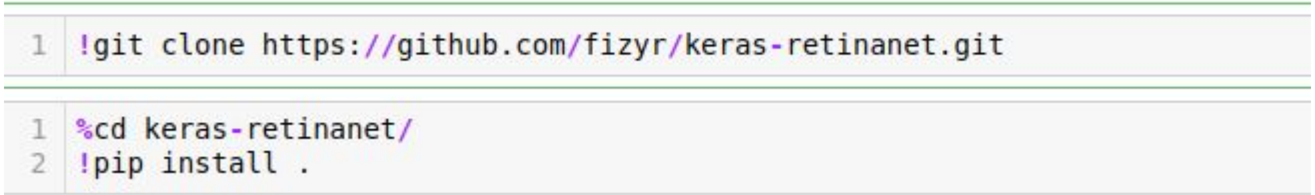
- Create Data Set.
- LabelImg : annotation tool lets you quickly annotate the bounding boxes of the objects.



```
!pip install labelImg
```

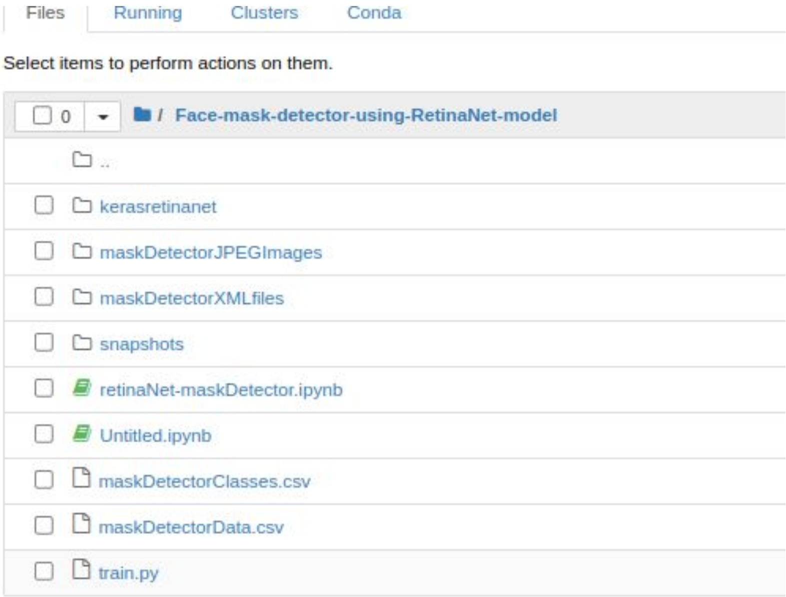


- Clone & install the keras-retinanet repository



```
!git clone https://github.com/fizyr/keras-retinanet.git
```

```
%cd keras-retinanet/
!pip install .
```

