Import Necessary Libraries

In [1]:

```
import numpy as np # For numerical computation
import os # To list the directory/files present with provided path parameter
import shutil # To transfer the list of images into train and val folder
import re # For finding subset string pattern
import cv2 # for selective search algorithm
import matplotlib.pyplot as plt # For showing the image
import json # To store all images annotation contains BB and their labels into json format
```

Explore Dataset

Let explore dataset directory which is provided by PASCAL VOC. (Homepage Link: http://host.robots.ox.ac.uk/pascal/VOC/)

In this notebook, we are considering **VOC2005 Challenge** Dataset (link: http://host.robots.ox.ac.uk/pascal/VOC/databases.html#VOC2005_1).

In [2]:

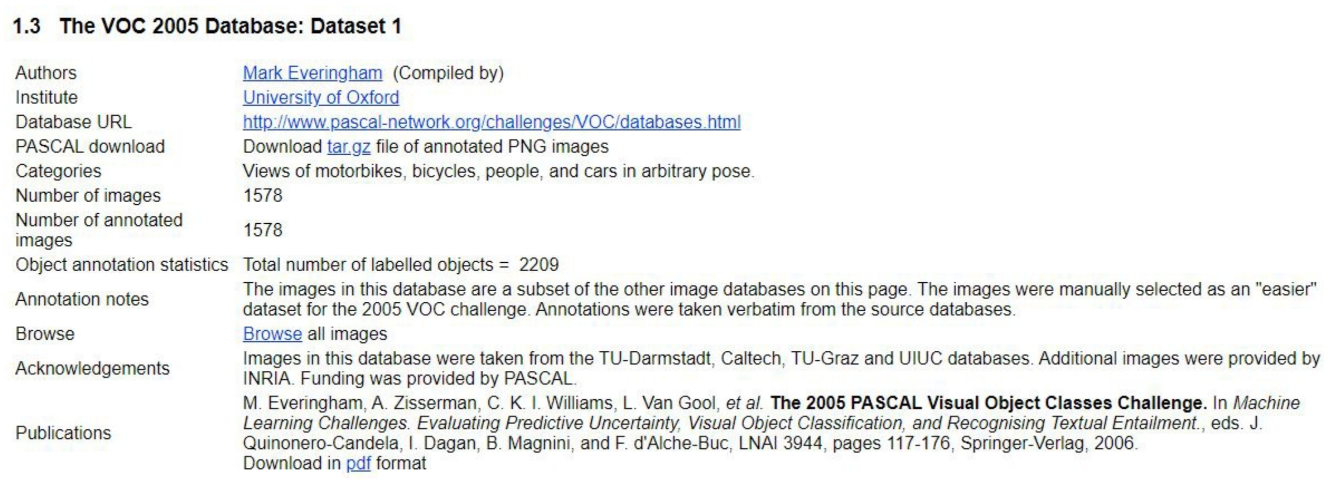
```
trainDir = 'TrainDataSet/VOCdevkit/VOC2006/'
testDir = 'TestDataSet/VOCdevkit/VOC2006/'
```

In [3]:

```
print('List of directory present in TrainDataset',os.listdir(trainDir))
```

```
List of directory present in TrainDataset ['Annotations', 'contrib.txt', 'ImageSets', 'PNGImages']
```

Description about dataset:



1.3 The VOC 2005 Database: Dataset 1

Authors	Mark Everingham (Compiled by)
Institute	University of Oxford
Database URL	http://www.pascal-network.org/challenges/VOC/databases.html
PASCAL download	Download tar.gz file of annotated PNG images
Categories	Views of motorbikes, bicycles, people, and cars in arbitrary pose.
Number of images	1578
Number of annotated images	1578
Object annotation statistics	Total number of labelled objects = 2209
Annotation notes	The images in this database are a subset of the other image databases on this page. The images were manually selected as an "easier" dataset for the 2005 VOC challenge. Annotations were taken verbatim from the source databases.
Browse	Browse all images
Acknowledgements	Images in this database were taken from the TU-Darmstadt, Caltech, TU-Graz and UIUC databases. Additional images were provided by INRIA. Funding was provided by PASCAL.
Publications	M. Everingham, A. Zisserman, C. K. I. Williams, L. Van Gool, *et al.* **The 2005 PASCAL Visual Object Classes Challenge.** In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment.*, eds. J. Quinonero-Candela, I. Dagan, B. Magnini, and F. d'Alche-Buc, LNAI 3944, pages 117-176, Springer-Verlag, 2006. Download in pdf format

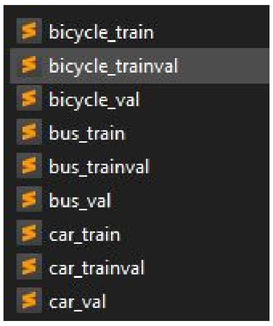
Inside Data

After observing all the directories and files (directory each of directories), the following are:

ImageSets folder

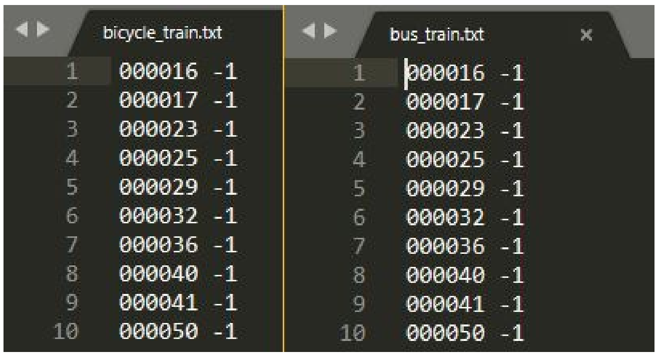
It contain list of text (which is label name) that created for train and val image. Inside the text, there is two columns: **imageName** and **their corresponding class label**.

These are list of files inside ImageSets


bicycle_train
bicycle_trainval
bicycle_val
bus_train
bus_trainval
bus_val
car_train
car_trainval
car_val

Each of text name is considered as **labelname with corresponding type of dataset (train/val)**.

Now if we go and read two files which is appended with train: **bicycle_train** and **bus_train**


bicycle_train.txt

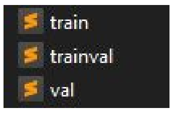
```
000016 -1
000017 -1
000023 -1
000025 -1
000029 -1
000032 -1
000036 -1
000040 -1
000041 -1
000050 -1
```

bus_train.txt

```
000016 -1
000017 -1
000023 -1
000025 -1
000029 -1
000032 -1
000036 -1
000040 -1
000041 -1
000050 -1
```

You can see, there are two columns which indicates **Imagename** and their corresponding **label value**. For example: Reading from bicycle_train.text file, in the above image, **000016 -1** indicating that for this **imagename 000016**, there is **no bicycle exist**. Similarily, you can read from bus_train.text, its same as it implies as for this **imagename 000016**, there is **no bus exist**

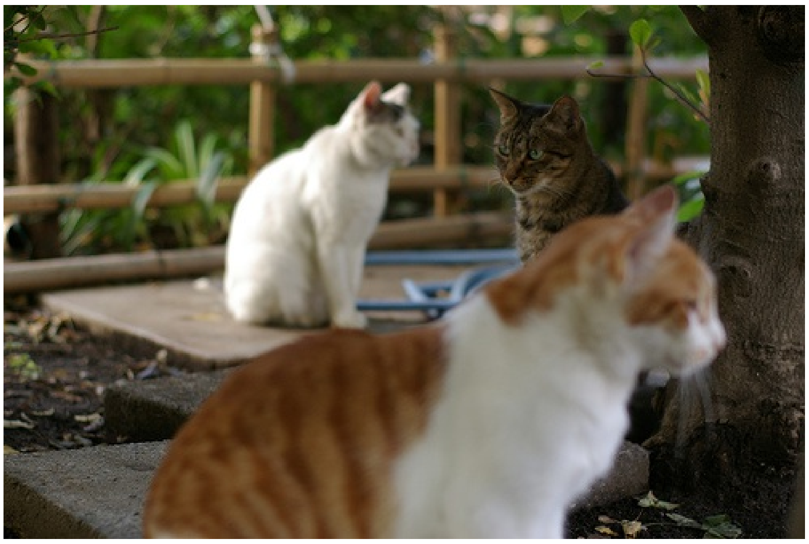
They create separate text files which contains list of image name whether these images belong to train data or valid data.


train
trainval
val

PNGImages folder

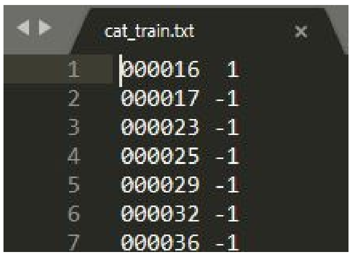
It contains list of images presents in it.

With the sample illustration above, let see with image name as **000016.png**


00016.png

So, you see, there is **no bicycle and bus** present in it.

And we should expect that this images must be present in cat_train.txt, right? Let chek it out.



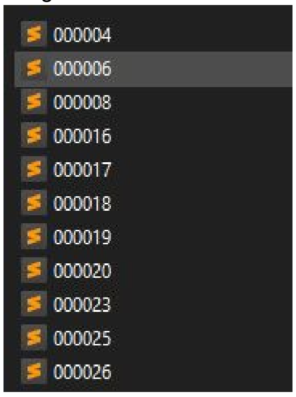
```
000016  1
000017 -1
000023 -1
000025 -1
000029 -1
000032 -1
000036 -1
```

Yes!! There is labelled as 1 in cat_train.txt.

.

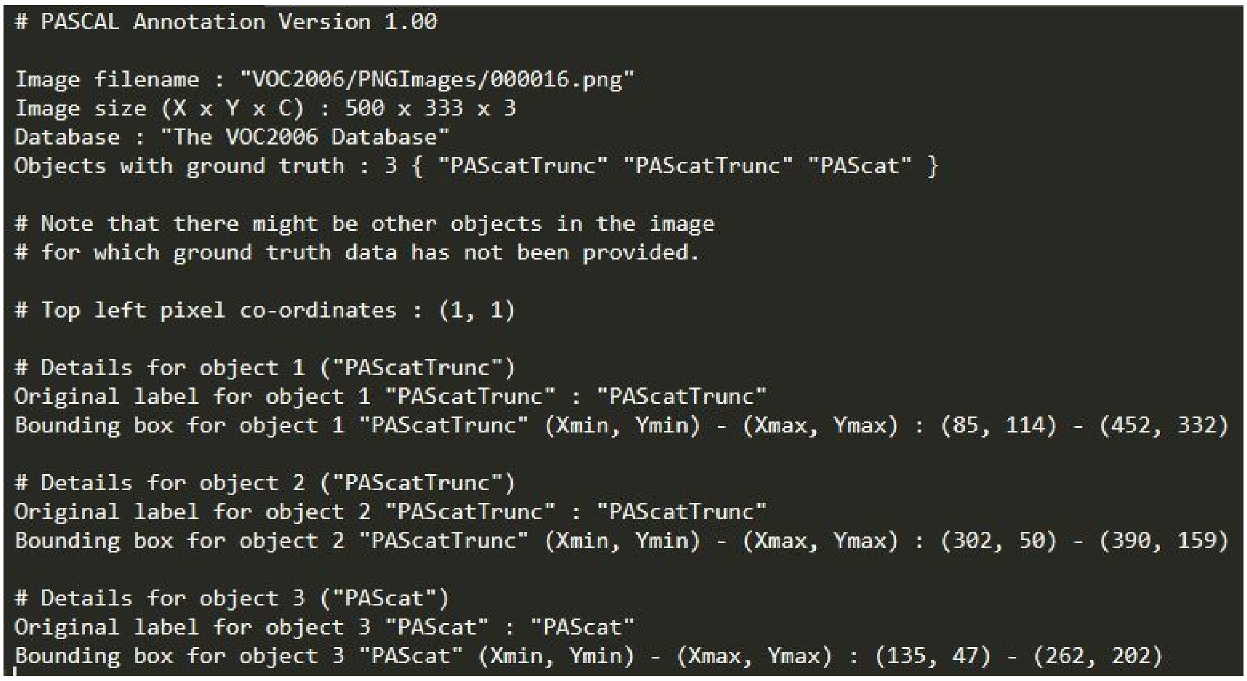
Annotation folder

It contain about object location of an object in each image.



```
000004
000006
000008
000016
000017
000018
000019
000020
000023
000025
000026
```

