B. TECH (CSE(AI)) (2022)

AMRITA VISHWA VIDYAPEETHAM, COIMBATORE

19AIE205 PYTHON FOR MACHINE LEARNING

OBJECT DETECTION USING R-CNN AND SVM

Date: 28.01.2022

Presented by: TEAM 04

Ameen Ashadhullah M - CB.EN.U4AIE20004

Bhoomika M - CB.EN.U4AIE20008

Ghaayathri Devi K – CB.EN.U4AIE20017

Gokul R - CB.EN.U4AIE20018

ACKNOWLEDGEMENT

*We would like to express our special thanks to our professor (**Dr. Neethu Mohan**) who gave us the golden opportunity to do this wonderful project on **OBJECT DETECTION USING R-CNN AND SVM**, which helped us in doing a lot of research and we came to know about so many new things. We owe our gratitude to our professor for continued support and encouragement. We offer our sincere appreciation for the learning opportunities provided, which is moulding us to go beyond our limits and dream for the best. We are thankful for the opportunity given.*

TABLE OF CONTENTS

ABSTRACT

Deep learning has gained an amazing impact on how the world adapts to Artificial Intelligence. The field of computer vision and discovery continues to evolve. Computer Vision is a field of research that helps to develop visual imagery and display techniques. It has different features such as image recognition, object detection and image creation, etc. Object detection is used for face detection, vehicle detection, web images, and security systems. Many algorithms and modules came along, Deep learning is now a mainstream method of object detection. Regional Based convolutional neural networks (R-CNN) networks have an important role to play in in-depth learning. It has amazing results to find in regular forums. However, under special circumstances, there may still be unsatisfactory performance of the detection, such as something with problems such as closure, deterioration, or small size. In this project, we used Region-based Convolutional Neural Networks (R-CNN) AND Support Vector Machine(SVM) for object detection. By using the selective search segmentation algorithm and annotation datasets we train the neural network model with Support Vector Machines. So finally we will get the plot of loss and accuracy. Then we will test the model, by importing a random image and predicting it, it returns the plot of image with a boundary box over the object in the image.

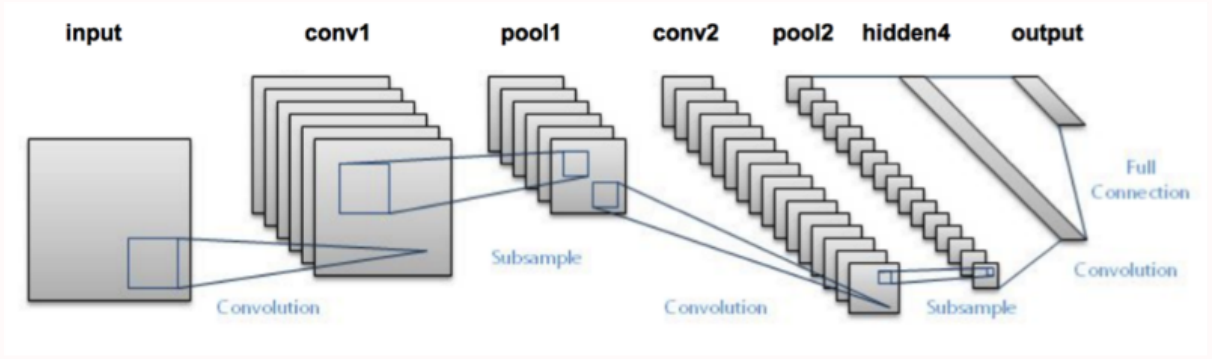
INTRODUCTION

We as humans on a day to day basis see different things and identify them , whether it is an object or another human or an animal etc. Furthermore we are even able to perfectly name all the things we see. This happens because of the neural network which allows our brain to be able to identify , classify objects. When it comes to Machine or a Computer, it has the ability to identify the image files as jpg , jpge , png , etc. But it does not have the ability to read those images and identify different objects in it. So in order for the machine to be able to do that we apply CNN concept which helps us to train the machine to identify the objects.

A machine learning approach and a deep learning approach can both be used to detect objects. The machine learning strategy necessitates the definition of features using multiple approaches, followed by classification using any technique, such as Support Vector Machines (SVMs). The deep learning approach, on the other hand, allows the entire detection process to be completed without explicitly identifying the features that will be used to classify the data. Convolutional Neural Networks are at the heart of the deep learning approach (CNNs).

