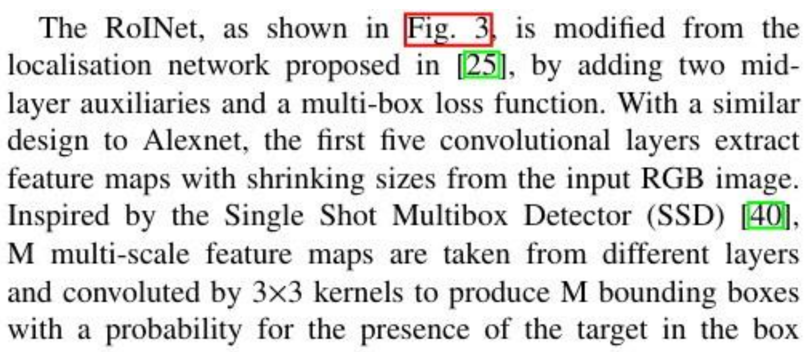
Phase 1- Final Report

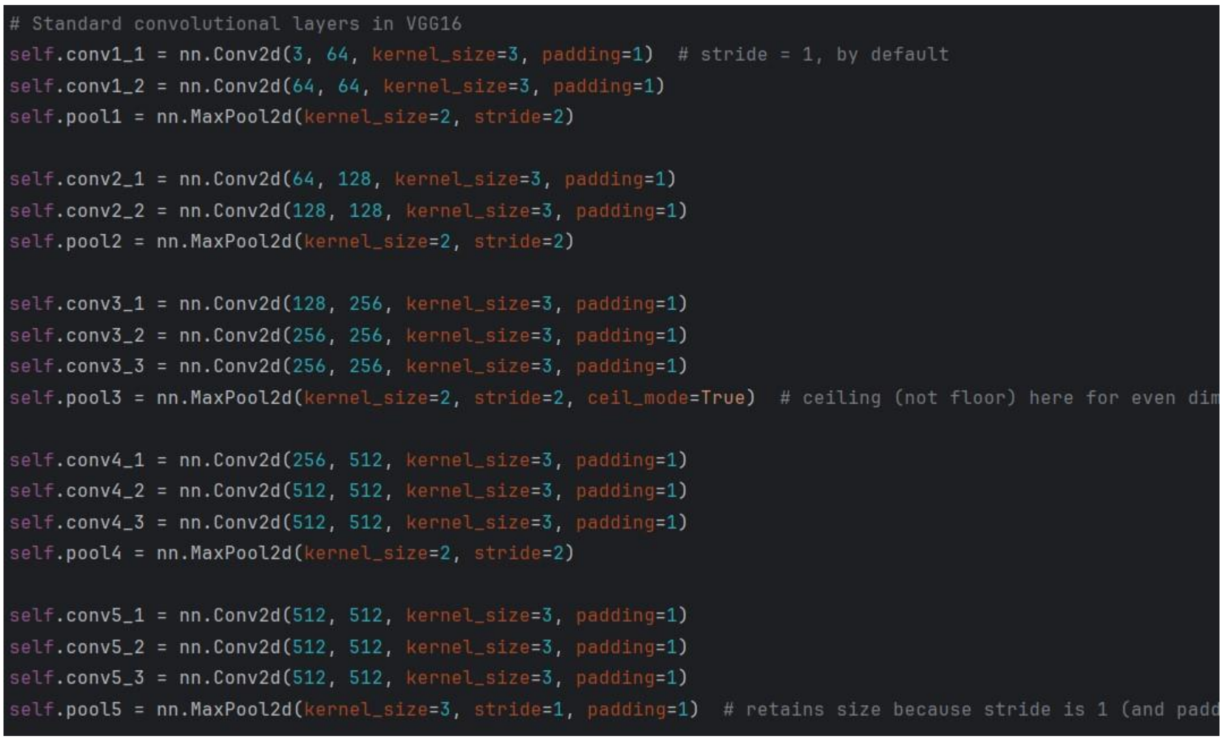
Starting from the **literature** you have given me to read and implement and on which this whole thing is based on :



The RoINet, as shown in Fig. 3, is modified from the localisation network proposed in [25], by adding two mid-layer auxiliaries and a multi-box loss function. With a similar design to Alexnet, the first five convolutional layers extract feature maps with shrinking sizes from the input RGB image. Inspired by the Single Shot Multibox Detector (SSD) [40], M multi-scale feature