Let open with same file name: 000016



```
# PASCAL Annotation Version 1.00

Image filename : "VOC2006/PNGImages/000016.png"
Image size (X x Y x C) : 500 x 333 x 3
Database : "The VOC2006 Database"
Objects with ground truth : 3 { "PAScatTrunc" "PAScatTrunc" "PAScat" }

# Note that there might be other objects in the image
# for which ground truth data has not been provided.

# Top left pixel co-ordinates : (1, 1)

# Details for object 1 ("PAScatTrunc")
Original label for object 1 "PAScatTrunc" : "PAScatTrunc"
Bounding box for object 1 "PAScatTrunc" (Xmin, Ymin) - (Xmax, Ymax) : (85, 114) - (452, 332)

# Details for object 2 ("PAScatTrunc")
Original label for object 2 "PAScatTrunc" : "PAScatTrunc"
Bounding box for object 2 "PAScatTrunc" (Xmin, Ymin) - (Xmax, Ymax) : (302, 50) - (390, 159)

# Details for object 3 ("PAScat")
Original label for object 3 "PAScat" : "PAScat"
Bounding box for object 3 "PAScat" (Xmin, Ymin) - (Xmax, Ymax) : (135, 47) - (262, 202)
```

its descripted as

- Image FileName
- Image size
- Number of objects present with ground truth value
- Diagonal coordinates (i.e. bounding box) of ground object

Preprocess the data

Create two folder: train and val data

And copy the images into either train and val data as per mention in train.txt and val.txt

In [4]:

```
# Read the train.txt file
with open(trainDir+'ImageSets/train.txt','r') as f:
    c = f.read()
c
```

Out[4]:

```
'000016\n000017\n000023\n000025\n000029\n000032\n000036\n000040\n000041\n000050\n000051\n000056\n000057
\n000062\n000070\n000074\n000079\n000082\n000083\n000086\n000092\n000094\n000096\n000103\n000104\n00011
4\n000116\n000120\n000122\n000124\n000127\n000128\n000132\n000136\n000137\n000149\n000150\n000154\n0001
58\n000159\n000162\n000167\n000180\n000188\n000190\n000199\n000200\n000207\n000210\n000219\n000221\n000
224\n000231\n000233\n000241\n000242\n000246\n000248\n000251\n000255\n000256\n000257\n000262\n000264\n00
0267\n000268\n000281\n000282\n000283\n000288\n000293\n000298\n000307\n000315\n000316\n000318\n000326\n0
00328\n000331\n000342\n000343\n000344\n000351\n000352\n000353\n000358\n000359\n000360\n000365\n000366\n
000368\n000377\n000378\n000388\n000390\n000392\n000393\n000395\n000397\n000403\n000407\n000415\n000424\
n000426\n000432\n000438\n000440\n000441\n000444\n000448\n000452\n000466\n000468\n000471\n000482\n000490
\n000492\n000493\n000494\n000502\n000506\n000522\n000524\n000529\n000530\n000535\n000536\n000541\n00054
6\n000547\n000554\n000557\n000562\n000563\n000572\n000579\n000584\n000591\n000595\n000596\n000600\n0006
01\n000604\n000610\n000612\n000622\n000623\n000626\n000630\n000633\n000636\n000642\n000645\n000653\n000
655\n000658\n000662\n000663\n000671\n000672\n000674\n000681\n000687\n000688\n000695\n000696\n000708\n00
0719\n000723\n000725\n000726\n000739\n000744\n000747\n000750\n000752\n000754\n000765\n000770\n000781\n0
00785\n000788\n000792\n000795\n000798\n000802\n000806\n000811\n000813\n000818\n000822\n000828\n000830\n
000831\n000832\n000838\n000853\n000858\n000859\n000864\n000865\n000867\n000868\n000874\n000875\n000880\
n000882\n000891\n000892\n000893\n000894\n000895\n000901\n000910\n000912\n000914\n000918\n000920\n000927
\n000928\n000933\n000934\n000939\n000958\n000963\n000975\n000976\n000985\n000988\n000989\n000990\n00100
1\n001009\n001012\n001013\n001024\n001033\n001034\n001043\n001044\n001046\n001047\n001054\n001060\n0010
63\n001066\n001071\n001072\n001076\n001085\n001090\n001091\n001093\n001095\n001101\n001102\n001107\n001
108\n001117\n001126\n001128\n001133\n001139\n001140\n001142\n001150\n001151\n001154\n001157\n001164\n00
1165\n001170\n001177\n001178\n001179\n001180\n001184\n001189\n001190\n001194\n001195\n001203\n001208\n0
01210\n001212\n001214\n001220\n001224\n001235\n001238\n001244\n001248\n001249\n001251\n001257\n001260\n
001264\n001272\n001276\n001280\n001296\n001300\n001301\n001309\n001311\n001313\n001317\n001318\n001326\
n001327\n001332\n001333\n001334\n001337\n001341\n001349\n001351\n001352\n001359\n001365\n001373\n001378
\n001379\n001384\n001389\n001391\n001398\n001404\n001408\n001409\n001422\n001426\n001432\n001433\n00143
6\n001439\n001441\n001447\n001452\n001456\n001461\n001467\n001469\n001481\n001485\n001492\n001496\n0014
97\n001499\n001501\n001502\n001506\n001507\n001518\n001521\n001523\n001530\n001531\n001534\n001541\n001
545\n001546\n001548\n001554\n001556\n001570\n001575\n001581\n001582\n001584\n001588\n001594\n001600\n00
1605\n001607\n001611\n001619\n001620\n001622\n001628\n001629\n001638\n001640\n001641\n001645\n001647\n0
01651\n001652\n001653\n001654\n001660\n001662\n001664\n001668\n001669\n001677\n001679\n001687\n001701\n
001705\n001709\n001712\n001713\n001714\n001719\n001722\n001725\n001740\n001741\n001744\n001745\n001749\
n001750\n001754\n001756\n001765\n001768\n001771\n001773\n001781\n001785\n001788\n001791\n001794\n001795
\n001798\n001804\n001808\n001810\n001812\n001813\n001814\n001825\n001830\n001848\n001849\n001853\n00185
5\n001856\n001859\n001863\n001869\n001873\n001878\n001883\n001887\n001897\n001910\n001914\n001915\n0019
18\n001919\n001932\n001936\n001941\n001944\n001946\n001951\n001954\n001955\n001960\n001965\n001974\n001
982\n001987\n001989\n001991\n001999\n002004\n002005\n002008\n002021\n002024\n002032\n002037\n002039\n00
2041\n002045\n002050\n002052\n002053\n002066\n002069\n002072\n002077\n002078\n002085\n002087\n002089\n0
02090\n002091\n002096\n002099\n002103\n002108\n002109\n002114\n002118\n002124\n002125\n002126\n002140\n
002145\n002149\n002152\n002163\n002170\n002174\n002175\n002179\n002181\n002184\n002187\n002188\n002189\
n002190\n002192\n002212\n002213\n002215\n002222\n002231\n002234\n002236\n002238\n002240\n002244\n002245
\n002252\n002265\n002269\n002273\n002284\n002285\n002286\n002287\n002289\n002290\n002294\n002299\n00230
0\n002304\n002308\n002315\n002318\n002321\n002325\n002326\n002332\n002338\n002346\n002348\n002350\n0023
52\n002362\n002367\n002374\n002379\n002386\n002388\n002393\n002395\n002400\n002405\n002410\n002421\n002
424\n002426\n002429\n002438\n002444\n002448\n002456\n002459\n002460\n002463\n002469\n002474\n002482\n00
2486\n002492\n002497\n002503\n002508\n002510\n002512\n002514\n002523\n002529\n002530\n002535\n002539\n0
02544\n002552\n002553\n002554\n002557\n002560\n002561\n002564\n002565\n002567\n002569\n002570\n002573\n
002576\n002577\n002578\n002583\n002584\n002587\n002590\n002596\n002597\n002601\n002614\n002620\n002621\
n002622\n002624\n002629\n002634\n002640\n002642\n002643\n002650\n002651\n002652\n002654\n002661\n002663
\n002665\n002674\n002676\n002677\n002683\n002692\n002698\n002699\n002701\n002702\n002709\n002710\n00271
3\n002722\n002724\n002725\n002727\n002735\n002739\n002751\n002752\n002753\n002756\n002759\n002766\n0027
69\n002771\n002773\n002776\n002781\n002783\n002784\n002787\n002788\n002789\n002794\n002798\n002802\n002
807\n002817\n002823\n002825\n002827\n002834\n002843\n002845\n002846\n002847\n002850\n002851\n002856\n00
2860\n002861\n002871\n002879\n002885\n002886\n002895\n002898\n002899\n002908\n002914\n002915\n002924\n0
02929\n002932\n002936\n002947\n002951\n002956\n002965\n002968\n002972\n002976\n002983\n002994\n002995\n
002996\n003001\n003002\n003010\n003014\n003016\n003021\n003026\n003027\n003036\n003037\n003041\n003043\
n003045\n003046\n003048\n003056\n003059\n003060\n003062\n003063\n003071\n003072\n003073\n003074\n003089
\n003096\n003097\n003100\n003102\n003112\n003113\n003116\n003124\n003128\n003133\n003135\n003138\n00314
3\n003148\n003150\n003158\n003159\n003160\n003169\n003172\n003174\n003180\n003186\n003187\n003188\n0031
96\n003198\n003201\n003208\n003209\n003216\n003220\n003227\n003234\n003235\n003238\n003239\n003247\n003
250\n003251\n003256\n003271\n003273\n003275\n003276\n003278\n003279\n003280\n003285\n003286\n003289\n00
3291\n003294\n003301\n003304\n003310\n003312\n003313\n003326\n003330\n003333\n003335\n003337\n003339\n0
03344\n003345\n003348\n003355\n003358\n003359\n003360\n003364\n003373\n003374\n003378\n003384\n003386\n
003392\n003401\n003406\n003407\n003409\n003415\n003418\n003421\n003422\n003423\n003427\n003431\n003435\
n003437\n003440\n003444\n003445\n003455\n003456\n003459\n003460\n003475\n003479\n003488\n003493\n003502
```

```
n003437\n003440\n003444\n003445\n003455\n003456\n003459\n003460\n003475\n003479\n003488\n003493\n003502
\n003505\n003506\n003510\n003516\n003517\n003521\n003537\n003543\n003544\n003548\n003550\n003555\n00355
7\n003558\n003561\n003563\n003566\n003567\n003568\n003570\n003573\n003575\n003585\n003590\n003592\n0036
00\n003602\n003612\n003613\n003614\n003616\n003618\n003619\n003621\n003629\n003631\n003633\n003635\n003
644\n003645\n003646\n003651\n003655\n003656\n003658\n003661\n003666\n003669\n003676\n003679\n003682\n00
3692\n003694\n003698\n003702\n003706\n003710\n003712\n003715\n003720\n003723\n003724\n003730\n003732\n0
03733\n003740\n003743\n003752\n003760\n003762\n003765\n003767\n003771\n003773\n003776\n003784\n003785\n
003787\n003790\n003793\n003802\n003805\n003831\n003833\n003834\n003836\n003844\n003852\n003854\n003859\
n003861\n003865\n003867\n003870\n003871\n003877\n003880\n003886\n003888\n003899\n003901\n003904\n003910
\n003914\n003918\n003923\n003924\n003926\n003932\n003934\n003944\n003946\n003948\n003950\n003958\n00395
9\n003961\n003963\n003965\n003969\n003971\n003975\n003976\n003986\n003987\n003988\n003989\n004008\n0040
09\n004010\n004016\n004017\n004019\n004022\n004024\n004027\n004031\n004039\n004040\n004052\n004053\n004
055\n004060\n004062\n004063\n004066\n004070\n004083\n004090\n004093\n004096\n004100\n004101\n004102\n00
4105\n004106\n004108\n004113\n004115\n004122\n004127\n004132\n004134\n004140\n004146\n004156\n004160\n0
04173\n004181\n004182\n004186\n004190\n004193\n004197\n004199\n004201\n004205\n004207\n004208\n004212\n
004217\n004218\n004228\n004229\n004239\n004242\n004244\n004245\n004260\n004264\n004272\n004273\n004276\
n004279\n004294\n004295\n004296\n004298\n004302\n004309\n004314\n004315\n004318\n004321\n004328\n004338
\n004339\n004341\n004350\n004352\n004353\n004357\n004358\n004359\n004360\n004368\n004375\n004383\n00438
8\n004394\n004396\n004398\n004403\n004406\n004410\n004418\n004422\n004425\n004427\n004428\n004431\n0044
32\n004434\n004437\n004438\n004440\n004445\n004451\n004453\n004467\n004470\n004471\n004474\n004481\n004
485\n004487\n004489\n004494\n004503\n004511\n004519\n004520\n004523\n004532\n004535\n004536\n004543\n00
4544\n004554\n004555\n004558\n004561\n004579\n004580\n004581\n004582\n004585\n004586\n004589\n004595\n0
04602\n004605\n004607\n004609\n004618\n004625\n004631\n004632\n004634\n004638\n004639\n004640\n004653\n
004655\n004656\n004662\n004664\n004665\n004673\n004677\n004684\n004688\n004691\n004692\n004696\n004700\
n004703\n004705\n004717\n004722\n004728\n004736\n004738\n004741\n004743\n004748\n004753\n004755\n004760
\n004766\n004767\n004781\n004783\n004790\n004799\n004800\n004807\n004818\n004819\n004822\n004829\n00483
0\n004831\n004832\n004836\n004843\n004846\n004847\n004851\n004857\n004866\n004867\n004869\n004877\n0048
79\n004885\n004892\n004893\n004894\n004900\n004905\n004907\n004912\n004915\n004916\n004917\n004926\n004
928\n004929\n004940\n004941\n004943\n004948\n004952\n004954\n004963\n004965\n004966\n004968\n004969\n00
4971\n004981\n004986\n004989\n004992\n004994\n004997\n005006\n005007\n005011\n005020\n005025\n005035\n0
05041\n005055\n005057\n005059\n005060\n005061\n005064\n005065\n005072\n005075\n005076\n005079\n005083\n
005087\n005095\n005101\n005104\n005105\n005116\n005122\n005124\n005128\n005137\n005138\n005142\n005152\
n005153\n005155\n005159\n005161\n005163\n005172\n005175\n005184\n005191\n005195\n005199\n005201\n005203
\n005216\n005218\n005220\n005221\n005225\n005234\n005235\n005240\n005241\n005244\n005245\n005248\n00525
3\n005262\n005266\n005268\n005272\n005273\n005276\n005278\n005279\n005284\n005290\n005291\n005294\n0052
98\n005299\n005303\n'
```