CONVOLUTIONAL NEURAL NETWORK

A Convolutional Neural Network is a powerful neural network that uses filters to extract features from images. It also does so in such a way that position information of pixels is retained. A convolution is a mathematical operation applied on a matrix. This matrix is usually the image represented in the form of pixels/numbers. The convolution operation extracts the features from the image.



We pass an image to the network, and it is then sent through various convolutions and pooling layers. Finally, we get the output in the form of the object's class. For each input image, we get a corresponding class as an output. Let us see how to implement object detection using CNN.

Step1: Take any image as input

Step2: Divide the image into various regions

Step 3: We will then consider each region as a separate image.

Step 4: Pass all these regions (images) to the CNN and classify them into various classes.

Step 5: Once we have divided each region into its corresponding class, we can combine all these regions to get the original image with the detected objects.

The problem with using this approach is that the objects in the image can have different aspect ratios and spatial locations. For instance, in some cases the object might be covering most of the image, while in others the object might only be covering a small percentage of the image. The shapes of the objects might also be different (happens a lot in real-life use cases).

As a result of these factors, we would require a very large number of regions resulting in a huge amount of computational time. So to solve this problem and reduce the number of regions, we can use region-based CNN, which selects the regions using a proposal method.

R-CNN

Instead of working on a massive number of regions, the RCNN algorithm proposes a bunch of boxes in the image and checks if any of these boxes contain any object. RCNN uses selective search to extract these boxes from an image (these boxes are called regions).

Selective Search Algorithm

There are basically four regions that form an object:

1. Varying scales
2. Colours
3. Textures
4. Enclosure

This algorithm identifies these patterns in the image and based on that, proposes various regions. Let us see how this algorithm works now.

Step1: Takes an image as input

Step2: Generates initial sub-segmentations so that we have multiple regions from this image

Step3: The technique then combines the similar regions to form a larger region (based on colour similarity, texture similarity, size similarity, and shape compatibility)

Step4: Finally, these regions then produce the final object locations (Region of Interest)

Now let us see the total way to detect objects using RCNN. We first take a pre-trained convolutional neural network.

1. Then, this model is retrained. We train the last layer of the network based on the number of classes that need to be detected.
2. The third step is to get the Region of Interest for each image. We then reshape all these regions so that they can match the CNN input size.

3. After getting the regions, we train SVM to classify objects and backgrounds. For each class, we train one binary SVM.
4. Finally, we train a linear regression model to generate tighter bounding boxes for each identified object in the image.

SUPPORT VECTOR MACHINE(SVM)

It is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. We use this SVM in object detection in our case. After RCNN maps the data into classes. It is SVM that classifies them into distinct objects.

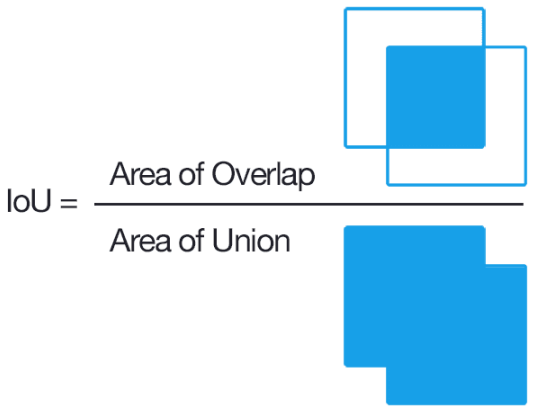
CONCEPT OF INTERSECTION OVER UNION

Intersection over Union is a statistic for determining how accurate an item detector is on a given dataset. IoU can be used to evaluate any method that produces predicted bounding boxes as an output.

In order to assess a (arbitrary) object detector using Intersection over Union, we'll need:

- Bounding boxes based on ground truth (i.e., the hand labelled bounding boxes from the testing set that specify where in the image our object is).
- The bounding boxes predicted by our model.

Computing Intersection over Union can therefore be determined via:



- ☐ When you look at this equation, you'll notice that Intersection over Union is just a ratio.
- ☐ We compute the area of overlap between the predicted and ground-truth bounding boxes in the numerator.
- ☐ The area of union, or, to put it another way, the area covered by both the anticipated and ground-truth bounding boxes, is the numerator.

- [ ] Our final score — the Intersection over Union — is calculated by dividing the amount of overlap by the area of union.