maps are taken from different layers and convoluted by 3×3 kernels to produce M bounding boxes with a probability for the presence of the target in the box

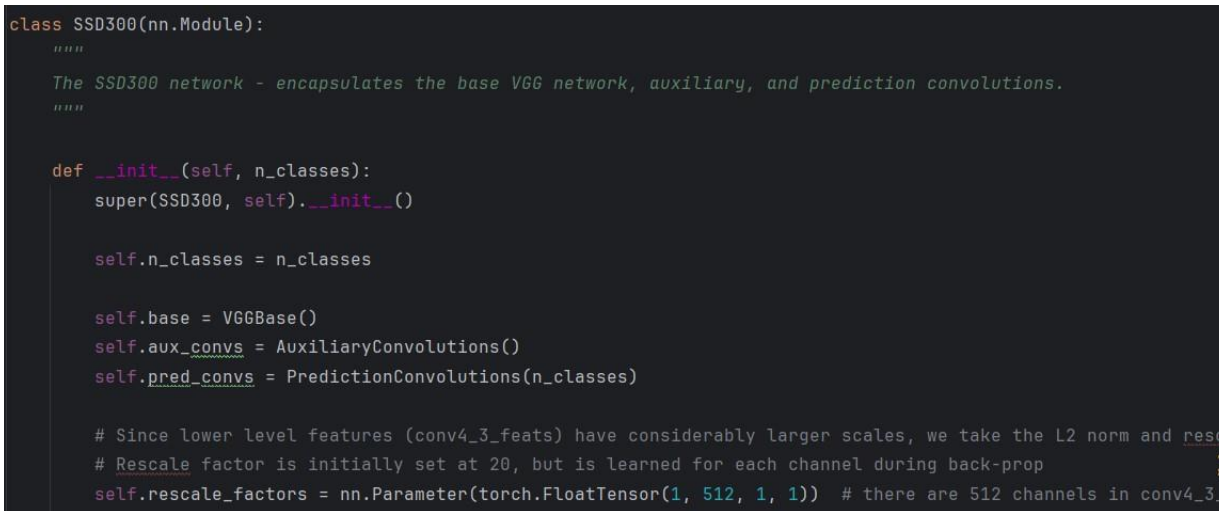
So I have implemented this using **AlexNet Architecture** + **Single Shot Detector (SSD)** with Multibox Loss Function . What basically I did is I take the **complex images** i.e images that consist of multiple objects in background and tested by algorithm on that because if its work for that it will eventually get adapted to **BONE** background so following code snippet will show my implementation..