It's seems it is separated by '\n' between each filename. Let separate it by '\n' to get the list of filename

In [5]:

```
# Separate by '\n'
c = c.split('\n')
# Print 10 elements of filenames
c[:10]
```

Out[5]:

```
['000016',
 '000017',
 '000023',
 '000025',
 '000029',
 '000032',
 '000036',
 '000040',
 '000041',
 '000050']
```

In [6]:

```
# Just append each filename with train directory from PNGImages
for i in c[:10]:
    # If string is not empty
    if i:
        print(trainDir+'PNGImages/'+i+'.jpg')
```

```
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000016.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000017.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000023.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000025.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000029.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000032.jpg
```

```
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000032.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000036.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000040.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000041.jpg
TrainDataSet/VOCdevkit/VOC2006/PNGImages/000050.jpg
```

In [7]:

```python
# Move all images into separate folders: train and val
def MovSepImage(train_imageDir, train_txtfile, imgDest):
    '''
    Separate ALL images into train and val images as per text mentioned.

    Parameters:
    train_imageDir: Source image Directory
    train_txtfile: Text filename which contain list of image name
    imgDest: Destination Image Directory
    '''
    with open(train_imageDir+train_txtfile) as f:
        c = f.read()
    c = c.split('\n')
    for i in c:
        # If string is not empty
        if i:
            # whether directory exist or not
            if imgDest not in os.listdir():
                os.makedirs(imgDest)

            # Copy this image and move to new directory called train
            shutil.copy(trainDir+'PNGImages/'+i+'.png', imgDest)

    print('Done!')
```

In [8]:

```python
MovSepImage(trainDir+'ImageSets/', 'train.txt', 'train')
```

```
Done!
```

In [9]:

```python
MovSepImage(trainDir+'ImageSets/', 'val.txt', 'val')
```

```
Done!
```

Store Bounding box value of each eimage and their corresponding label

We will store in `dict()` format

```
{
    ImageName1: [[x1,y1,x2,y2,object_label1],
                 [x1,y1,x2,y2,object_label2],
                 ...
                 ]
    ImageName2: [[x1,y1,x2,y2,object_label1],
                 [x1,y1,x2,y2,object_label2],
                 ...
                 ]
    ...
}
```

In [10]:

```python
# This is variable which contain only train iamges
train_img = 'train/'
```

```
for i in os.listdir(train_img):
    print(i)
    with open(trainDir+'Annotations/'+i.split('.')[0]+'.txt') as f:
        lines = f.readlines()
    break
```

```
000016.png
```

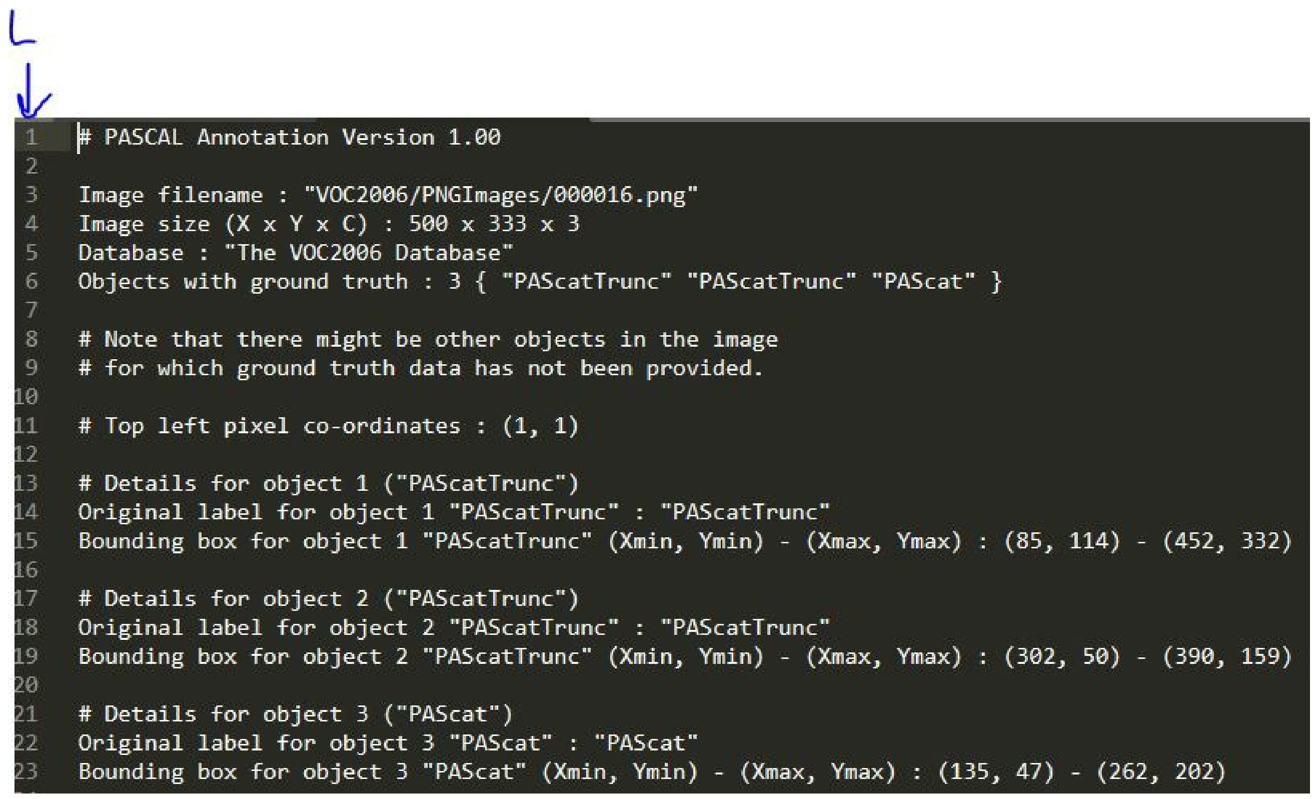
In [11]:

```
lines[0]
```

Out[11]:

```
'# PASCAL Annotation Version 1.00\n'
```

Observe **L** from below indicate line number for annotation text file.



```
 1  # PASCAL Annotation Version 1.00
 2
 3  Image filename : "VOC2006/PNGImages/000016.png"
 4  Image size (X x Y x C) : 500 x 333 x 3
 5  Database : "The VOC2006 Database"
 6  Objects with ground truth : 3 { "PAScatTrunc" "PAScatTrunc" "PAScat" }
 7
 8  # Note that there might be other objects in the image
 9  # for which ground truth data has not been provided.
10
11  # Top left pixel co-ordinates : (1, 1)
12
13  # Details for object 1 ("PAScatTrunc")
14  Original label for object 1 "PAScatTrunc" : "PAScatTrunc"
15  Bounding box for object 1 "PAScatTrunc" (Xmin, Ymin) - (Xmax, Ymax) : (85, 114) - (452, 332)
16
17  # Details for object 2 ("PAScatTrunc")
18  Original label for object 2 "PAScatTrunc" : "PAScatTrunc"
19  Bounding box for object 2 "PAScatTrunc" (Xmin, Ymin) - (Xmax, Ymax) : (302, 50) - (390, 159)
20
21  # Details for object 3 ("PAScat")
22  Original label for object 3 "PAScat" : "PAScat"
23  Bounding box for object 3 "PAScat" (Xmin, Ymin) - (Xmax, Ymax) : (135, 47) - (262, 202)
```

Since, python arrqay index **start with 0**. So, lines[0] is a line number 1 in text file.

Observing above to illurating how to fetch particular pattern

In Line number 6, Number of object present in it (i.e.3) which means there will be 3 object_label and their corresponding bounding box value, right?

Output should be like this:

> dict() has key and values pairs where key stored image name and values store all bounding box and their label present in image name in 2D matrix

```
{
    000016.png: [[85,114,452,332,PAScatTrunc],
                 [302,50,390,159,PAScatTrunc],
                 [135,47,262,202,PAScat]]
}
```

So, Line Number 6 will tell you number of object present and object label. Bounding box start from line number 6 to pattern like line number as 15, 19, 23... (i.e. 6+9= 15, 15+4 = 19, 19+4 = 23, ...)

So, we will jump first time with 9 and then add 4 consequence to get bounding box value as per number of objects present in it.

Illustration of finding label name

In [12]:

```
print('Line Number 6 from text:\n',lines[5])
print('*'*50)
print('Split by \':\' \n',lines[5].split(':'))
print('Interested in second part. Move to that second index\n', lines[5].split(':')[1])
print('*'*50)
print('Split by \'(space)\' \n',lines[5].split(':')[1].split(' '))
print('Second index tell the number of object present. And start from 4 index to till number of object
present in it')
print('*'*50)
num_obj = int(lines[5].split(':')[1].split(' ')[1])
for i in range(num_obj):
    print('Label Name:',lines[5].split(':')[1].split(' ')[i+3])
```

```
Line Number 6 from text:
 Objects with ground truth : 3 { "PAScatTrunc" "PAScatTrunc" "PAScat" }

**************************************************
Split by ':'
 ['Objects with ground truth ', ' 3 { "PAScatTrunc" "PAScatTrunc" "PAScat" }\n']
Interested in second part. Move to that second index
  3 { "PAScatTrunc" "PAScatTrunc" "PAScat" }

**************************************************
Split by '(space)'
 ['', '3', '{', '"PAScatTrunc"', '"PAScatTrunc"', '"PAScat"', '}\n']
Second index tell the number of object present. And start from 4 index to till number of object present
in it
**************************************************
Label Name: "PAScatTrunc"
Label Name: "PAScatTrunc"
Label Name: "PAScat"
```

In [13]:

```
# Let try for another annotation text file to make sure everything is working alright
with open(trainDir+'Annotations/000023.txt') as f:
        lines = f.readlines()
```

In [14]:

```
print('Line Number 6 from text:\n',lines[5])
print('*'*50)
print('Split by \':\' \n',lines[5].split(':'))
print('Interested in second part. Move to that second index\n', lines[5].split(':')[1])
print('*'*50)
print('Split by \'(space)\' \n',lines[5].split(':')[1].split(' '))
print('Second index tell the number of object present. And start from 4 index to till number of object
present in it')
print('*'*50)
num_obj = int(lines[5].split(':')[1].split(' ')[1])
for i in range(num_obj):
    print('Label Name:',lines[5].split(':')[1].split(' ')[i+3])
```

```
Line Number 6 from text:
 Objects with ground truth : 1 { "PASdogFrontalTrunc" }

**************************************************
Split by ':'
 ['Objects with ground truth ', ' 1 { "PASdogFrontalTrunc" }\n']
Interested in second part. Move to that second index
  1 { "PASdogFrontalTrunc" }

**************************************************
Split by '(space)'
```

```
 ['', '1', '{', '"PASdogFrontalTrunc"', '}\n']
Second index tell the number of object present. And start from 4 index to till number of object present
in it
**************************************************
Label Name: "PASdogFrontalTrunc"
```

Rectify the label name

Its seems to working perfectly alright!