ALGORITHM AND CODE EXPLANATION

- ➔ We are using two datasets, image dataset which have the images of aeroplanes and the second dataset will have the annotation excel files that have the coordinates to draw the boundary boxes over the aeroplane in the corresponding image.
- ➔ We will use the selective search algorithm to find the region of interest in an image.
- ➔ We will find the ROI of each image and will store the values in an array.
- ➔ We will consider only 2000 ROI per image
- ➔ Now we will find the IOU for boundary box (BB1) given in data set and Boundary box (BB2) of each ROI
- ➔ Now for each image we will take the 30 positive counters(IOU > 0.7) and 30 negative counters(IOU > 0.3)
- ➔ For those samples we will append the resized image (that particular region of counter) in train_image and append the train_label
- ➔ Using vgg16 we will create and train a neural network model by keeping the last 2 layers non trainable.
- ➔ We will initialise svm_image and svm_label
- ➔ Now we will take all the annotated box in the dataset as the counter and we will take the some false counter and like we already done, now will append the svm_image and svm _label.
- ➔ We will add the SVM layer in the last 2 layers of model and train the model
- ➔ We can get the Loss, Validation Loss and Accuracy in the training time
- ➔ Finally we will test the model with a random image and get the output with a boundary box over the aeroplane in the image.

Importing the required modules

```
In [1]: import os  #PROVIDES FUNCTION FOR INTERACTING WITH OPERATING SYSTEM
        import cv2 #FOR HANDLING THE IMAGE AND IMAGE PROESSING
        import keras #OPEN SOURCE NEURAL NETWORK WORKS ON TENSOR FLOW
        import pandas as pd #DATA SCIENCE AND DATA ANALYTICS
        import matplotlib.pyplot as plt #PLOTTING AND GRAPHICAL
        import numpy as np #BASIC MATHEMATICAL COMPUTATION
        import tensorflow as tf #HELPS TO CREATE A DEEP LEARNING MODEL
        import h5py #TO HANDLE HDF5 FILES
```

Defining the path to access the datasets

```
In [2]: path = "C:/Users/ameen/Documents/AMRITA/3rd sem/PML/____project/rcnn/final/Images
        annot = "C:/Users/ameen/Documents/AMRITA/3rd sem/PML/____project/rcnn/final/Airpl
```

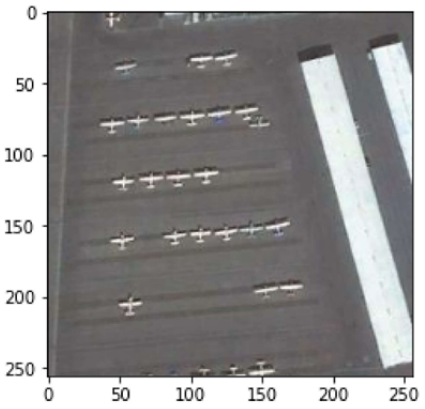
SHOWING AN EXAMPLE OF THE PREDICTION USING R-CNN

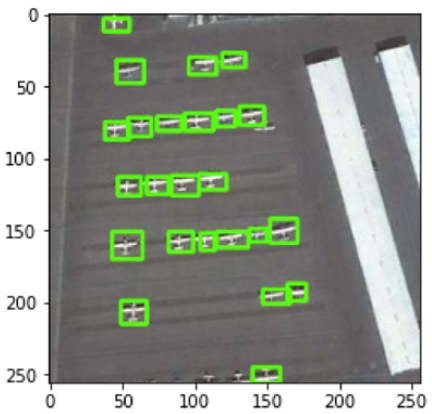
Importing the image file and annotation file(have coordinates of diagonals of the boundary box) and plotting the image with the boundary box given in dataset

In [3]:
```
#Intalizing the index with the image no.
Index=170
#Concatenating the file name
filename = "airplane_"+str(Index)+".jpg"
print(filename)

#Reading the image in the file path specified
img = cv2.imread(os.path.join(path,filename))
#Reading the Annotations file in the path specified
df = pd.read_csv(os.path.join(annot,filename.replace(".jpg",".csv")))
#printing the image
plt.imshow(img)

#Getting the x and y coordinates for the box from the Annotation file
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),(85,300,10), 2)

#Plotting the Box around the object
plt.figure()
plt.imshow(img)
```

```
airplane_170.jpg
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x1b517dc8af0>
```