First the main **model.py** snippet

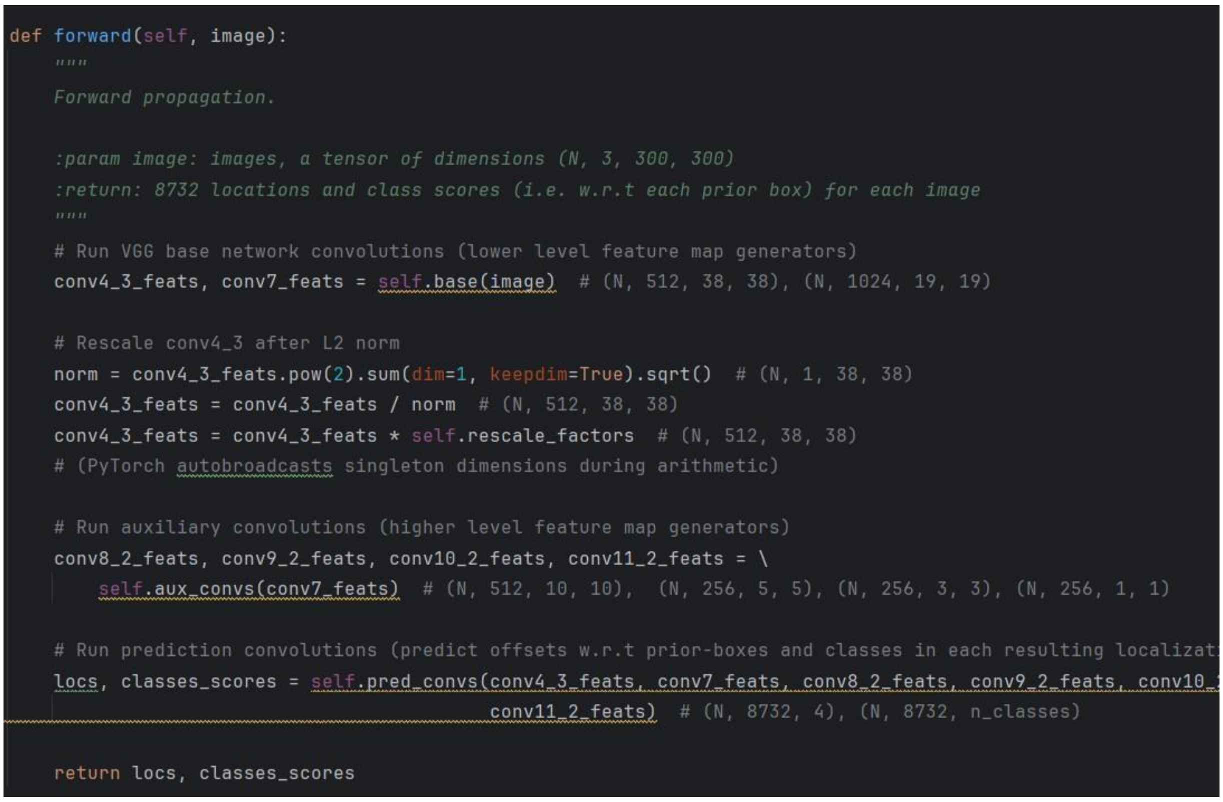


```
# Standard convolutional layers in VGG16
self.conv1_1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)  # stride = 1, by default
self.conv1_2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv2_1 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
self.conv2_2 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv3_1 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
self.conv3_2 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
self.conv3_3 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)  # ceiling (not floor) here for even dim

self.conv4_1 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
self.conv4_2 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
self.conv4_3 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv5_1 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
self.conv5_2 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
self.conv5_3 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
self.pool5 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)  # retains size because stride is 1 (and padd
```

SSD implementation in model.py

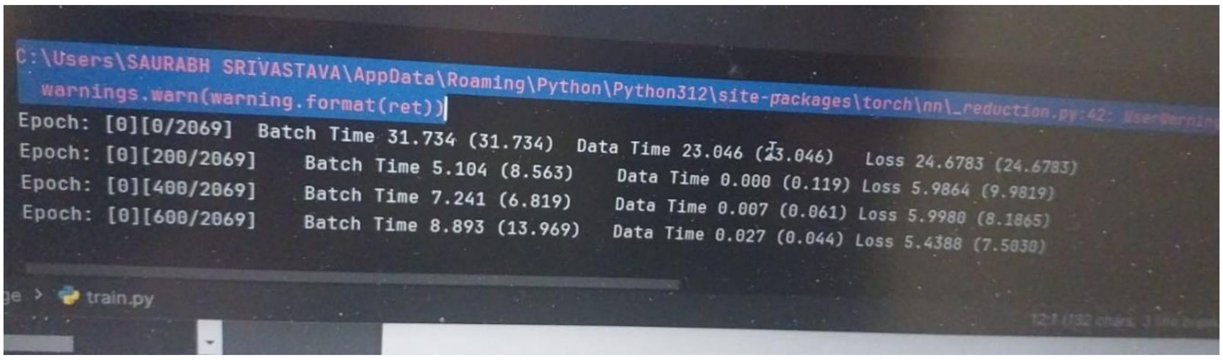


```
class SSD300(nn.Module):
    """
    The SSD300 network - encapsulates the base VGG network, auxiliary, and prediction convolutions.