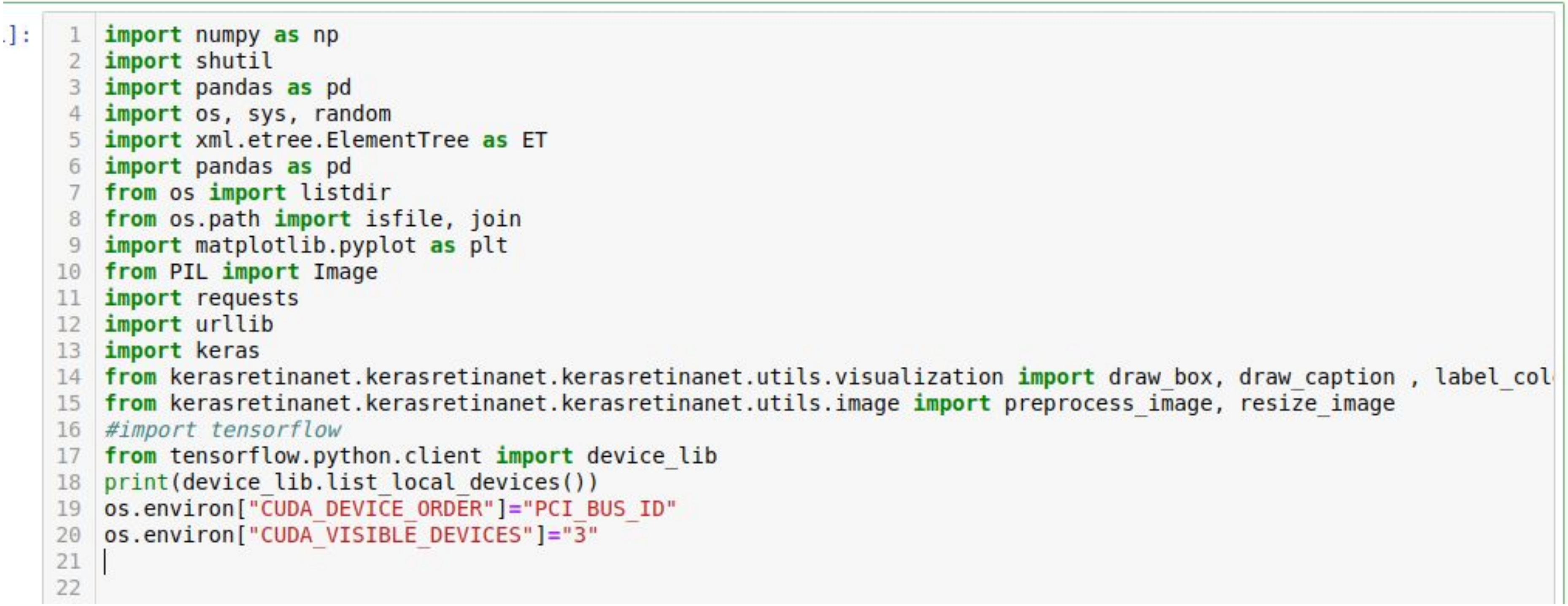


Select items to perform actions on them.

□ 0 ▾ / Face-mask-detector-using-RetinaNet-model

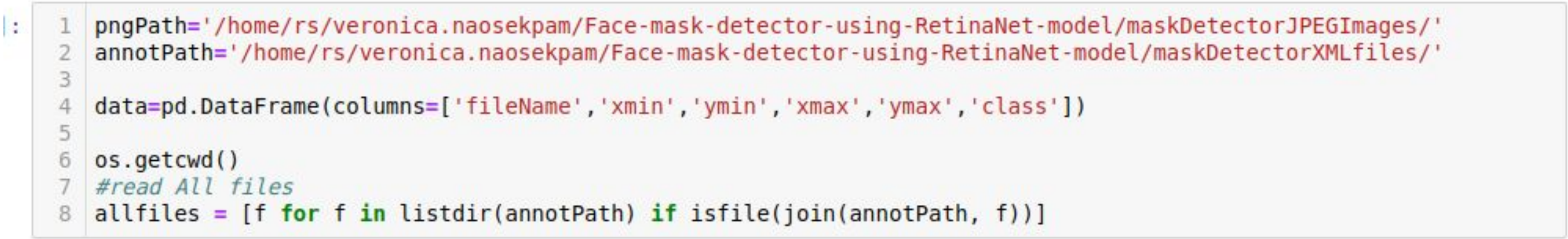
- ..
- □ kerasretinanet
- □ maskDetectorJPEGImages
- □ maskDetectorXMLfiles
- □ snapshots
- □ retinaNet-maskDetector.ipynb
- □ Untitled.ipynb
- □ maskDetectorClasses.csv
- □ maskDetectorData.csv
- □ train.py

- Import all required libraries.