Now **one last things**, label name has been to refined. There are 10 labels (Check in Imagesets folder) follow as:

1. bicycle
2. bus
3. car
4. cat
5. cow
6. dog
7. horse
8. motorbike
9. person
10. sheep
11. background (explicitly added to make sure whether it is object or just plain)

In [15]:

```python
label = ['bicycle','bus','car','cat','cow','dog','horse','motorbike','person','sheep', 'background']
```

In [16]:

```python
num_obj = int(lines[5].split(':')[1].split(' ')[1])
for i in range(num_obj):
    for j in label:
        pat_mat = re.search(j,lines[5].split(':')[1].split(' ')[i+3])
        if pat_mat:
            print('Label Name',j)
            break
```

```
Label Name dog
```

Locating the bounding box value

In [17]:

```python
# Just like line number starting from 15, 19, 23, ... (number of object label)
print('Line number 15:\n',lines[14])
print('*'*50)
print('Split by \':\'\n',lines[14].split(':'))
print('We need to go into second index as it contain bounding box value\n',lines[14].split(':')[1])
print('*'*50)
print('Remove all the special character\n',re.sub(r'[^a-zA-Z0-9]+',' ',lines[14].split(':')[1]))
print('Split by \'.\'\n', re.sub(r'[^a-zA-Z0-9]+',' ',lines[14].split(':')[1]).split(' '))
```

```
Line number 15:
 Bounding box for object 1 "PASdogFrontalTrunc" (Xmin, Ymin) - (Xmax, Ymax) : (158, 117) - (362, 375)

**************************************************
Split by ':'
 ['Bounding box for object 1 "PASdogFrontalTrunc" (Xmin, Ymin) - (Xmax, Ymax) ', ' (158, 117) - (362, 3
75)\n']
We need to go into second index as it contain bounding box value
  (158, 117) - (362, 375)

**************************************************
Remove all the special character
  158 117 362 375
  158 117 362 375
Split by '.'
 ['', '158', '117', '362', '375', '']
```

Now, if we have more than one object label, just add `4k` where **k** denotes **number of objects label**.

`re.sub(r'[^a-zA-Z0-9]+',' ',lines[14+4*k].split(':')[1]).split(' ')`

when k=0 (basically means one object), lines[14]

when k=1 (means 2 object labels), lines[14+4*k] -> lines[14+4*1] = lines[18]

> **Kindly remember, array index start with 0, that why here it is illustrated as starting index as 0**

Combined everything into one

In [18]:

```
def annot_label(train_img, train_dir):
    '''
    Annotate Image label with bounding box in Dictionary format

    Parameters:
    train_img: Train/Val Image Directory
    train_dir: Directory which contain Annotation folder

    Return:
    dict() format contain key as imageName and value as bounding box (x1,y1,x2,y2) and their object lab
el
    '''

    annot_dict = dict()

    label = ['bicycle','bus','car','cat','cow','dog','horse','motorbike','person','sheep']

    # Iterate all image filename fr0m train image folder
    for i in os.listdir(train_img):

        # create variable for values in dict() (which is bounding box value and their object label)
        array2D = []

        # Read image file image from Annotation Folder which describe about bounding box
        with open(train_dir+'Annotations/'+i.split('.')[0]+'.txt') as f:

            # Store all the text files linewise
            lines = f.readlines()

            # Go to line number 5 and store the number of objects
            num_obj = int(lines[5].split(':')[1].split(' ')[1])

            # Go to bounding box value starting from line number 15.
            # (Kindly go through above description which is explained it line number pattern.)
            for j in range(num_obj):

                # Now iterate over label value to get the actual label value
                # as label in Annotation text file not defined very well
                for k in label:

                    # Find the pattern if any of the label present in substring in object_label in text
file
                    # For example: PASdogFrontalTrunc  is labelled as dog
                    pat_mat = re.search(k,lines[5].split(':')[1].split(' ')[j+3])

                    # If any label pattern found, consider this as a label
                    if pat_mat:

                        # Now, go the bounding box value from line number pattern 15, 15+4, 15+4+4, ...
(as number of objects)
                        # 15, 15+4, 15+4*2,....
                        bb_val = re.sub(r'[^a-zA-Z0-9]+',' ',lines[14+4*j].split(':')[1]).split(' ')
                        array2D.append([int(bb_val[1]),int(bb_val[2]),int(bb_val[3]),int(bb_val[4]),k])
                        break
                    break

            # Store in dict()
            annot_dict[i] = array2D

    return annot_dict
```

In [19]:

```
train_annotLabel = annot_label(train_img, trainDir)
```

In [20]:

```
# You can change the image name in train folder and cross verify the object label
train_annotLabel['000016.png']
```

Out[20]:

```
[[85, 114, 452, 332, 'cat'],
 [302, 50, 390, 159, 'cat'],
 [135, 47, 262, 202, 'cat']]
```

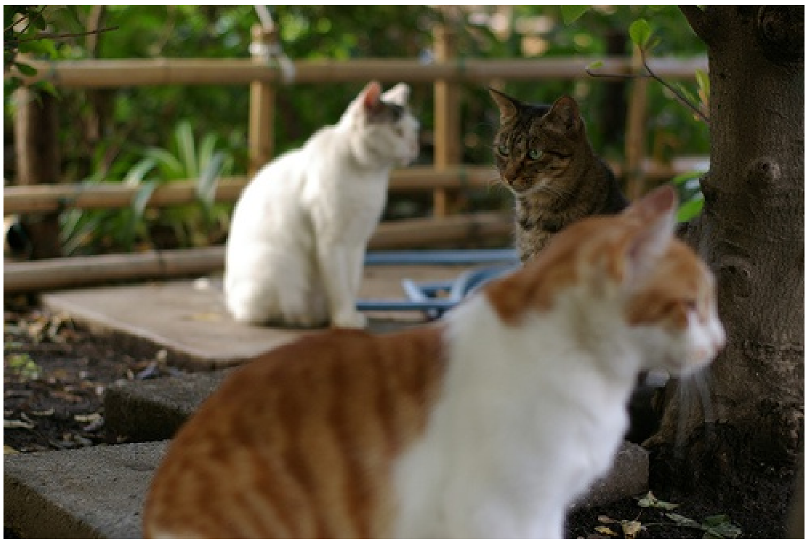


Figure ImageFile: 000016.jpg

In [21]:

```
val_img = 'val/'
val_annotLabel = annot_label(val_img, trainDir)
```

In [22]:

```
# You can change the image name in val folder and cross verify the object label
val_annotLabel['000263.png']
```

Out[22]:

```
[[301, 118, 382, 375, 'person'],
 [93, 115, 175, 375, 'person'],
 [1, 2, 500, 362, 'bus']]
```

Figure ImageFile: 000256.jpg

Plot images with ground truth

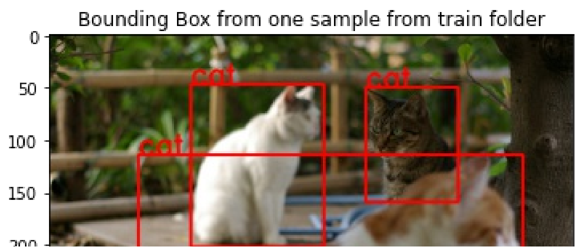
Let check it out the ground value bounding box after going through all of these.

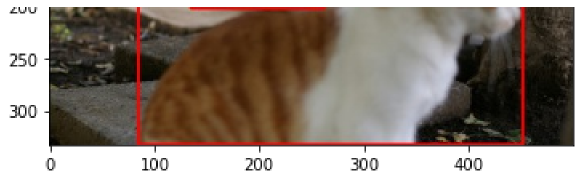
In [23]:

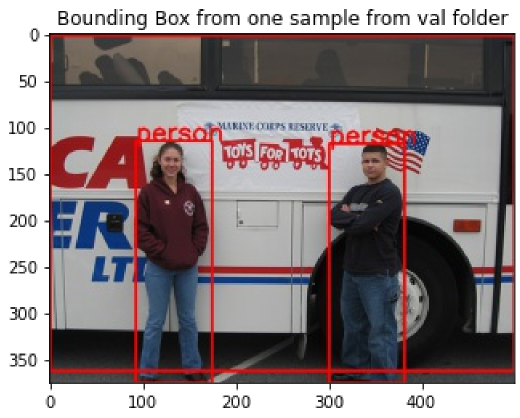
```
# Assign any one image name from train folder
imageName = '000016.png'

# Plot the images
plt.figure()
# Read the image using cv2
im1 = cv2.imread(train_img+imageName)
# By default cv2 read as BGR. So convert into RGB format channel
im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
# Since there are multuiple object labels present, we will iterate
# through number of label present
for i in range(len(train_annotLabel[imageName])):
    # Fetch x1,y1,x2,y2,label from train_annotLabel variable
    # which have same imageName (i.e. '000016.png').
    x1,y1,x2,y2,label = train_annotLabel[imageName][i]
    # We got two coordinates, just plot the box.
    imBB = cv2.rectangle(im1, (x1,y1), (x2,y2), (255, 0, 0), 2)
    # Put the label outsided the bounding box
    cv2.putText(imBB, label, (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
    # If it containes more object, it will go back and getch the next object label
    # and their bounding box value/

plt.title('Bounding Box from one sample from train folder')
plt.imshow(im1)

# Same explaination aws above.
# Assign any one image from val folder
imageName = '000263.png'
plt.figure()
im1 = cv2.imread(val_img+imageName)
im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
for i in range(len(val_annotLabel[imageName])):
    x1,y1,x2,y2,label = val_annotLabel[imageName][i]
    imBB = cv2.rectangle(im1, (x1,y1), (x2,y2), (255, 0, 0), 2)
    cv2.putText(imBB, label, (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
plt.title('Bounding Box from one sample from val folder')
plt.imshow(im1)
```

Out[23]:

```
<matplotlib.image.AxesImage at 0x28b4c936bc8>
```







Save All progress into files

Keep all the variables into files so that we dont go through all these cells compile again.

In [24]:

```
# Create a file name called 'train_annotLabel.json' in json format
with open('train_annotLabel.json','w') as f:
    json.dump(train_annotLabel, f)
```

In [25]:

```
# Similarily for val annotate label stored in json format
with open('val_annotLabel.json','w') as f:
    json.dump(val_annotLabel, f)
```

Next time how to read json files that was dumped

In [26]:

```
with open('train_annotLabel.json') as f:
    data = json.load(f)
```

In [27]:

```
data['000016.png']
```

Out[27]:

```
[[85, 114, 452, 332, 'cat'],
 [302, 50, 390, 159, 'cat'],
 [135, 47, 262, 202, 'cat']]
```

Here we are done: **Checkpoint I.**

Till Now we done:

1. Separate all images into train and val folder
2. Create a dictionary where keys as imageName and their values is in 2D array which contain bounding box value and object label. (Same operation for val operation also)
3. Dump into json file. (Same Operation done for val images also)

Selective Search Algorithm

Now in the previous, we got images and their corresponding labels and bounding box.

In [1]:

```
import cv2 # for selective search algorithm
import matplotlib.pyplot as plt # For showing the image
import os # To get the list of files for particular folder
import json # To read json files which contain all images annotation contains BB and their labels
import tqdm # For displaying loading progress bar
import numpy as np # For storing train data and label
```

In [2]:

```
trainDir = 'train/'
valDir = 'val/'
```

In [3]:

```
with open('train_annotLabel.json') as f:
    train_annotLabel = json.load(f)

with open('val_annotLabel.json') as f:
    val_annotLabel = json.load(f)
```

In [4]:

```
train_annotLabel['000016.png']
```

Out[4]:

```
[[85, 114, 452, 332, 'cat'],
 [302, 50, 390, 159, 'cat'],
 [135, 47, 262, 202, 'cat']]
```

Notice!

Before going ahead the algorith, I've noticed that those image whose aspect ratio is greater than 2.2 will throw the error by SelectiveSearch Allgorithm provided by OpenCV.

So, we need to resize those images whose aspect ratio is greater than 2.2. Since we are changing the size, their ground truth coordination value has to be changes.

(Run One Time Only)

Remember! DO also on testing stage also.