    """

    def __init__(self, n_classes):
        super(SSD300, self).__init__()

        self.n_classes = n_classes

        self.base = VGGBase()
        self.aux_convs = AuxiliaryConvolutions()
        self.pred_convs = PredictionConvolutions(n_classes)

        # Since lower level features (conv4_3_feats) have considerably larger scales, we take the L2 norm and resc
        # Rescale factor is initially set at 20, but is learned for each channel during back-prop
        self.rescale_factors = nn.Parameter(torch.FloatTensor(1, 512, 1, 1))  # there are 512 channels in conv4_3_
```

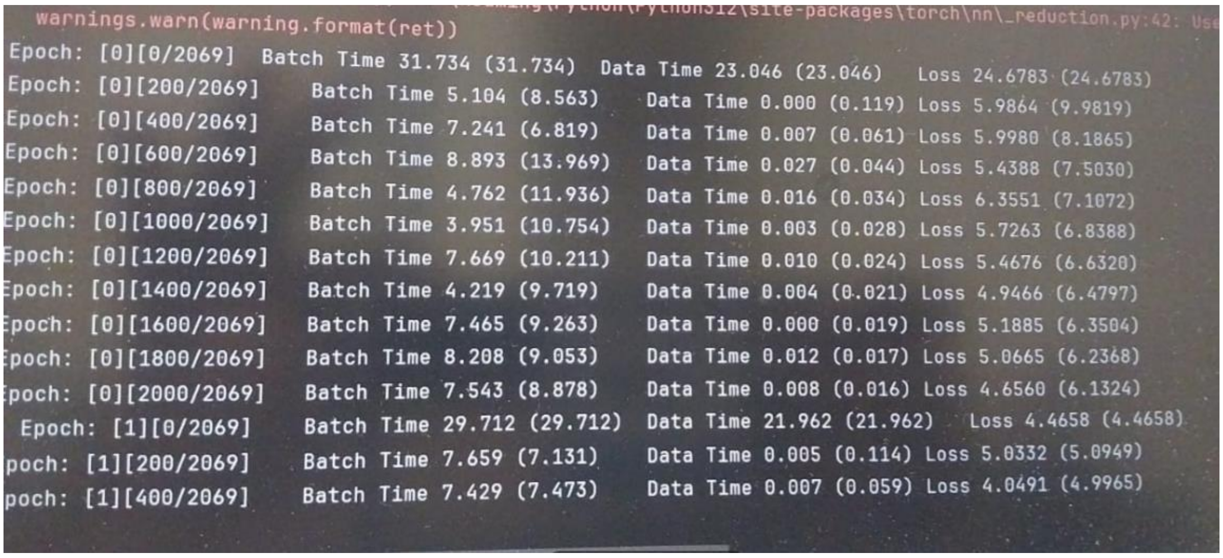


```
def forward(self, image):
    """
    Forward propagation.