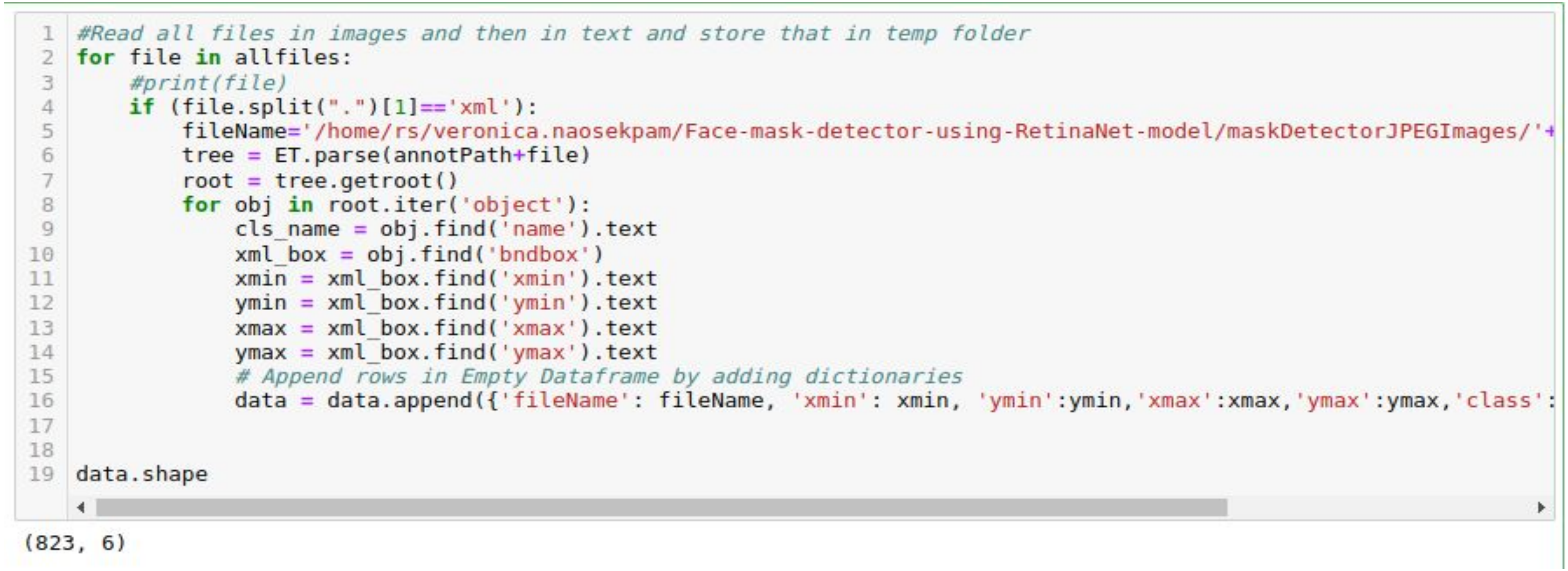


```python
import numpy as np
import shutil
import pandas as pd
import os, sys, random
import xml.etree.ElementTree as ET
import pandas as pd
from os import listdir
from os.path import isfile, join
import matplotlib.pyplot as plt
from PIL import Image
import requests
import urllib
import keras
from kerasretinanet.kerasretinanet.kerasretinanet.utils.visualization import draw_box, draw_caption , label_col
from kerasretinanet.kerasretinanet.kerasretinanet.utils.image import preprocess_image, resize_image
#import tensorflow
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="3"
```

- Import JPEG & xml data

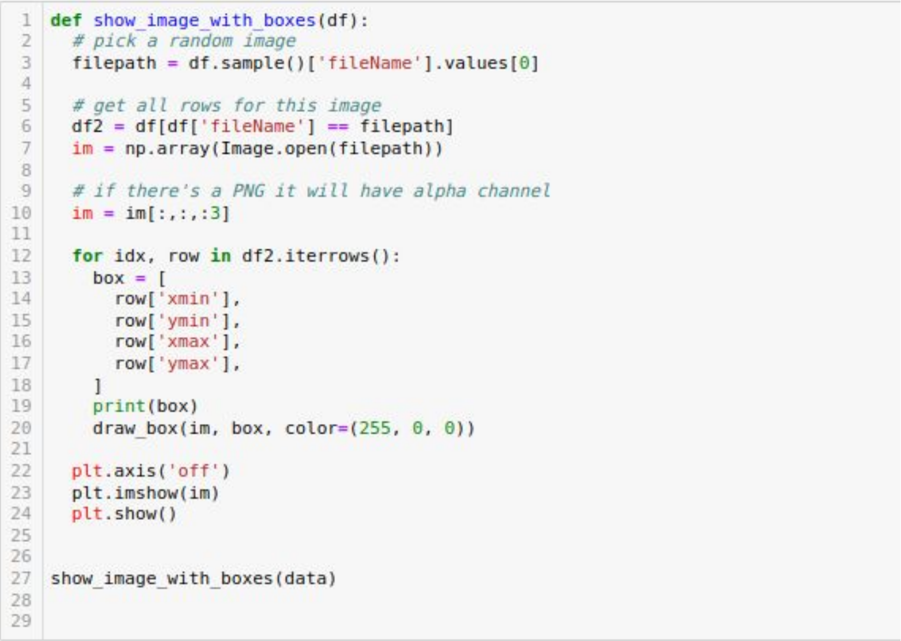


```
pngPath='/home/rs/veronica.naosekpam/Face-mask-detector-using-RetinaNet-model/maskDetectorJPEGImages/'
annotPath='/home/rs/veronica.naosekpam/Face-mask-detector-using-RetinaNet-model/maskDetectorXMLfiles/'

data=pd.DataFrame(columns=['fileName','xmin','ymin','xmax','ymax','class'])

os.getcwd()
#read All files
allfiles = [f for f in listdir(annotPath) if isfile(join(annotPath, f))]
```