In [6]:

```
# For training image

for i in os.listdir(trainDir):
    # Read Image
    img = cv2.imread(trainDir+i)
    # Store height and width of the image
    h_img, w_img = img.shape[0], img.shape[1]
    # Just make sure that aspect ratio should be less than 2.2
    # If height is greater than width and their aspect ratio is greater than 2.2
    # Just take the half of greater one (i.e height/2) and also half the y coordinates.
    if (h_img > w_img) and (h_img/w_img) > 2.2:
        img = cv2.resize(img, (w_img, h_img//2))
        for j in range(len(train_annotLabel[i])):
            # Fetch BB and label
            x1,y1,x2,y2,label = train_annotLabel[i][j]
            # Update value of coordinates
            y1, y2 = y1//2, y2//2
            # update annotation after updating coordinates
            train_annotLabel[i][j] = [x1,y1,x2,y2,label]

    # Similarly same if width is greater than height and aspect ratio is greater than 2.2
    # Just take he half of the width and also half the x coordinates.
    if (w_img > h_img) and (w_img/h_img) > 2.2:
        img = cv2.resize(img, (w_img//2, h_img))
        for j in range(len(train_annotLabel[i])):
            # Fetch BB and label
            x1,y1,x2,y2,label = train_annotLabel[i][j]
            # Update value of coordinates
            x1, x2 = x1//2, x2//2
            # update annotation after updating coordinates
            train_annotLabel[i][j] = [x1,y1,x2,y2,label]
    # Overwrite images into same directory 'train'
    cv2.imwrite(trainDir+i, img)

# For val image
for i in os.listdir(valDir):
    # Read Image
    img = cv2.imread(valDir+i)
    # Store height and width of the image
    h_img, w_img = img.shape[0], img.shape[1]
    # Just make sure that aspect ratio should be less than 2.2
    # If height is greater than width and their aspect ratio is greater than 2.2
    # Just take the half of greater one (i.e height/2) and also half the y coordinates.
    if (h_img > w_img) and (h_img/w_img) > 2.2:
        img = cv2.resize(img, (w_img, h_img//2))
        for j in range(len(val_annotLabel[i])):
            # Fetch BB and label
            x1,y1,x2,y2,label = val_annotLabel[i][j]
            # Update value of coordinates
            y1, y2 = y1//2, y2//2
            # update annotation after updating coordinates
            val_annotLabel[i][j] = [x1,y1,x2,y2,label]

    # Similarly same if width is greater than height and aspect ratio is greater than 2.2
    # Just take he half of the width and also half the x coordinates.
    if (w_img > h_img) and (w_img/h_img) > 2.2:
        img = cv2.resize(img, (w_img//2, h_img))
        for j in range(len(val_annotLabel[i])):
            # Fetch BB and label
            x1,y1,x2,y2,label = val_annotLabel[i][j]
            # Update value of coordinates
            x1, x2 = x1//2, x2//2
            # update annotation after updating coordinates
            val_annotLabel[i][j] = [x1,y1,x2,y2,label]
    # Overwrite images into same directory 'val'
    cv2.imwrite(valDir+i, img)

# Update json file of both train and val annotation BB
with open('train_annotLabel.json','w') as f:
    json.dump(train_annotLabel, f)

with open('val_annotLabel.json','w') as f:
    json.dump(val_annotLabel, f)
```

Algorithm

In [4]:

```
img = cv2.imread(trainDir+'000016.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Create the instance of Selective Search Algorithm
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
# Select the base image you want to perform selective search algorithm
ss.setBaseImage(img)
# There is two ways to proceed in Selective Search.
# 1. Switch to fast but low recall Selective Search method
ss.switchToSelectiveSearchFast()
# 2. Switch to high recall but slow Selective Search method
# ss.switchToSelectiveSearchQuality()
# Run algorithm on input image
result_ss = ss.process()
```

In [5]:

```
result_ss.shape
```

Out[5]:

```
(1272, 4)
```

It means that there are 1276 bounding boxes generated. And each bounding box consist of 4 coordinates: (x1,y1) and (w,h). i.e. (x2,y2) = (x1+w,y1+h)

In [6]:

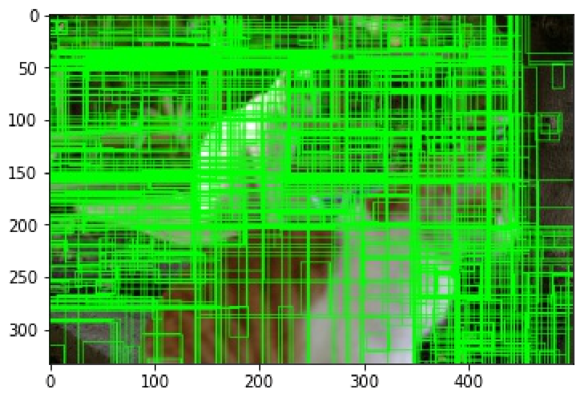
```
# Create copy of input image so that we don't need to read an image ovver again and again
img_dup = img.copy()
```

In [7]:

```
# Iterate over all number of bounding box
for i in range(result_ss.shape[0]):
    # Fetch each bounding box sample values
    x1,y1,w,h = result_ss[i]
    # We got four coordinates, just plot the box.
    cv2.rectangle(img, (x1,y1), (x1+w,y1+h), (0, 255, 0), 1)
plt.imshow(img)
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x21795745248>
```



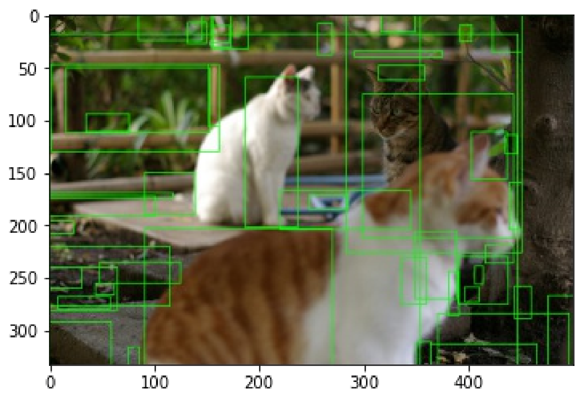
Too much fuzzyy, let reduce number of bounding box to get good views.

In [8]:

```
# Number of bounding box you need
num_BB = 50
# Assign restore image (which made a copy of an image)
img = img_dup

# Iterate over all number of bounding box
for i in range(result_ss.shape[0]):
    if i < num_BB:
        # Fetch each bounding box sample values
        x1,y1,w,h = result_ss[i]
        # We got four coordinates, just plot the box.
        cv2.rectangle(img, (x1,y1), (x1+w,y1+h), (0, 255, 0), 1)
plt.imshow(img)
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x21796801908>
```



In [4]:

```python
def SelectiveSearch(imgDir, ss_fast=True):
    '''
    Perform Selective Search Algorithm

    Parameter:
    imgDir: Enter the image directory
    ss_fast: Select the approach of Selective Search Approach: True for Fast or False for Quality

    Return:
    Return list of all bounding boxex (i.e. shape of [Number of BB, 4])
    where '4' value indicate as (x,y,w,h)
    '''
    img = cv2.imread(imgDir)

    # cv2 by default read as BGR channel. Convert into RGB channel
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Create the instance of Selective Search Algorithm
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

    # Select the base image you want to perform selective search algorithm
    ss.setBaseImage(img)

    # There is two ways to proceed in Selective Search.
    # 1. Switch to fast but low recall Selective Search method
    if ss_fast:
        ss.switchToSelectiveSearchFast()
    # 2. Switch to high recall but slow Selective Search method
    else:
        ss.switchToSelectiveSearchQuality()

    # Run algorithm on input image
    result_ss = ss.process()

    return result_ss
```

IOU

In [3]:

```python
def get_IOU(bb1,bb2):
    '''
    Calculate IOU of two bounding box

    Parameters:
    bb1: First bounding box in dict() format which contain 4 coordinates with labelled as 'x1','y1','x2
','y2'
    bb2: Second bounding box in dict() format which contain 4 coordinates with labelled as 'x1','y1','x
2','y2'
```

```
    Return:
    IOU value
    '''
    # Make sure in both Bounding boxes coordinates are valid
    assert bb1['x1'] < bb1['x2']
    assert bb1['y1'] < bb1['y2']
    assert bb2['x1'] < bb2['x2']
    assert bb2['y1'] < bb2['y2']

    # Finding intersection
    x_left = max(bb1['x1'],bb2['x1'])
    y_bottom = max(bb1['y1'],bb2['y1'])
    x_right = min(bb1['x2'],bb2['x2'])
    y_top = min(bb1['y2'],bb2['y2'])

    # Checking whether there is any intersection avaiable
    # If not, return area of intersection as 0.
    if (x_left > x_right) or (y_bottom > y_top):
        return 0.0
    # Otheriwise
    area_intersect = (x_right - x_left) * (y_top - y_bottom)

    # To bring into percentage, calculate area of each bounding box
    bb1_area = (bb1['x2'] - bb1['x1']) * (bb1['y2'] - bb1['y1'])
    bb2_area = (bb2['x2'] - bb2['x1']) * (bb2['y2'] - bb2['y1'])
    # p(AUB) = p(A) + p(B) - p(A,B)
    total_area = bb1_area + bb2_area - area_intersect
    # Finally in percentage
    iou = area_intersect / total_area

    # Make sure, IOU values within 0 to 1
    assert iou >= 0.0
    assert iou <= 1.0

    return iou
```

Let take one image as a example (which exceed aspect ratio greater than 2.2) and get the best IOU and BB values

In [11]:

```
# This image has a width = 500 and height: 189.
# Since w > h, so compute w/h which is greater than 2.2
# So, we updated the coordinate in annotLael.
# Let check it out whether it works or not.
imgName = '001091.png'

max_ = 0

for i in range(len(train_annotLabel[imgName])):
    img = cv2.imread(trainDir+imgName)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    x1,y1,x2,y2,label = train_annotLabel[imgName][i]
    gt_coord = {'x1': x1, 'y1':y1, 'x2': x2, 'y2': y2}

    img_ssBB = SelectiveSearch(trainDir+imgName)
    for j in range(img_ssBB.shape[0]):
        x1,y1,w,h = img_ssBB[j]
        x2,y2 = x1+w, y1+h
        pred_coord = {'x1': x1, 'y1':y1, 'x2': x2, 'y2': y2}
        iou_val = get_IOU(gt_coord, pred_coord)
        if max_ < iou_val:
            max_ = iou_val
            best_iou_coord = pred_coord
```

In [12]:

```
print('Maimum IOU Score abd BB value', max_,best_iou_coord)
```

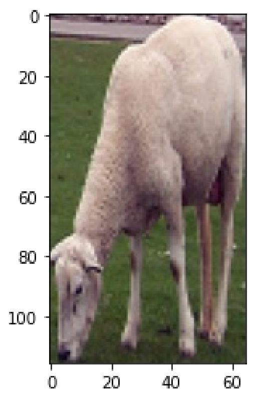
```
Maimum IOU Score abd BB value 0.9322249016372636 {'x1': 126, 'y1': 55, 'x2': 191, 'y2': 171}
```

In [13]:

```
# Let crop and select those BB value coordinate we got the best one
# plt.imshow(img[47:200, 139:261])
plt.imshow(img[55:171, 126:191])

# It seems that by convential cv2 read as
# img[y1:y2, x1:x2]
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x2179683afc8>
```



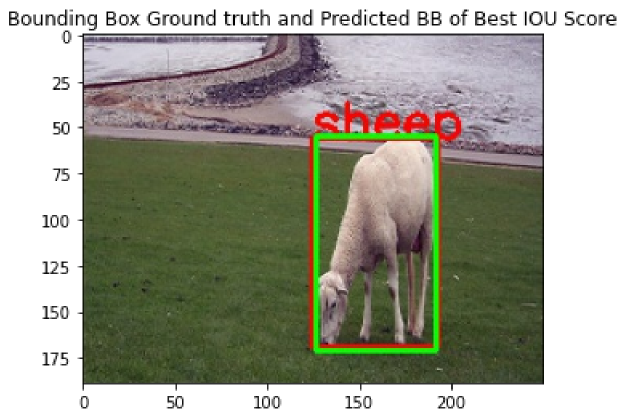
Let visualize by looking in an image and see the ground truth and preddict BB

In [14]:

```
plt.figure()

for i in range(len(train_annotLabel[imgName])):
    x1,y1,x2,y2,label = train_annotLabel[imgName][i]
    imBB = cv2.rectangle(img, (x1,y1), (x2,y2), (255, 0, 0), 2)
    cv2.putText(imBB, label, (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

cv2.rectangle(imBB, (best_iou_coord['x1'], best_iou_coord['y1']), (best_iou_coord['x2'], best_iou_coord
['y2']), \
              (0, 255, 0), 2)
plt.title('Bounding Box Ground truth and Predicted BB of Best IOU Score')
plt.imshow(img)
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x217968a88c8>
```



Bounding Box Ground truth and Predicted BB of Best IOU Score
sheep
0
25
50
75
100
125
150
175
0
50
100
150
200

By observation, we found the best IOU score between ground truth and predict BB (by selective search).