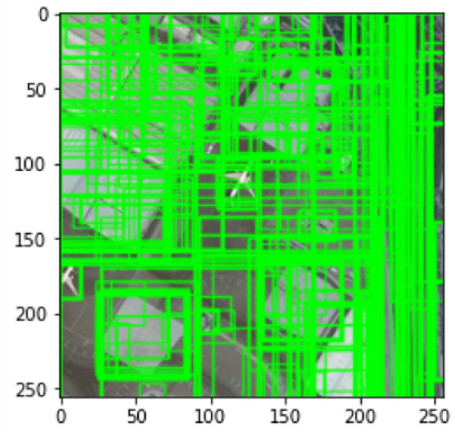
NOW LETS SEE THE OUTPUT OF AN SELECTIVE SEARCH SEGMENTATION BY USING THE CV2 MODULE:

In [4]:
```
cv2.setUseOptimized(True);
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
```

5
```
#Reads the image file frm the specified path
im = cv2.imread(os.path.join(path,"42850.jpg"))
#Set a image used by switch* functions to initialize the class.
ss.setBaseImage(im)
#
ss.switchToSelectiveSearchFast()

#Based on all images, graph segmentations and stragies, computes all possible rec
rects = ss.process()
print(rects)
# copies the image to imOut
imOut = im.copy()

#Gets all the rects coordinates and plots them.
for i, rect in (enumerate(rects)):
    x, y, w, h = rect
    cv2.rectangle(imOut, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
plt.imshow(imOut)
```

```
[[209   0  47 256]
 [  0   0  37  29]
 [  0  63  47  29]
 ...
 [169   0  87 256]
 [145   0  79 144]
 [  0   0 109 164]]
```

Out[5]: <matplotlib.image.AxesImage at 0x1b517ef63d0>



Here we can see that there are N boxes we got as output as REGION OF INTEREST , but we will take only the top 2000 regions for checking.

This get_iou function helps us to find the Intersection Over Union of BB1 and BB2

```
6]: 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
```

Initialising the variable train_image and train_label

```
In [7]: # At the end of below code we will have our train data in these lists
        train_images=[]
        train_labels=[]
```

In this for loop it will import the image and annotation file and the stores the values of coordinates into a variable x1, y1, x2, y2 and assigning into the dictionary “gtvalues” Now image will undergoes selective search segmentation and then gets the results to be stored in “ssresult” which will have the 4 coordinates of the boundary box of ROI.

Then we will get the IOU value of the function between bb1 and bb2 using the get_iou function. After that we will check it is positive counter or negative counter and will take each of 30 samples per image and we will append the train_image and train_label

```
8   for e,i in enumerate(os.listdir(annot)):
        try:
            # If the file name starts with "airplane"
            if i.startswith("airplane"):
                # The .csv in the file extension is replaced with .jpg
                filename = i.split(".")[0]+".jpg"
                #printing the filename
                print(e,filename)
                #Reading the image in the file path specified
                image = cv2.imread(os.path.join(path,filename))
                #Reading the Annotations file in the path specified
                df = pd.read_csv(os.path.join(annot,i))
                gtvalues=[]
```

```
        #Getting the x and y coordinates for the box from the Annotation file
        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])
            #Appending the x and y coordinates in a dictnoary
            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 bo
        ssresults = ss.process()     # processing to get the outputs
        #Copies the image object to imout
        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 regior
                for gtval in gtvalues:
                    x,y,w,h = result
                    iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})  #
                    if counter < 30:       # getting only 30 psoitive example
                        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), interpola
                            train_images.append(resized)
                            train_labels.append(1)
                            counter += 1
                    else :
                        fflag =1              # to insure we have collected a
                    if falsecounter <30:      # 30 negatve examples are allow
                        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), interpola
                            train_images.append(resized)
                            train_labels.append(0)
                            falsecounter += 1
                    else :
                        bflag = 1             #to ensure we have collected al
                if fflag == 1 and bflag == 1:
```



```
                    print("inside")
                    flag = 1        # to signal the complition of data extact
    except Exception as e:
        print(e)
        print("error in "+filename)
        continue
```

```
17 airplane_001.jpg
inside
18 airplane_002.jpg
19 airplane_003.jpg
20 airplane_004.jpg
inside
21 airplane_005.jpg
```

```
21 airplane_005.jpg
inside
22 airplane_006.jpg
23 airplane_007.jpg
inside
24 airplane_008.jpg
25 airplane_009.jpg
26 airplane_010.jpg
27 airplane_011.jpg
28 airplane_012.jpg
inside
29 airplane_013.jpg
30 airplane_014.jpg
```

Now we will start the training process, assigning the train data into an array X_new and Y_new.