    :param image: images, a tensor of dimensions (N, 3, 300, 300)
    :return: 8732 locations and class scores (i.e. w.r.t each prior box) for each image
    """
    # Run VGG base network convolutions (lower level feature map generators)
    conv4_3_feats, conv7_feats = self.base(image)  # (N, 512, 38, 38), (N, 1024, 19, 19)

    # Rescale conv4_3 after L2 norm
    norm = conv4_3_feats.pow(2).sum(dim=1, keepdim=True).sqrt()  # (N, 1, 38, 38)
    conv4_3_feats = conv4_3_feats / norm  # (N, 512, 38, 38)
    conv4_3_feats = conv4_3_feats * self.rescale_factors  # (N, 512, 38, 38)
    # (PyTorch autobroadcasts singleton dimensions during arithmetic)

    # Run auxiliary convolutions (higher level feature map generators)
    conv8_2_feats, conv9_2_feats, conv10_2_feats, conv11_2_feats = \
        self.aux_convs(conv7_feats)  # (N, 512, 10, 10),  (N, 256, 5, 5), (N, 256, 3, 3), (N, 256, 1, 1)

    # Run prediction convolutions (predict offsets w.r.t prior-boxes and classes in each resulting localizat
    locs, classes_scores = self.pred_convs(conv4_3_feats, conv7_feats, conv8_2_feats, conv9_2_feats, conv10_
                                           conv11_2_feats)  # (N, 8732, 4), (N, 8732, n_classes)

    return locs, classes_scores
```

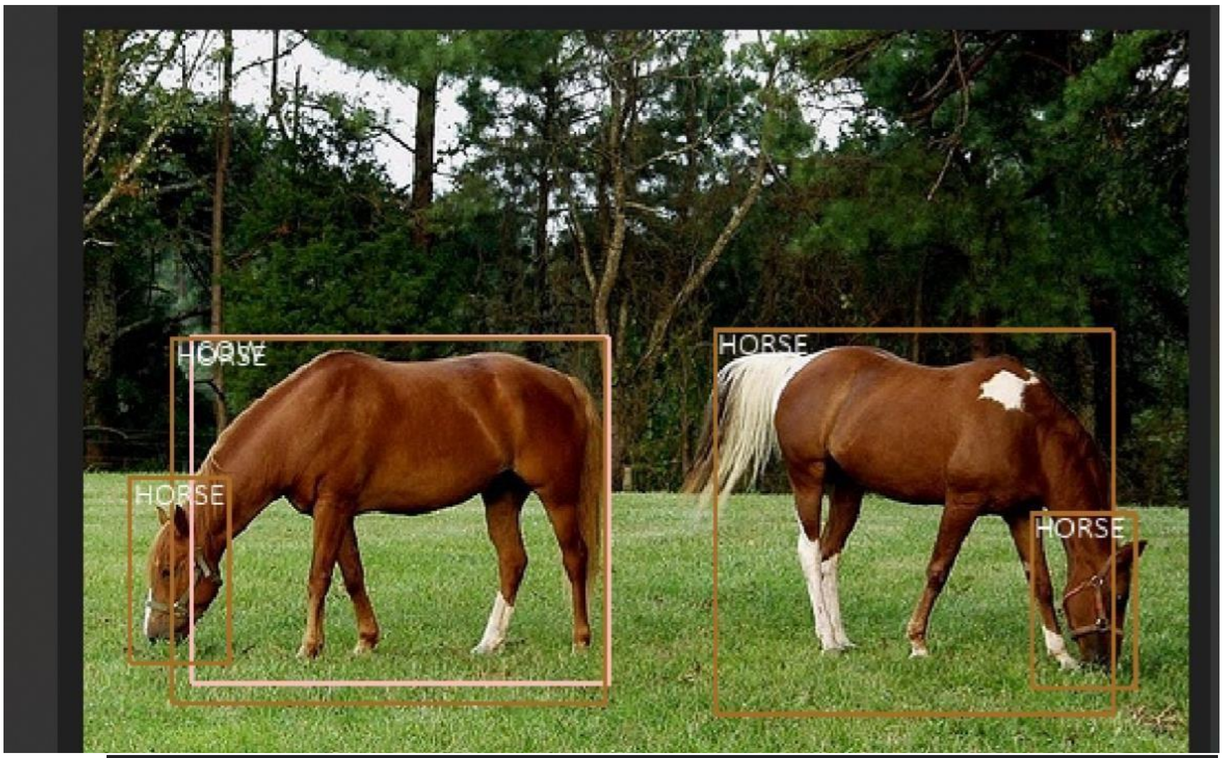