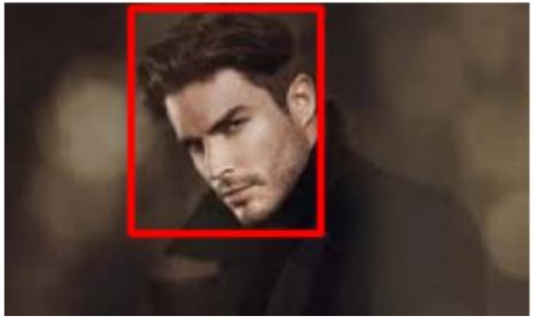
```
#Read all files in images and then in text and store that in temp folder
for file in allfiles:
    #print(file)
    if (file.split(".")[1]=='xml'):
        fileName='/home/rs/veronica.naosekpam/Face-mask-detector-using-RetinaNet-model/maskDetectorJPEGImages/'+
        tree = ET.parse(annotPath+file)
        root = tree.getroot()
        for obj in root.iter('object'):
            cls_name = obj.find('name').text
            xml_box = obj.find('bndbox')
            xmin = xml_box.find('xmin').text
            ymin = xml_box.find('ymin').text
            xmax = xml_box.find('xmax').text
            ymax = xml_box.find('ymax').text
            # Append rows in Empty Dataframe by adding dictionaries
            data = data.append({'fileName': fileName, 'xmin': xmin, 'ymin':ymin,'xmax':xmax,'ymax':ymax,'class':


data.shape
```

```
(823, 6)
```



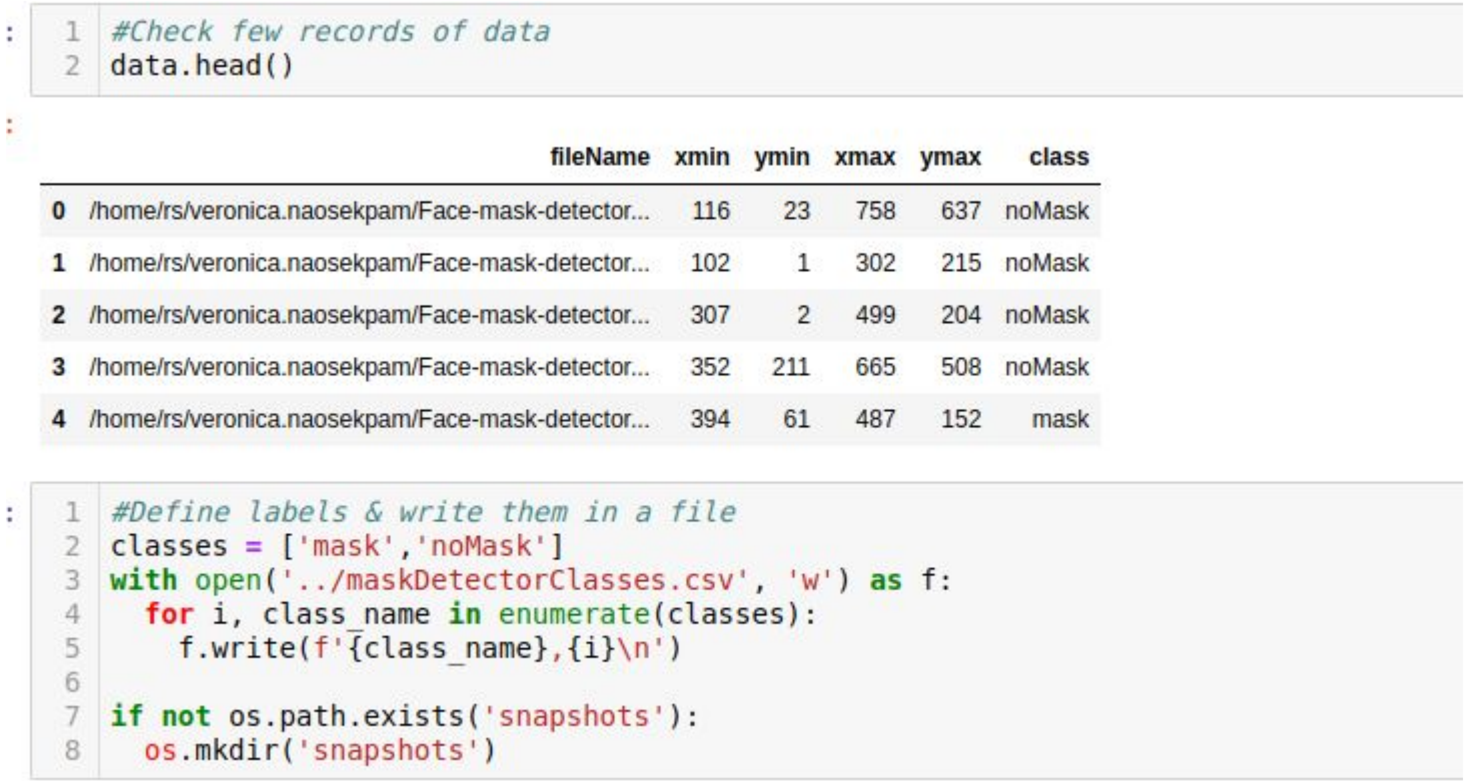
```python
def show_image_with_boxes(df):
  # pick a random image
  filepath = df.sample()['fileName'].values[0]

  # get all rows for this image
  df2 = df[df['fileName'] == filepath]
  im = np.array(Image.open(filepath))

  # if there's a PNG it will have alpha channel
  im = im[:,:,:3]

  for idx, row in df2.iterrows():
    box = [
      row['xmin'],
      row['ymin'],
      row['xmax'],
      row['ymax'],
    ]
    print(box)
    draw_box(im, box, color=(255, 0, 0))

  plt.axis('off')
  plt.imshow(im)
  plt.show()


show_image_with_boxes(data)
```

```
['57', '2', '143', '104']
```