.:. Red color: Ground truth BB , Green color: Predicted BB by Selective Search

Now, there are multiple objects in an image, so we need to **compare each BB from selective search** to **every object in an**

Now, there are multiple objects in an image, so we need to **compare each BB from selective search** to **every object in an image** and choose whether they belong any partiular object with greater 70% of IOU or not. Otherwise set as background

In [15]:

```
# Assign label names as index value
label = ['bicycle','bus','car','cat','cow','dog','horse','motorbike','person','sheep','background']
label_dict = dict()

for i,j in enumerate(label):
    label_dict[j] = i
label_dict
```

Out[15]:

```
{'bicycle': 0,
 'bus': 1,
 'car': 2,
 'cat': 3,
 'cow': 4,
 'dog': 5,
 'horse': 6,
 'motorbike': 7,
 'person': 8,
 'sheep': 9,
 'background': 10}
```

ROI using Selective Search

1. Create the ROI using Selective Search and get Bounding box whose IOU is greater than 70% and less than 30%.
2. Store all the list of the images only ROI (crop ROI and resize into same input for all) and their corresponding labels.
3. We are storing regional Proposal BB `P` and their ground truth BB `G`. This will be used when we do bounding box regression.

Positive sample: {All labels except background} Negative Sample: background only

In [16]:

```
train_X = []
train_y = []
P = []
G = []

# Iterate over all images present in train folder
for i in tqdm.tqdm_notebook(os.listdir(trainDir)):
    # Read image
    img = cv2.imread(trainDir+i)
    # Number of region selected
    num_reg = 0
    # Counter for positive samples (which contain only object)
    pos_sample = 0
    # Counter for negative samples (which contain background)
    neg_sample = 0

    # cv2 by default read as BGR channel. Convert into RGB channel
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Get the BB Using Selective Search Algorithm
    img_ssBB = SelectiveSearch(trainDir+i)

    # Iterate over all sample BB which processed using Selective Search Algorithm
    for j in range(img_ssBB.shape[0]):
        # If number of region exceed 2000, terminate this loop
        # And proceed for next image
        if num_reg > 2000:
            break

        # Fetch bounding box coordinate
        x1,y1,w,h = img_ssBB[j]
        x2,y2 = x1+w, y1+h
        pred_coord = {'x1': x1, 'y1':y1, 'x2': x2, 'y2': y2}

        # Compare with all objects present in original image
        for k in range(len(train_annotLabel[i])):
            # Fetch ground truth bounding box coordinate
            g_x1,g_y1,g_x2,g_y2,g_label = train_annotLabel[i][k]
            gt_coord = {'x1': g_x1, 'y1':g_y1, 'x2': g_x2, 'y2': g_y2}
            # Get IOU score between ground truth and predicted BB
            iou_score = get_IOU(gt_coord, pred_coord)
            # Just to balance it, we made to have 5 number of positive sample
            if iou_score > 0.80 and pos_sample < 5:
                # Crop the image
                tr_img = img[y1:y2, x1:x2]
                # Resize the image into 227x227
                tr_resize = cv2.resize(tr_img, (227,227), cv2.INTER_AREA)
                tr_resize = tr_resize.astype('float32')
                tr_resize = tr_resize/255
                # Add to train_X list as well as their corresponding label
                train_X.append(tr_resize)
                train_y.append(label_dict[g_label])
                # Predicted BB and Ground truth BB
                P.append([pred_coord['x1'],pred_coord['y1'],pred_coord['x2'],pred_coord['y2']])
                G.append([gt_coord['x1'],gt_coord['y1'],gt_coord['x2'],gt_coord['y2']])
                pos_sample += 1
            # Just to balance it, we made to have 5 number of negative sample
            elif iou_score < 0.30 and neg_sample < 5:
                # Crop the image
                tr_img = img[y1:y2, x1:x2]
                # Resize the image into 227x227
                tr_resize = cv2.resize(tr_img, (227,227), cv2.INTER_AREA)
                # Add to train_X list as well as their corresponding label
                tr_resize = tr_resize.astype('float32')
                tr_resize = tr_resize/255
                train_X.append(tr_resize)
                train_y.append(label_dict['background'])
                # Predicted BB and Ground Truth BB set to zero Because
                # we are not interested to detect background.
                P.append([0,0,0,0])
                G.append([0,0,0,0])
                neg_sample += 1
            else:
                # If by chance both we got 10 positive sample as well as negative sample
                # No need to go further to select next region. Terminate here and
                # Move on to the next region
                if pos_sample > 5 and neg_sample > 5:
                    break
                # If one of the has to be filled, carrying on.
                else:
                    continue
```

```
C:\Anaconda3\envs\tf-gpu\lib\site-packages\ipykernel_launcher.py:7: TqdmDeprecationWarning: This functi
on will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  import sys
```

In [17]:

```
np.array(train_X).shape
```

Out[17]:

```
(10838, 227, 227, 3)
```

In [18]:

```
len(train_y)
```

Out[18]:

```
10838
```

In [19]:

```
len(P), len(G)
```

Out[19]:

```
(10838, 10838)
```

Save all training ROI images and their corresponding labels

In [20]:

```
import gc
```

In [21]:

```
P = np.array(P)
G = np.array(G)

np.save('P',P)
np.save('G',G)

del P, G
gc.collect()
```

Out[21]:

```
104
```

In [22]:

```
train_y = np.array(train_y)
np.save('train_y',train_y)

del train_y
gc.collect()
```

Out[22]:

```
20
```

In [23]:

```
train_X = np.array(train_X)
np.save('train_X',train_X)

del train_X
print('Done Saving')
gc.collect()
```

```
Done Saving
```

Out[23]:

```
20
```

Checkpoint II

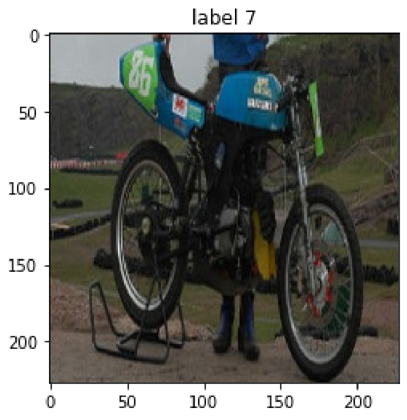
1. We have reduce some of the images which exceed aspect ratio greater than 2.2 and also updated annoted BB value also. So, we got updated train and val images as well as annotLabel.json files
2. We perform Selective Search Algorithm in all train images and crop the image whose IOU is greater than 70% of actual ground truth value.
3. We stored all list of images in train_X.npy as well as label train_y.npy

In [12]:

```
# Try display some images by changing i from 0 to 10838 (number of images present in train_X) with labe
l index
i = 12112
plt.title('label '+str(train_y[i]))
plt.imshow(train_X[i])
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x26a8e6a5b48>
```



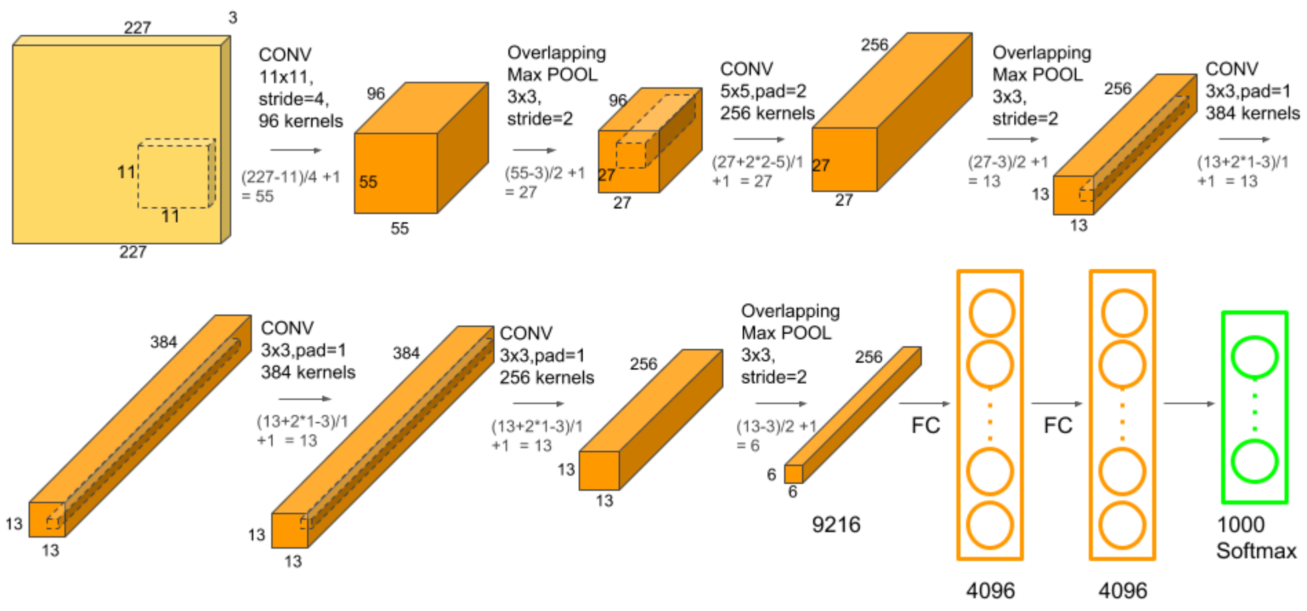
Training Architecture and Phase

In [1]:

```
import numpy as np # To load the saved train_X and train_y
import tensorflow as tf # For Deep learning framework
import os # To list of images for val
```

Alexnet Architecture

Image taken from https://www.learnopencv.com/understanding-alexnet/



Just minor changes, instead of 1000 classes in output layer, we set as per our number of classes (i.e. 10 classes)

Just minor changes, instead of 1000 classes in output layer, we set as per our number of classes (i.e. 10 classes)

In [2]:

```
def CNN5FC2(num_classes=10, input_shape=(227,227,3)):

    input_layer = tf.keras.layers.Input(shape=input_shape)
    conv1 = tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu')(i
nput_layer)
    maxpool1 = tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(conv1)
    conv2 = tf.keras.layers.Conv2D(filters=256, kernel_size=(5,5), padding='same', activation='relu')(m
axpool1)
    maxpool2 = tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(conv2)
    conv3 = tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), padding='same', activation='relu')(m
axpool2)
    conv4 = tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), padding='same', activation='relu')(c
onv3)
    conv5 = tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='relu')(c
onv4)
    maxpool3 = maxpool2 = tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2))(conv5)
    flat = tf.keras.layers.Flatten()(maxpool3)
    fc1 = tf.keras.layers.Dense(units=4096, activation='relu')(flat)
    fc2 = tf.keras.layers.Dense(units=4096, activation='relu')(fc1)
    output_layer = tf.keras.layers.Dense(units=num_classes, activation='softmax')(fc2)

    return input_layer, output_layer
```

In [3]:

```
# Load the train_X and train_y we just preprocessed
train_X = np.load('train_X.npy')
train_y = np.load('train_y.npy')
train_X.shape, train_y.shape
```

Out[3]:

```
((10838, 227, 227, 3), (10838,))
```

In [4]:

```
# Convert label into OneHot Encoding (for Categorical)
n_classes = len(np.unique(train_y))
train_y = tf.keras.utils.to_categorical(train_y, num_classes=n_classes)
```

In [5]:

```
# Instance of AlexNet Model
input_layer, output_layer = CNN5FC2(num_classes = n_classes)
alexnet_model = tf.keras.models.Model(input_layer, output_layer)
```

In [6]:

```
# Show the layers and their outpur shape and number of trainable parameters
alexnet_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 227, 227, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 384)       885120
```

```
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 384)       1327488   
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 256)       884992    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0         
_________________________________________________________________
dense (Dense)                (None, 4096)              37752832  
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312  
_________________________________________________________________
dense_2 (Dense)              (None, 11)                45067     
=================================================================
Total params: 58,326,411
Trainable params: 58,326,411
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
# Defiine Loss, Optimization and Metrics
opt_obj = tf.keras.optimizers.SGD(lr=0.001)
loss_obj = tf.keras.losses.CategoricalCrossentropy()
alexnet_model.compile(loss=loss_obj, optimizer=opt_obj, metrics=['accuracy'])
```

In [8]:

```
# Batch size for training
BATCH_SIZE = 16
```

Now, we are going to train the AlexNet Model. The purpose is that we want to have 2nd last layer (which is FC2) which contain 4096 features. So, once we trained the network, we can pass the train and val data to get 4096.

Remember, we are not here yet for object detection. We want to find 4096 features for each image. And DL has shown the promising about feature extraction from each image.