In [9]:
```
# conversion of train data into arrays for further training
X_new = np.array(train_images)
Y_new = np.array(train_labels)
```

Importing the required model for training the neural network and here we are using vgg16 neural network to create and train the model. And as we already mentioned, we make the last 2 layers of model as non trainlable layer.

In [10]:
```
from keras.layers import Dense
from keras import Model
from keras import optimizers
```

In [11]:
```
# For object detection and classification through layerwise computation
vgg = tf.keras.applications.vgg16.VGG16(include_top=True, weights='imagenet', inp
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'])
```

Getting summary and fitting and training the model

12

```
# The summary can be created by calling the summary() function on the model that
# The summary is textual and includes information about:

# The layers and their order in the model.
# The output shape of each layer.
# The number of parameters (weights) in each layer.
# The total number of parameters (weights) in the model.

model.summary()

# Model fitting is a measure of how well a machine learning model generalizes to

model.fit(X_new,Y_new,batch_size = 64,epochs = 3, verbose = 1,validation_split=.
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
```

```
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 dense (Dense)               (None, 1)                 4097

=================================================================
Total params: 134,264,641
Trainable params: 16,785,409
Non-trainable params: 117,479,232
_________________________________________________________________
Epoch 1/3
449/449 [==============================] - 5041s 11s/step - loss: 0.6318 - acc:
0.8108 - val_loss: 0.5075 - val_acc: 0.8016
Epoch 2/3
449/449 [==============================] - 4685s 10s/step - loss: 0.3339 - acc:
0.8530 - val_loss: 0.5479 - val_acc: 0.7844
Epoch 3/3
449/449 [==============================] - 4232s 9s/step - loss: 0.2860 - acc:
0.8764 - val_loss: 0.5598 - val_acc: 0.7890
```

Out[12]: <keras.callbacks.History at 0x1b5192691c0>

Assigning the variable for the SVM dataset

In [13]:
```
svm_image = [];
svm_label = [];
```

In the upcoming for loop it will import the image and annotation file and the stores the values of coordinates into a variable x1, y1, x2, y2 and assigning into the dictionary "gtvalues"
Now image will undergoes selective search segmentation and then gets the results to be stored in "ssresult" which will have the 4 coordinates of the boundary box of ROI.

Then we will get the IOU value of the function between bb1 and bb2 using the get_iou function. After that we will check if it is a positive counter or negative counter and will take a few samples of false counters per image and it will all the boundary regions given in the dataset as positive counters. Thenwe will append the svm_image and svm_label

14
```
for e,i in enumerate(os.listdir(annot)):
    try:
        # If the file name starts with "airplane"
        if i.startswith("airplane"):
            # The .csv in the file extension is replaced with .jpg
            filename = i.split(".")[0]+".jpg"
            #Printing the file names
            print(e,filename)
            #Reading the image in the file path specified
            image = cv2.imread(os.path.join(path,filename))
            #Reading the Annotations file in the path specified
            df = pd.read_csv(os.path.join(annot,i))
            gtvalues=[]
```

```
        #Getting the x and y coordinates for the box from the Annotation file
        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])
            #Appending the x and y coordinates in a dictnoary
            gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})
            timage = image[x1:x2,y1:y2]
            #Resizing All the images to a defined size
            resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER
            #Appending the images to the svm_image
            svm_image.append(resized)
            #Appending the labels of the image to svm_image
            svm_label.append([0,1])
         # setting given image as base image
        ss.setBaseImage(image)
        # running selective search on bae image
        ss.switchToSelectiveSearchFast()
        # processing to get the outputs
        ssresults = ss.process()
        #Copies the image object to imout
        imout = image.copy()
        counter = 0
        falsecounter = 0
        flag = 0
        for e,result in enumerate(ssresults):
            if e < 2000 and flag == 0: # till 2000 to get top 2000 regions or
                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: # getting only 5 negative examples
                        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), interpola
                            #Appending the images to the svm_image
                            svm_image.append(resized)
                            #Appending the labels of the image to svm_image
                            svm_label.append([1,0])
                            falsecounter += 1
                    else :
                        flag = 1
except Exception as e:
```

0

```
        print(e)
        print("error in "+filename)
        continue
```

```
17 airplane_001.jpg
18 airplane_002.jpg
19 airplane_003.jpg
20 airplane_004.jpg
21 airplane_005.jpg
22 airplane_006.jpg
23 airplane_007.jpg
24 airplane_008.jpg
25 airplane_009.jpg
26 airplane_010.jpg
```