- Show bounding boxes on the training dataset.

- Check few records of data.
- Define labels & write them in a file.

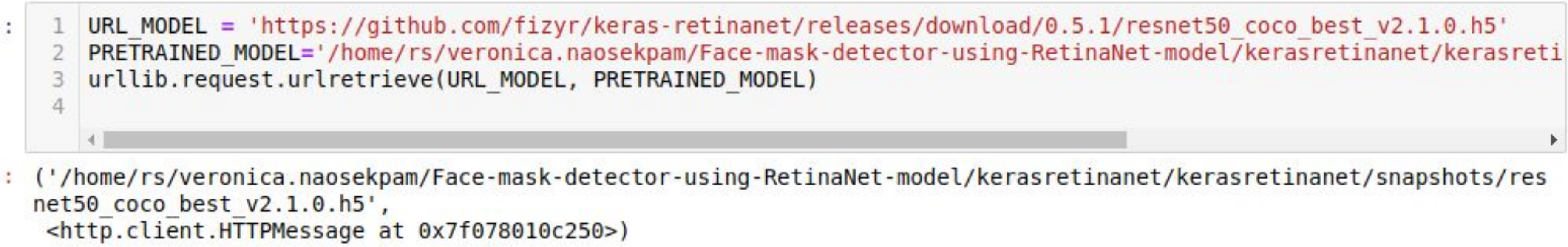


```
#Check few records of data
data.head()
```

	fileName	xmin	ymin	xmax	ymax	class
0	/home/rs/veronica.naosekpam/Face-mask-detector...	116	23	758	637	noMask
1	/home/rs/veronica.naosekpam/Face-mask-detector...	102	1	302	215	noMask
2	/home/rs/veronica.naosekpam/Face-mask-detector...	307	2	499	204	noMask
3	/home/rs/veronica.naosekpam/Face-mask-detector...	352	211	665	508	noMask
4	/home/rs/veronica.naosekpam/Face-mask-detector...	394	61	487	152	mask

```
#Define labels & write them in a file
classes = ['mask','noMask']
with open('../maskDetectorClasses.csv', 'w') as f:
  for i, class_name in enumerate(classes):
    f.write(f'{class_name},{i}\n')

if not os.path.exists('snapshots'):
  os.mkdir('snapshots')
```