In [9]:

```
alexnet_model.fit(train_X, train_y, batch_size=BATCH_SIZE, epochs=10)
```

```
Train on 10838 samples
Epoch 1/10
10838/10838 [==============================] - 44s 4ms/sample - loss: 1.7186 - accuracy: 0.5845
Epoch 2/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.4214 - accuracy: 0.5996
Epoch 3/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.2536 - accuracy: 0.6212
Epoch 4/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.1682 - accuracy: 0.6345
Epoch 5/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.1217 - accuracy: 0.6441
Epoch 6/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.0938 - accuracy: 0.6484
Epoch 7/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.0677 - accuracy: 0.6557s - los
Epoch 8/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.0454 - accuracy: 0.6634
Epoch 9/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 1.0207 - accuracy: 0.6721
Epoch 10/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.9954 - accuracy: 0.6792
```

Out[9]:

```
<tensorflow.python.keras.callbacks.History at 0x22888120048>
```

In [10]:

```
alexnet_model.fit(train_X, train_y, batch_size=BATCH_SIZE, epochs=10)
```

```
Train on 10838 samples
Epoch 1/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.9732 - accuracy: 0.6904
Epoch 2/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.9560 - accuracy: 0.6931s - l
Epoch 3/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.9370 - accuracy: 0.6996
Epoch 4/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.9167 - accuracy: 0.7082
Epoch 5/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.8949 - accuracy: 0.7142
Epoch 6/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.8758 - accuracy: 0.7230
Epoch 7/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.8545 - accuracy: 0.7256
Epoch 8/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.8339 - accuracy: 0.7350
Epoch 9/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.8091 - accuracy: 0.7432ETA: 0s
- loss: 0.8096 - accuracy: 0.74 - ETA: 0s - loss: 0.8091 - accuracy: 0.74
Epoch 10/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.7822 - accuracy: 0.7506
```

Out[10]:

```
<tensorflow.python.keras.callbacks.History at 0x2288f08ec88>
```

In [11]:

```
alexnet_model.fit(train_X, train_y, batch_size=BATCH_SIZE, epochs=10)
```

```
Train on 10838 samples
Epoch 1/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.7575 - accuracy: 0.7595
Epoch 2/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.7329 - accuracy: 0.7645
Epoch 3/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.7086 - accuracy: 0.7753
Epoch 4/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.6782 - accuracy: 0.7841s - loss
: 0
Epoch 5/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.6472 - accuracy: 0.7948s - loss
: 0.645
Epoch 6/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.6186 - accuracy: 0.8013
Epoch 7/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.5819 - accuracy: 0.8135s - l
Epoch 8/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.5516 - accuracy: 0.8242
Epoch 9/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.5212 - accuracy: 0.8326s - loss
: 0 - ETA: 0s - los
Epoch 10/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.4831 - accuracy: 0.8472
```

Out[11]:

```
<tensorflow.python.keras.callbacks.History at 0x22904779688>
```

In [12]:

```
alexnet_model.fit(train_X, train_y, batch_size=BATCH_SIZE, epochs=10)
```

```
Train on 10838 samples
Epoch 1/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.4423 - accuracy: 0.8595
Epoch 2/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.4052 - accuracy: 0.8736
```

```
Epoch 3/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.3676 - accuracy: 0.8871s - loss
: 0.3665 - accura
Epoch 4/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.3369 - accuracy: 0.8946
Epoch 5/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.2975 - accuracy: 0.9081
Epoch 6/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.2742 - accuracy: 0.9196s - loss
: 0.2739 - accuracy: 0.
Epoch 7/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.2405 - accuracy: 0.9277
Epoch 8/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.2023 - accuracy: 0.9400
Epoch 9/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.1861 - accuracy: 0.9461
Epoch 10/10
10838/10838 [==============================] - 40s 4ms/sample - loss: 0.1687 - accuracy: 0.9505s - loss
: 0.1701 -  - ETA: 0s - los
```

Out[12]:

```
<tensorflow.python.keras.callbacks.History at 0x2288ddbb1c8>
```

In [13]:

```python
# store architecture into json format
arch_json = alexnet_model.to_json()
```

In [14]:

```python
# write and save architecture is json file
with open('alexnet_arch.json','w') as f:
    f.write(arch_json)
```

In [15]:

```python
# save the weights only.
alexnet_model.save_weights('alexnet_weights.hdf5')
```

Feature Extraction

Now,We are going to feature extract from the second last layer which have 4096 units (from trained Alexnet)

In [1]:

```python
import tensorflow as tf # For DL framework
import numpy as np # For saving variables into npy format
import cv2 # To operate on images and perform selective search algorithm
import json # To read/write json file
import tqdm # To display the progress bar
import os # To get the list of files for particular folder
```

In [2]:

```python
# Load Architecture and weights
json_file = open('alexnet_arch.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = tf.keras.models.model_from_json(loaded_model_json)
loaded_model.load_weights("alexnet_weights.hdf5")
```

In [3]:

```python
# Create another instantiate for removing last layer.
model = tf.keras.models.Model(loaded_model.input,loaded_model.layers[-2].output)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 227, 227, 3)]     0         
_________________________________________________________________
conv2d (Conv2D)              (None, 55, 55, 96)        34944     
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 256)       614656    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256)       0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 384)       885120    
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 384)       1327488   
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 256)       884992    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0         
_________________________________________________________________
dense (Dense)                (None, 4096)              37752832  
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312  
=================================================================
Total params: 58,281,344
Trainable params: 58,281,344
Non-trainable params: 0
_________________________________________________________________
```

In [4]:

```
# Load Train images performed ROI using SelectiveSearchAlgorithm
train_X = np.load('train_X.npy')
```

In [5]:

```
import tqdm # To Display the progress bar

# Find 4096 image feature extraction from pretrained Alexnet for each image
train_feat = []
for i in tqdm.tqdm_notebook(range(train_X.shape[0])):
    train_feat.append(list(model.predict(np.expand_dims(train_X[i], axis=0))[0]))
```

```
C:\Anaconda3\envs\tf-gpu\lib\site-packages\ipykernel_launcher.py:4: TqdmDeprecationWarning: This functi
on will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  after removing the cwd from sys.path.
```

In [6]:

```
del train_X
# Store the profress of 4096 feature extracted image
train_feat = np.array(train_feat)
np.save('train_X_feat',train_feat)
```

In [7]:

```
# Display some top value
print(train_feat[:10])
```

```
[[2.0644286  0.         1.3766023  ... 1.331283   0.3779589  0.        ]
 [2.0644286  0.         1.3766023  ... 1.331283   0.3779589  0.        ]
 [2.0644286  0.         1.3766023  ... 1.331283   0.3779589  0.        ]
 ...
 [2.5206459  0.         0.7407776  ... 1.8021966  0.         0.53547543]
 [2.9217916  0.         0.8671905  ... 1.8351811  0.         0.3917648 ]
 [2.3756118  0.         0.7365738  ... 1.8417557  0.         0.48135987]]
```

Similarily perform for val data

In [1]:

```python
import numpy as np # To load/save the variable in .npy format
import tensorflow as tf # To load the Alexnet architecture and predict 4096 features
import json # To read the annotation label of val
import cv2 # To perform SelectiveSearchAlgorithm for val images
import os # To get the list of images present in val folder
import tqdm # To display the progress bar
```

1. Don't worry about this below stuff. This is exactly same as we have done in training image. Just go to cell under the section `ROI using Selective Search`
2. Please make sure compile the cells which each contain two functions: `SelectiveSearchAlgorithm()` and `get_IOU()`

In [5]:

```python
valDir = 'val/'
with open('val_annotLabel.json') as f:
    val_annotLabel = json.load(f)

val_X = []
val_y = []
P_val = []
G_val = []

# Assign label names as index value
label = ['bicycle','bus','car','cat','cow','dog','horse','motorbike','person','sheep','background']
label_dict = dict()

for i,j in enumerate(label):
    label_dict[j] = i

# Iterate over all images present in val folder
for i in tqdm.tqdm_notebook(os.listdir(valDir)):
    # Read image
    img = cv2.imread(valDir+i)
    # Number of region selected
    num_reg = 0
    # Counter for positive samples (which contain only object)
    pos_sample = 0
    # Counter for negative samples (which contain background)
    neg_sample = 0

    # cv2 by default read as BGR channel. Convert into RGB channel
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Get the BB Using Selective Search Algorithm
    img_ssBB = SelectiveSearch(valDir+i)

    # Iterate over all sample BB which processed using Selective Search Algorithm
    for j in range(img_ssBB.shape[0]):
        # If number of region exceed 2000, terminate this loop
        # And proceed for next image
        if num_reg > 2000:
            break

        # Fetch bounding box coordinate
        x1,y1,w,h = img_ssBB[j]
        x2,y2 = x1+w, y1+h
        pred_coord = {'x1': x1, 'y1':y1, 'x2': x2, 'y2': y2}

        # Compare with all objects present in original image
        for k in range(len(val_annotLabel[i])):
            # Fetch ground truth bounding box coordinate
            g_x1,g_y1,g_x2,g_y2,g_label = val_annotLabel[i][k]
            gt_coord = {'x1': g_x1, 'y1':g_y1, 'x2': g_x2, 'y2': g_y2}
            # Get IOU score between ground truth and predicted BB
            iou_score = get_IOU(gt_coord, pred_coord)
            # Just to balance it, we made to have 5 number of positive sample
            if iou_score > 0.80 and pos_sample < 5:
                # Crop the image
                val_img = img[y1:y2, x1:x2]
                # Resize the image into 227x227
                val_resize = cv2.resize(val_img, (227,227), cv2.INTER_AREA)
                val_resize = val_resize.astype('float32')
                val_resize = val_resize/255
                # Add to train_X list as well as their corresponding label
                val_X.append(val_resize)
                val_y.append(label_dict[g_label])
                P_val.append([pred_coord['x1'],pred_coord['y1'],pred_coord['x2'],pred_coord['y2']])
                G_val.append([gt_coord['x1'],gt_coord['y1'],gt_coord['x2'],gt_coord['y2']])
                pos_sample += 1
            # Just to balance it, we made to have 5 number of negative sample
            elif iou_score < 0.30 and neg_sample < 5:
                # Crop the image
                val_img = img[y1:y2, x1:x2]
                # Resize the image into 227x227
                val_resize = cv2.resize(val_img, (227,227), cv2.INTER_AREA)
                # Add to train_X list as well as their corresponding label
                val_resize = val_resize.astype('float32')
                val_resize = val_resize/255
                val_X.append(val_resize)
                val_y.append(label_dict['background'])
                P_val.append([0,0,0,0])
                G_val.append([0,0,0,0])
                neg_sample += 1
            else:
                # If by chance both we got 10 positive sample as well as negative sample
                # No need to go further to select next region. Terminate here and
                # Move on to the next region
                if pos_sample > 5 and neg_sample > 5:
                    break
                # If one of the has to be filled, carrying on.
                else:
                    continue
```

```
C:\Anaconda3\envs\tf-gpu\lib\site-packages\ipykernel_launcher.py:18: TqdmDeprecationWarning: This funct
ion will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

In [6]:

```
import gc # to collect garbage after applying del operation to free up the space in RAM
```

In [7]:

```
# Store 'P' and 'G' value which will used in Predicting BB regression problem
P_val = np.array(P_val)
G_val = np.array(G_val)
np.save('P_val',P_val)
np.save('G_val',G_val)

del P_val, G_val
gc.collect() # Collect garbage when del operation perform in previous line to free up the RAM
```

Out[7]:

```
44
```

In [8]:

```
# Convert into numpy array
val_y = np.array(val_y)

# Save val label into npy format
np.save('val_y',val_y)
del val_y
gc.collect()
```

Out[8]:

```
20
```

In [9]:

```
# Convert into numpy array
val_X = np.array(val_X)

# Save val images performed ROI using SelectiveSearchAlgorithm into npy format
np.save('val_X',val_X)
```

Kindly restart the kernel to free up the space.

And make sure compile first 3 cells under the `Feature Extraction` section.

In [4]:

```
import tqdm # To display the progress bar

# Load the val images performed ROI using SelectiveSearchAlgorithm
val_X = np.load('val_X.npy')

# Find 4096 image feature extraction from pretrained Alexnet for each image
val_feat = []
for i in tqdm.tqdm_notebook(range(val_X.shape[0])):
    val_feat.append(list(model.predict(np.expand_dims(val_X[i], axis=0))[0]))
```

```
C:\Anaconda3\envs\tf-gpu\lib\site-packages\ipykernel_launcher.py:5: TqdmDeprecationWarning: This functi
on will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """
```

In [5]:

```
del val_X

# Store the 4096 feature extraction value into npy
val_feat = np.array(val_feat)
np.save('val_X_feat',val_feat)
```

In [6]:

```
# Display some top value
print(val_feat[:10])
```

```
[[0.8762383  0.         0.         ... 0.         1.7803026  0.        ]
 [0.8762383  0.         0.         ... 0.         1.7803026  0.        ]
 [0.8762383  0.         0.         ... 0.         1.7803026  0.        ]
 ...
 [1.49502    0.         0.3817966  ... 0.42603645 0.         0.        ]
 [3.1071346  0.         1.2334569  ... 2.027348   0.         0.        ]
 [3.1114233  0.         1.2108529  ... 2.0256095  0.         0.        ]]
```

SVM and BB

Now, we have extracted and stored 4096 features of both train and val images. Let get start training for Object detection.

There will be two things to be see: `Classify Object` and `Predicted BB`

Remember! If we got classify object as background, we are not going to Predicted BB (of course) because we are not interested in background. We more interested in finding objects.

Classify Object

Let just go first one with 'Classify Object' part which is easy one.

In [2]:

```
from sklearn.svm import SVC # For classifying object
import numpy as np # To load the .npy file
```

In [5]:

```
# Load images from both train and val
train_X = np.load('train_X_feat.npy')
val_X = np.load('val_X_feat.npy')

# Load label from both train and val
train_y = np.load('train_y.npy')
val_y = np.load('val_y.npy')
```

In [6]:

```
# Display to crosscheck the dimension of the variables
train_X.shape, train_y.shape, val_X.shape, val_y.shape
```

Out[6]:

```
((10838, 4096), (10838,), (11344, 4096), (11344,))
```

In [4]:

```
# Hyperparameter value for SVM
params = {'C':[10**i for i in range(-4,5)], 'gamma': ['scale','auto']}
params
```

Out[4]:

```
{'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000],
 'gamma': ['scale', 'auto']}
```

In [5]:

```
# Store the best_param you have got
best_param = dict()

# Exactly like a GridSearch If you know.
# Iterative process over every value present in params to find best parameter possible
for c in params['C']:
    val_maxscore = 0
    for g in params['gamma']:
        # Instantiate the SVM model
        svm_clf = SVC(random_state=1, kernel='linear', C=c, gamma=g, probability=True)
        # Fit the train data
        svm_clf.fit(train_X,train_y)
        # Find the score on both train and val
        tr_score = svm_clf.score(train_X, train_y)
        val_score = svm_clf.score(val_X,val_y)
        # Update the score and params which give best parameter value.
        if val_maxscore < val_score:
            val_maxscore = val_score
            best_param['C']=c
            best_param['gamma']=g
        print('C={},gamma={} : train score={}, cv score={}'.format(c,g,tr_score,val_score))
```

```
C=0.0001,gamma=scale : train score=0.891123823583687, cv score=0.7272566995768688
C=0.0001,gamma=auto : train score=0.891123823583687, cv score=0.7272566995768688
C=0.001,gamma=scale : train score=0.9865288798671341, cv score=0.733515514809591
C=0.001,gamma=auto : train score=0.9865288798671341, cv score=0.733515514809591
C=0.01,gamma=scale : train score=0.9988927846466138, cv score=0.7304301833568406
C=0.01,gamma=auto : train score=0.9988927846466138, cv score=0.7304301833568406
C=0.1,gamma=scale : train score=0.9999077320538845, cv score=0.7303420310296191
C=0.1,gamma=auto : train score=0.9999077320538845, cv score=0.7303420310296191
C=1,gamma=scale : train score=0.9999077320538845, cv score=0.7303420310296191
C=1,gamma=auto : train score=0.9999077320538845, cv score=0.7303420310296191
C=10,gamma=scale : train score=0.9999077320538845, cv score=0.7303420310296191
C=10,gamma=auto : train score=0.9999077320538845, cv score=0.7303420310296191
C=100,gamma=scale : train score=0.9999077320538845, cv score=0.7303420310296191
C=100,gamma=auto : train score=0.9999077320538845, cv score=0.7303420310296191
C=1000,gamma=scale : train score=0.9999077320538845, cv score=0.7303420310296191
C=1000,gamma=auto : train score=0.9999077320538845, cv score=0.7303420310296191
C=10000,gamma=scale : train score=0.9999077320538845, cv score=0.7303420310296191
C=10000,gamma=auto : train score=0.9999077320538845, cv score=0.7303420310296191
```

We will consider parameters as `C` =0.001, `gamma` ='auto' and `kernel` ='linear'.

Remember! we have used only 5 positive and 5 negative samples from an image only as per my resource is very limited and that why we are getting some same cv score. However, I just want to apply in practical implementation to understand how actually works. If you have more resources avaiable, you are free to play/modify around it!

In [7]:

```
svm_clf = SVC(random_state=1, kernel='linear', C=0.001, gamma='auto', probability=True)
svm_clf.fit(train_X,train_y)
```

Out[7]:

```
SVC(C=0.001, gamma='auto', kernel='linear', probability=True, random_state=1)
```

In [8]:

```
# Save the model
from joblib import dump
dump(svm_clf,'Classify_object.pkl')
```

Out[8]:

```
['Classify_object.pkl']
```

Predict BB (Bounding Box)

$$
\begin{aligned}
t_x &= (G_x - P_x)/P_w \\
t_y &= (G_y - P_y)/P_h \\
t_w &= \log(G_w/P_w) \\
t_h &= \log(G_h/P_h).
\end{aligned}
$$

Here `P` is region proposal BB generated by Selective Search Algorithm and `G` is ground truth BB in coordinate (x,y,w,h).

So, every value is present, just put it in equation above we get `t` (desired/target) value.

In [1]:

```
import numpy as np # To load  P  and  G  BB value.
from sklearn.linear_model import Ridge # To solve regression problem
```

In [2]:

```
# Exactly formulated as given above image
def t_BB(P,G):
    '''
    Return target value `t` and also update P and G with coordinate [x,y,w,h] instead of [x1,y1,x2,y2]

    Parameters:
    P: Predicted BB of coordinates [x1,y1,x2,y2] as list
    G: Ground truth BB of coordinates [x1,y1,x2,y2] as list

    Return:
    G: Ground-truth BB of coordinates (x,y,w,h)
    P: Proposal BB of coordinates (x,y,w,h)
    t: target BB of coordinates (x,y,w,h)
    '''

    # Ground truth width and height
    # Width: (x2 - x1); Height: (y2 - y1)
    G_w = G[2] - G[0]
    G_h = G[3] - G[1]

    # Similarly for Proposal P width and height
    P_w = P[2] - P[0]
    P_h = P[3] - P[1]

    # Apply equation as above
    t_x = (G[0] - P[0])/P_w
    t_y = (G[1] - P[1])/P_h
    t_w = np.log(G_w/P_w)
    t_h = np.log(G_h/P_h)

    return [G[0],G[1],G_w,G_h],[P[0],P[1],P_w,P_h],[t_x,t_y,t_w,t_h]
```

In [3]:

```
# Load `P` and `G` from both train and val
P = np.load('P.npy')
G = np.load('G.npy')
P_val = np.load('P_val.npy')
G_val = np.load('G_val.npy')
```

In [4]:

```
# Display some value
index = 46
print('Proposal P coordinates:',P[index])
print('Ground-truth G coordinates:', G[index])
```

```
Proposal P coordinates: [ 77  46 580 423]
Ground-truth G coordinates: [ 63  45 584 458]
```

If `P` and `G` are showing all 0's value, it means the image was **background class** which we are not interested in. That why I set the value 0 in both.

In [5]:

```
# Pass the value to see the result after applying to equation
t_BB(P[index],G[index])
```

Out[5]:

```
([63, 45, 521, 413],
 [77, 46, 503, 377],
 [-0.027833001988071572,
  -0.002652519893899204,
 0.03515987165362 7724,
 0.09120240551302194])
```

In [6]:

```
# For training
# Store the target value `t`
t = []
# Update the `G` and `P` with coordinates as [x,y,w,h] instead of [x1,y1,x2,y2]
G_ = []
P_ = []
for i in range(P.shape[0]):
    if P[i][0] == 0 and P[i][1] == 0 and P[i][2] == 0 and P[i][3] == 0:
        G_.append([0,0,0,0])
        P_.append([0,0,0,0])
        t.append([0,0,0,0])
    else:
        G_temp,P_temp,t_temp = t_BB(P[i],G[i])
        G_.append(G_temp)
        P_.append(P_temp)
        t.append(t_temp)

# For val
# Store the target value `t`
t_val = []
# Update the `G` and `P` with coordinates as [x,y,w,h] instead of [x1,y1,x2,y2]
Gval_ = []
Pval_ = []
for i in range(P_val.shape[0]):
    if P_val[i][0] == 0 and P_val[i][1] == 0 and P_val[i][2] == 0 and P_val[i][3] == 0:
        Gval_.append([0,0,0,0])
        Pval_.append([0,0,0,0])
        t_val.append([0,0,0,0])
    else:
        G_temp,P_temp,t_temp = t_BB(P_val[i],G_val[i])
        Gval_.append(G_temp)
        Pval_.append(P_temp)
        t_val.append(t_temp)
```

In [7]:

```
# Convert into numpy array for both train and val
# for train
G_ = np.array(G_)
P_ = np.array(P_)
t = np.array(t)

# for val
Gval_ = np.array(Gval_)
Pval_ = np.array(Pval_)
t_val = np.array(t_val)

# Dislay the shape to cross the dimensions among them
# There are the variables we are going to use
G_.shape, P_.shape, t.shape, Gval_.shape, Pval_.shape, t_val.shape,
```

Out[7]:

```
((10838, 4), (10838, 4), (10838, 4), (11344, 4), (11344, 4), (11344, 4))
```

Objective function:

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^{\mathrm{T}} \phi_5(P^i))^2 + \lambda \left\| \hat{\mathbf{w}}_\star \right\|^2 .$$

We already computed `t` and we already have `P`. We need to find optimal `w*` from Regression model to achieve this objective function.

In [8]:

```
# Hyperparameter on alpha (or lambda) on regression problem to find the best optimal weights
for i in range(-4,5):
    ridge_reg = Ridge(alpha=10**i)
    ridge_reg.fit(P_,t)
    tr_score=ridge_reg.score(P_,t)
    val_score=ridge_reg.score(Pval_,t_val)
    print('For alpha={}: tr_loss:{}, val_loss:{}'.format(10**i,tr_score,val_score))
```

```
For alpha=0.0001: tr_loss:0.04574987015571669, val_loss:0.06188022282213512
For alpha=0.001: tr_loss:0.045749870155720046, val_loss:0.061880222821909026
For alpha=0.01: tr_loss:0.04574987015571297, val_loss:0.06188022281963129
For alpha=0.1: tr_loss:0.04574987015560669, val_loss:0.0618802227967587
For alpha=1: tr_loss:0.045749870155600475, val_loss:0.061880222569173804
For alpha=10: tr_loss:0.04574987015567705, val_loss:0.06188022029337481
For alpha=100: tr_loss:0.045749870155449096, val_loss:0.06188019753448776
For alpha=1000: tr_loss:0.04574987013628076, val_loss:0.06187996993473166
For alpha=10000: tr_loss:0.0457498682180999, val_loss:0.061877692808850054
```

I found `alpha` =10000 to be less loss value

In [9]:

```
# Keep the best paramters after hyperparameter
ridge_reg = Ridge(alpha=10000)
ridge_reg.fit(P_,t)
```

Out[9]:

```
Ridge(alpha=10000)
```

This is the predicted BB value. This is where you will compare actual BB with predicted G_hat

$$
\begin{aligned}
\hat{G}_x &= P_w d_x(P) + P_x \\
\hat{G}_y &= P_h d_y(P) + P_y \\
\hat{G}_w &= P_w \exp(d_w(P)) \\
\hat{G}_h &= P_h \exp(d_h(P)).
\end{aligned}
$$

`d*(P)` which is basically `w.P*`. And we got optimal `w` using regression model. Now we can get predicted BB value

In [10]:

```
# Exactly written same as per above image
def pred_BB(P, dP):
    '''
    Return Predicted BB value

    Parameters:
    P: Proposal BB in coordinates [x,y,w,h]
    df: Predicted value by regression model. [x,y,w,h]
    '''

    G_x = P[2]*dP[0] + P[0]
    G_y = P[3]*dP[1] + P[1]
    G_w = P[2]*np.exp(dP[2])
    G_h = P[3]*np.exp(dP[3])

    return [G_x,G_y,G_w,G_h]
```

In [11]:

```
# Perform some operation to see everything is alright?
index_1 = 1469

print('Ground Truth value:',G_[index_1])
pred_val = ridge_reg.predict(np.reshape(P_[index_1], (1,-1)))[0]
print('d*(P) from regression model:',pred_val)
print('Predict BB value:',pred_BB(P_[index_1],pred_val))
```

```
Ground Truth value: [176 115 362 185]
d*(P) from regression model: [ 0.00195473 -0.00975921 -0.01123311  0.0232984 ]
Predict BB value: [177.695884471663, 120.52635889862682, 352.023389376763, 154.5593614230331]
```

Very close!! But still worked it with small positive and negative samples.

In [12]:

```
# Save the model
from joblib import dump
dump(ridge_reg,'BB_regression.pkl')
```

Out[12]:

```
['BB_regression.pkl']
```

Point to be Remember

1. BB_regression.joblib model feed the input with coordinates contain in the [x,y,w,h] format (NOT [x1,y1,x2,y2])

In [ ]:

```
```