For each time to train the models it takes more time, so we are saving the model in saved format and the .h5 models and we are loading the model into a new variable then checking if both trained model and the loaded model are same or not.

saving model

In [15]:
```
model.save("my_model")
```

```
INFO:tensorflow:Assets written to: my_model\assets
```

In [16]:
```
model.save("my_h5_model.h5")
```

In [17]:
```
l_model_h5 = keras.models.load_model("my_h5_model.h5")
l_model = keras.models.load_model("my_model")
```

In [18]:
```
print(l_model.get_config() == model.get_config())
```

```
True
```

In [19]:
```
print(l_model_h5.get_config() == model.get_config())
```

```
True
```

Now we are adding the svm layers in the last layers of model and training it

0
```
#adding svm to last layer,we will remove the last layer and as the last 2 models
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_h5_model.h5', by_name=True)
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
```

```
block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

flatten (Flatten)           (None, 25088)             0

fc1 (Dense)                 (None, 4096)              102764544
```

2

```
 fc2 (Dense)                (None, 4096)              16781312

 dense_1 (Dense)            (None, 2)                 8194

=================================================================
Total params: 134,268,738
Trainable params: 16,789,506
Non-trainable params: 117,479,232
_________________________________________________________________
```

In [ ]:

Now fitting and training the final model

```
1  # finally we are fitting and train the model with 20 epochs
   hist_final = final_model.fit(np.array(svm_image),np.array(svm_label),batch_size=
```

```
Epoch 1/20
167/167 [==============================] - 695s 4s/step - loss: 0.6659 - accu
racy: 0.7294 - val_loss: 0.7504 - val_accuracy: 0.6263
Epoch 2/20
167/167 [==============================] - 928s 6s/step - loss: 0.5284 - accu
racy: 0.7713 - val_loss: 0.7361 - val_accuracy: 0.6370
Epoch 3/20
167/167 [==============================] - 1177s 7s/step - loss: 0.4724 - acc
uracy: 0.8000 - val_loss: 0.7358 - val_accuracy: 0.6655
Epoch 4/20
167/167 [==============================] - 1082s 6s/step - loss: 0.4336 - acc
uracy: 0.8219 - val_loss: 0.7683 - val_accuracy: 0.7117
```

```
Epoch 5/20
167/167 [==============================] - 720s 4s/step - loss: 0.3868 - accu
racy: 0.8406 - val_loss: 0.7403 - val_accuracy: 0.6940
Epoch 6/20
167/167 [==============================] - 732s 4s/step - loss: 0.3544 - accu
racy: 0.8539 - val_loss: 0.7621 - val_accuracy: 0.7438
Epoch 7/20
167/167 [==============================] - 727s 4s/step - loss: 0.3175 - accu
racy: 0.8691 - val_loss: 0.7482 - val_accuracy: 0.7117
Epoch 8/20
167/167 [==============================] - 732s 4s/step - loss: 0.2963 - accu
racy: 0.8820 - val_loss: 0.8463 - val_accuracy: 0.7082
Epoch 9/20
167/167 [==============================] - 722s 4s/step - loss: 0.2829 - accu
racy: 0.8867 - val_loss: 0.8196 - val_accuracy: 0.6833
Epoch 10/20
167/167 [==============================] - 721s 4s/step - loss: 0.2667 - accu
racy: 0.8936 - val_loss: 0.8402 - val_accuracy: 0.7011
Epoch 11/20
167/167 [==============================] - 722s 4s/step - loss: 0.2405 - accu
racy: 0.9030 - val_loss: 1.1973 - val_accuracy: 0.6157
Epoch 12/20
167/167 [==============================] - 721s 4s/step - loss: 0.2357 - accu
racy: 0.9075 - val_loss: 0.8696 - val_accuracy: 0.7082
Epoch 13/20
167/167 [==============================] - 720s 4s/step - loss: 0.2059 - accu
racy: 0.9191 - val_loss: 0.7400 - val_accuracy: 0.7189
Epoch 14/20
167/167 [==============================] - 718s 4s/step - loss: 0.1924 - accu
racy: 0.9256 - val_loss: 0.8584 - val_accuracy: 0.6940
Epoch 15/20
167/167 [==============================] - 724s 4s/step - loss: 0.1804 - accu
racy: 0.9277 - val_loss: 0.8546 - val_accuracy: 0.7011
Epoch 16/20
167/167 [==============================] - 715s 4s/step - loss: 0.1722 - accu
racy: 0.9344 - val_loss: 0.8064 - val_accuracy: 0.7189
Epoch 17/20
167/167 [==============================] - 717s 4s/step - loss: 0.1763 - accu
racy: 0.9301 - val_loss: 0.9074 - val_accuracy: 0.7046
Epoch 18/20
```



```
167/167 [==============================] - 718s 4s/step - loss: 0.1536 - accu
racy: 0.9427 - val_loss: 0.9582 - val_accuracy: 0.7295
Epoch 19/20
167/167 [==============================] - 717s 4s/step - loss: 0.1341 - accu
racy: 0.9500 - val_loss: 0.8787 - val_accuracy: 0.7616
Epoch 20/20
167/167 [==============================] - 718s 4s/step - loss: 0.1558 - accu
racy: 0.9388 - val_loss: 0.8846 - val_accuracy: 0.7438
```