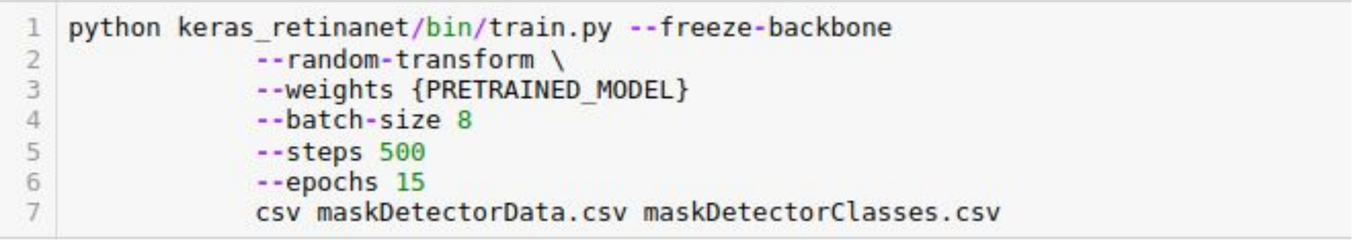
- Start with a pre-trained model : ResNet50 model pre-trained on the Coco dataset.



```
URL_MODEL = 'https://github.com/fizyr/keras-retinanet/releases/download/0.5.1/resnet50_coco_best_v2.1.0.h5'
PRETRAINED_MODEL='/home/rs/veronica.naosekpam/Face-mask-detector-using-RetinaNet-model/kerasretinanet/kerasreti
urllib.request.urlretrieve(URL_MODEL, PRETRAINED_MODEL)
```

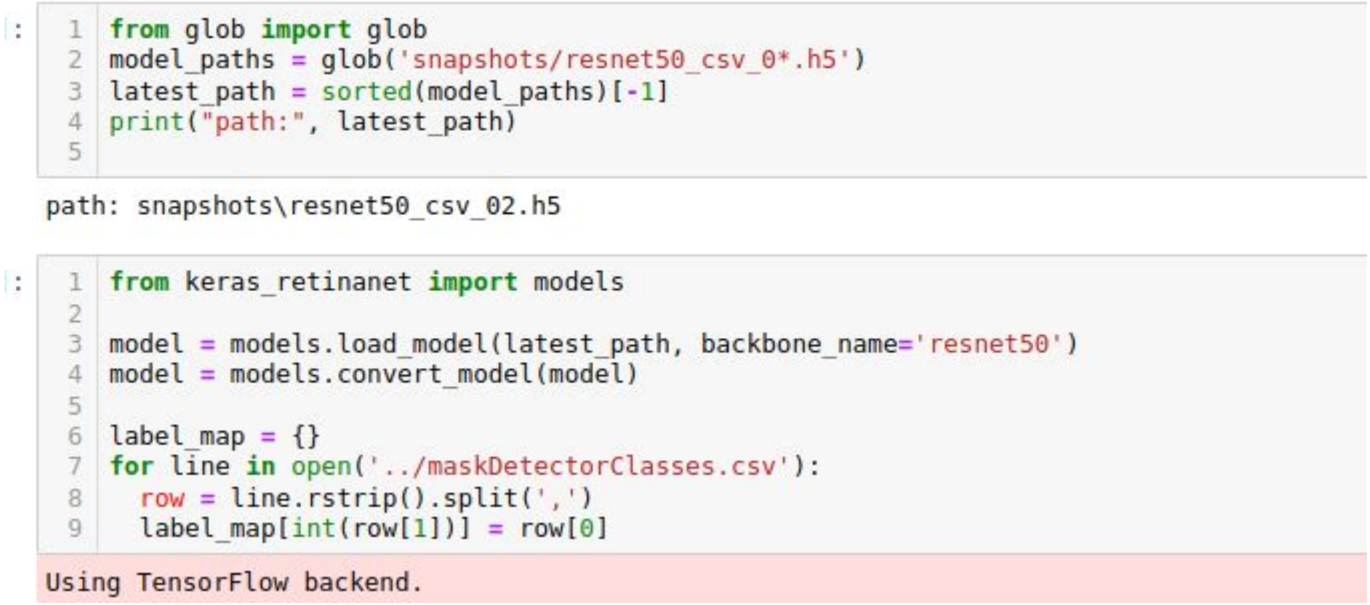
```
('/home/rs/veronica.naosekpam/Face-mask-detector-using-RetinaNet-model/kerasretinanet/kerasretinanet/snapshots/res
net50_coco_best_v2.1.0.h5',
 <http.client.HTTPMessage at 0x7f078010c250>)
```