Saving the final model

saving final model

In [22]:
```
final_model.save("fin_model")
final_model.save("fin_h5_model.h5")
```

```
INFO:tensorflow:Assets written to: fin_model\assets
```

In [23]:
```
l_fin_model_h5 = keras.models.load_model("fin_h5_model.h5")
l_fin_model = keras.models.load_model("fin_model")
```

In [24]:
```
print(l_fin_model.get_config() == final_model.get_config())
```

```
True
```

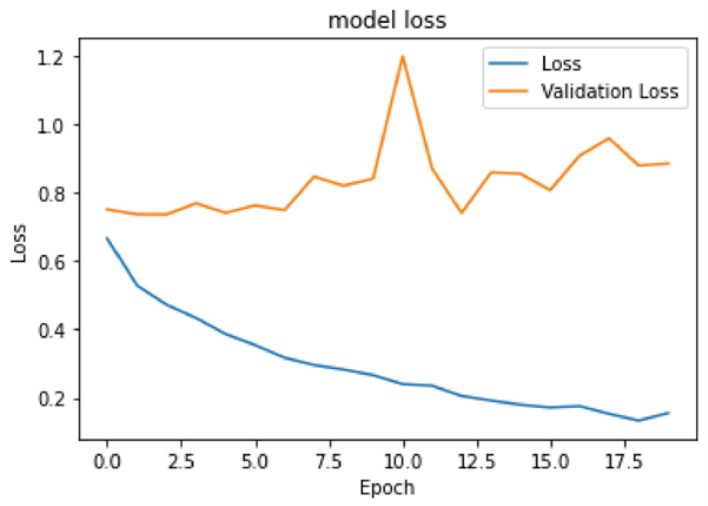
In [25]:
```
print(l_fin_model_h5.get_config() == final_model.get_config())
```

```
True
```

In [26]:
```
### PLOTTING
```

Plotting

7
```
import matplotlib.pyplot as plt
plt.plot(hist_final.history['loss'])
plt.plot(hist_final.history['val_loss'])
plt.title("model loss")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()
plt.savefig('chart loss.png')
```




```
<Figure size 432x288 with 0 Axes>
```

NOW IT'S TIME TO TEST THE MODEL

We will test the model with some random image having the aeroplanes. It will detect and plot the image having a boundary box on the aeroplanes present in the image.