- freeze-backbone: freeze the backbone layers, particularly useful when we use a small dataset, to avoid overfitting
- random-transform: randomly transform the dataset to get data augmentation
- weights: initialize the model with a pre-trained model.
- batch-size: training batch size, the higher value gives a smoother learning curve
- steps: Number of steps for epochs
- epochs: number of epochs to train
- csv: annotations files generated by the script above



```
python keras_retinanet/bin/train.py --freeze-backbone
            --random-transform \
            --weights {PRETRAINED_MODEL}
            --batch-size 8
            --steps 500
            --epochs 15
            csv maskDetectorData.csv maskDetectorClasses.csv
```

- Load the trained model.



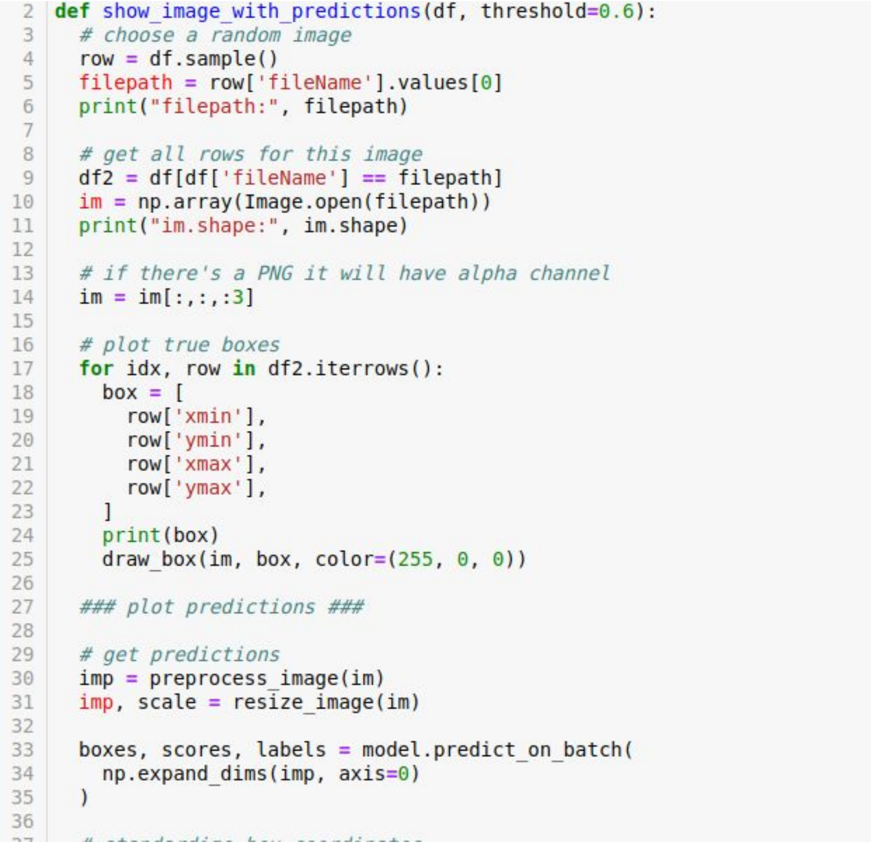
```
from glob import glob
model_paths = glob('snapshots/resnet50_csv_0*.h5')
latest_path = sorted(model_paths)[-1]
print("path:", latest_path)