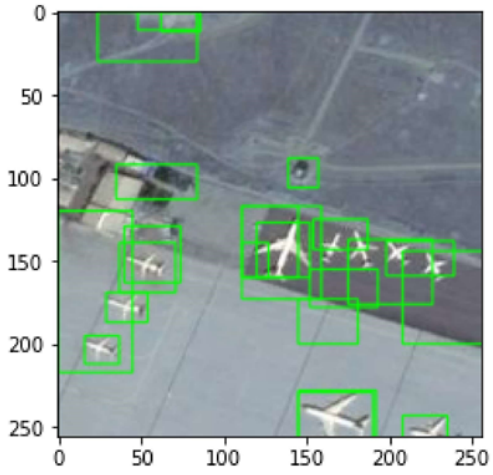
In [18]:
```
###########  it's time for test a image    ##########
image = cv2.imread(os.path.join(path,'airplane_020.jpg'))
ss.setBaseImage(image)
ss.switchToSelectiveSearchFast()
ssresults = ss.process()

imOut = image.copy()
boxes = []
count = 0
for e,result in enumerate(ssresults):
    if e < 50:
        x,y,w,h = result
        timage = imout[x:x+w,y:y+h]
        resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
        resized = np.expand_dims(resized,axis = 0)
        out = l_fin_model.predict(resized)
        print(e,out)
        if(out[0][0]<out[0][1]):
            boxes.append([x,y,w,h])
            count+=1

for box in boxes:
    x, y, w, h = box
    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)
```

```
0 [[-0.95339614  0.9962526 ]]
1 [[ 6.309066 -6.156432]]
2 [[ 1.8611535 -1.6347631]]
3 [[-0.23102656  0.26116407]]
4 [[ 3.4487553 -3.3755305]]
5 [[-0.5596184  0.5770244]]
6 [[ 5.1030517 -4.864478 ]]
7 [[-0.3411736   0.34123874]]
8 [[-0.06704099  0.09032804]]
9 [[-5.07469    5.0534225]]
10 [[-0.3951628   0.41282725]]
11 [[ 6.3888087 -6.4570675]]
12 [[ 0.775805   -0.75543576]]
13 [[ 6.5270653 -6.3699455]]
14 [[ 3.361725  -3.2180521]]
15 [[-4.792223  4.791295]]
16 [[-0.09838206  0.11770827]]
17 [[-1.2168206  1.2390999]]
18 [[-1.9806737  2.0325627]]
19 [[ 7.0926423 -7.1581078]]
20 [[-0.7415333   0.92859143]]
21 [[ 11.268065 -11.366391]]
22 [[ 0.6937352  -0.67388177]]
23 [[ 6.813528 -6.652695]]
24 [[ 1.1680994 -1.1515431]]
25 [[ 8.931741 -8.786428]]
26 [[-0.5728311   0.56407946]]
```

0

```
27 [[-0.9796781  1.0074323]]
28 [[-3.1895604  3.240692 ]]
29 [[-4.2412596  4.276059 ]]
30 [[ 2.8607206 -2.6603153]]
31 [[-4.276557  4.303374]]
32 [[-6.8356543  6.8881183]]
33 [[ 7.262119  -7.1240034]]
34 [[-0.5683039   0.57593066]]
35 [[-0.14719597  0.1602742 ]]
36 [[ 1.2746645 -1.2590009]]
37 [[ 0.6093295  -0.62037283]]
38 [[ 2.2613387 -2.168457 ]]
39 [[ 5.5859857 -5.458085 ]]
40 [[-0.82442355  0.84640616]]
41 [[ 13.277257 -13.142993]]
42 [[ 0.07284192 -0.04677291]]
43 [[ 15.52557  -15.725347]]
44 [[ 0.16647762 -0.14384276]]
45 [[-1.0584882  1.0967635]]
46 [[ 0.7046981 -0.6908234]]
47 [[ 0.72288686 -0.46239263]]
48 [[-0.3353633   0.36689255]]
49 [[ 8.262816  -7.9009385]]
198 138 41 21
152 155 41 23
29 169 25 18
111 139 16 21
16 195 21 17
24 0 60 30
145 228 47 28
120 127 32 33
35 92 49 21
62 0 24 12
111 117 34 43
0 120 45 97
139 88 18 18
37 139 34 30
175 137 51 39
145 173 36 27
208 144 48 56
111 117 48 56
48 0 36 11
145 229 45 27
40 129 34 34
208 243 27 13
154 125 33 18
```

Out[18]: <matplotlib.image.AxesImage at 0x18259db3a60>


0
50
100
150
200
250
0
50
100
150
200
250

LIMITATIONS

One disadvantage of Faster R-CNN is that the RPN is trained using a single image to extract all anchors in the mini-batch of size 256. The network may take a long time to attain convergence because all samples from a single image may be correlated (i.e. their features are similar).

- Using a selective search, we were able to extract 2,000 zones for each image.
- For each image region, CNN is used to extract features. The amount of CNN features will be N*2,000 if we have N images.
- The complete object detection procedure using RCNN is divided into three models:
    - CNN is used to extract features.
    - Object recognition with a linear SVM classifier
    - For tightening the bounding boxes, use a regression model.
- RCNN is extremely sluggish as a result of all of these procedures. When presented with a massive dataset, it takes roughly 40-50 seconds to produce predictions for each new image, making the model unwieldy and almost impossible to create.

CONCLUSION

Using RCCN we have successfully implemented the Algorithm which classifies and detects the objects in an image. The accuracy and precision of the detection was found to be more than average. Here we have trained the model using Annotations dataset with multiple iterations. And tested it using a few other images , while testing the algorithm was able to identify and detect the positions of the aeroplanes placed in the image and draw the bounding boxes to it.