```

```
path: snapshots\resnet50_csv_02.h5
```

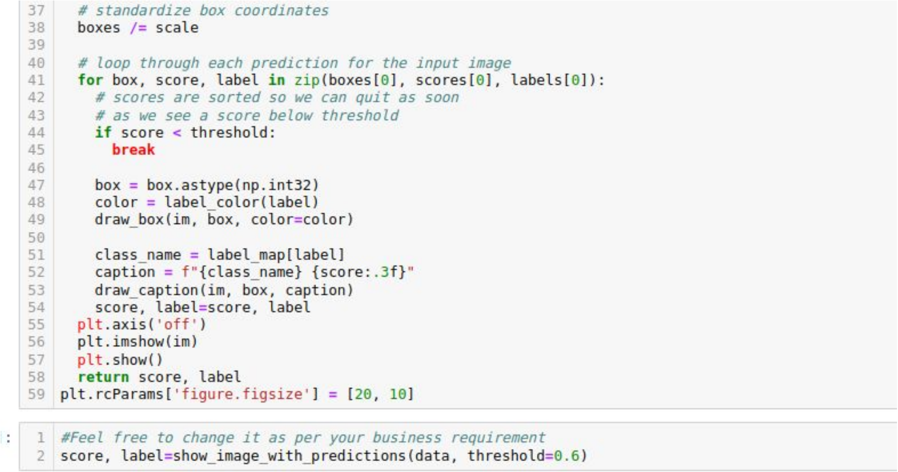
```
from keras_retinanet import models

model = models.load_model(latest_path, backbone_name='resnet50')
model = models.convert_model(model)

label_map = {}
for line in open('../maskDetectorClasses.csv'):
  row = line.rstrip().split(',')
  label_map[int(row[1])] = row[0]
```

```
Using TensorFlow backend.
```

- Model Testing : Predict using trained model



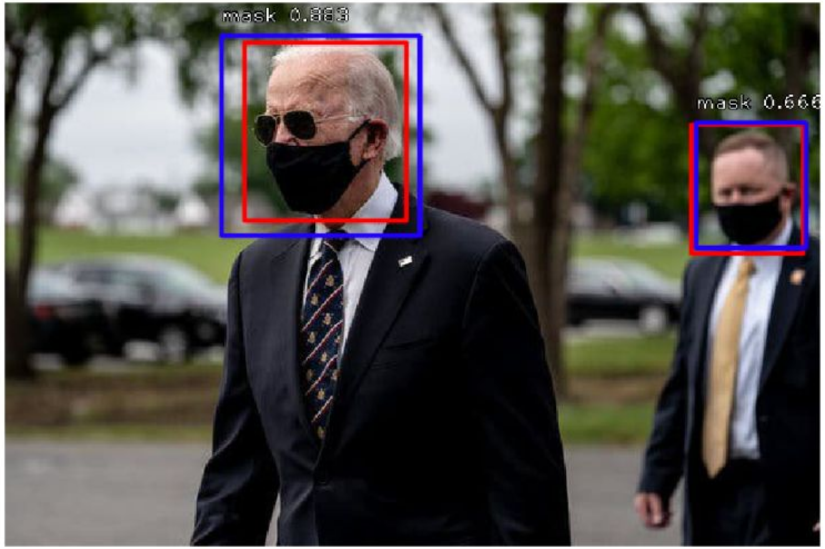
```python
def show_image_with_predictions(df, threshold=0.6):
  # choose a random image
  row = df.sample()
  filepath = row['fileName'].values[0]
  print("filepath:", filepath)

  # get all rows for this image
  df2 = df[df['fileName'] == filepath]
  im = np.array(Image.open(filepath))
  print("im.shape:", im.shape)

  # if there's a PNG it will have alpha channel
  im = im[:,:,:3]

  # plot true boxes
  for idx, row in df2.iterrows():
    box = [
      row['xmin'],
      row['ymin'],
      row['xmax'],
      row['ymax'],
    ]
    print(box)
    draw_box(im, box, color=(255, 0, 0))

  ### plot predictions ###

  # get predictions
  imp = preprocess_image(im)
  imp, scale = resize_image(im)

  boxes, scores, labels = model.predict_on_batch(
    np.expand_dims(imp, axis=0)
  )

```



```python
  # standardize box coordinates
  boxes /= scale

  # loop through each prediction for the input image
  for box, score, label in zip(boxes[0], scores[0], labels[0]):
    # scores are sorted so we can quit as soon
    # as we see a score below threshold
    if score < threshold:
      break

    box = box.astype(np.int32)
    color = label_color(label)
    draw_box(im, box, color=color)

    class_name = label_map[label]
    caption = f"{class_name} {score:.3f}"
    draw_caption(im, box, caption)
    score, label=score, label
  plt.axis('off')
  plt.imshow(im)
  plt.show()
  return score, label
plt.rcParams['figure.figsize'] = [20, 10]
```

```python
#Feel free to change it as per your business requirement
score, label=show_image_with_predictions(data, threshold=0.6)
```

```
im.shape: (400, 600, 3)
['176', '29', '295', '160']
['505', '89', '587', '184']
```



```
im.shape: (392, 696, 3)
['211', '45', '306', '148']
['429', '21', '542', '128']
```