Some Results:

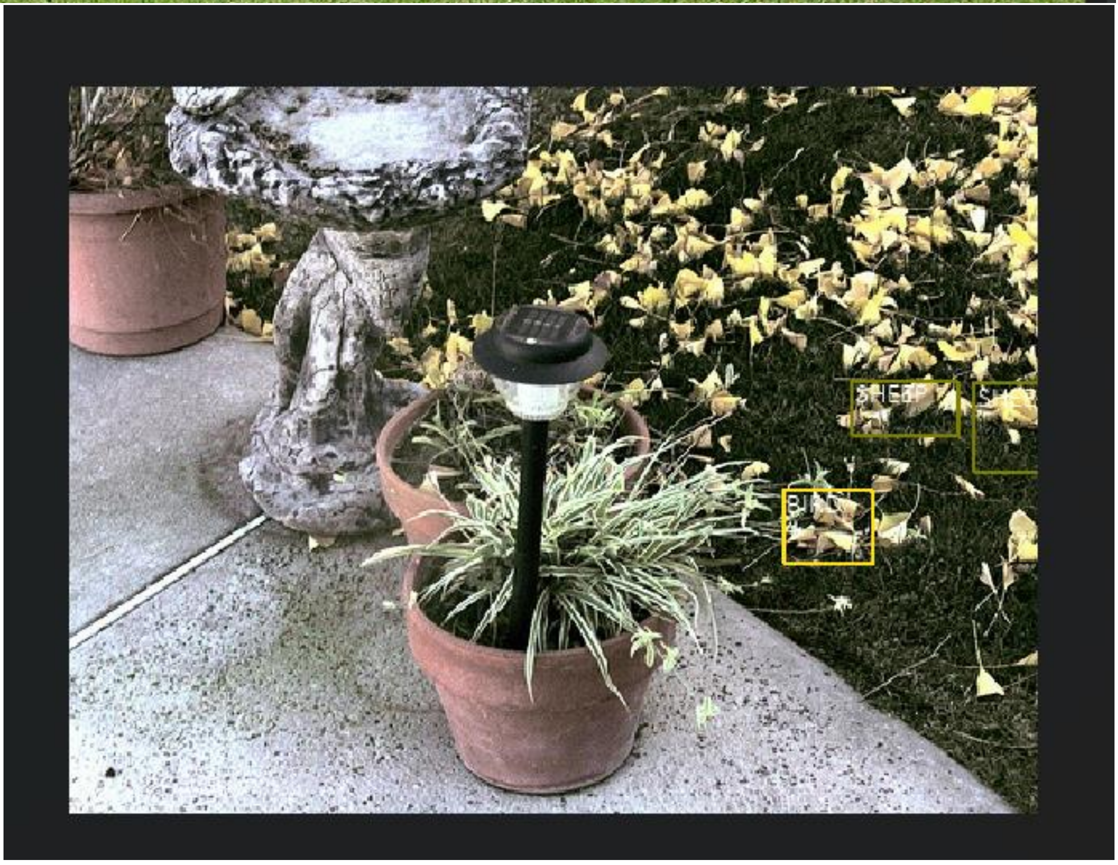


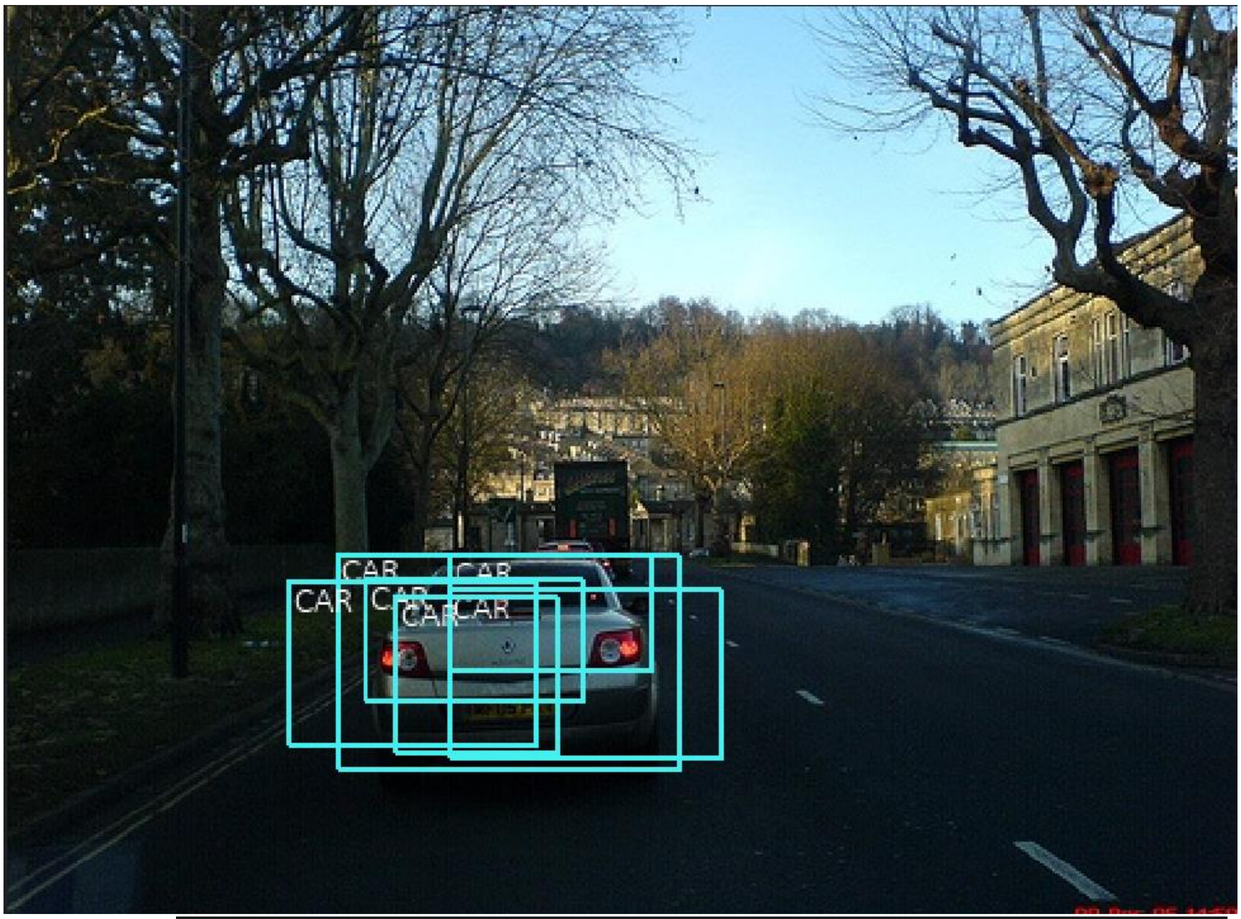
```
C:\Users\SAURABH SRIVASTAVA\AppData\Roaming\Python\Python312\site-packages\torch\nn\_reduction.py:42: UserWarning
  warnings.warn(warning.format(ret))
Epoch: [0][0/2069]  Batch Time 31.734 (31.734)  Data Time 23.046 (23.046)   Loss 24.6783 (24.6783)
Epoch: [0][200/2069]    Batch Time 5.104 (8.563)    Data Time 0.000 (0.119) Loss 5.9864 (9.9819)
Epoch: [0][400/2069]    Batch Time 7.241 (6.819)    Data Time 0.007 (0.061) Loss 5.9980 (8.1865)
Epoch: [0][600/2069]    Batch Time 8.893 (13.969)   Data Time 0.027 (0.044) Loss 5.4388 (7.5030)
```

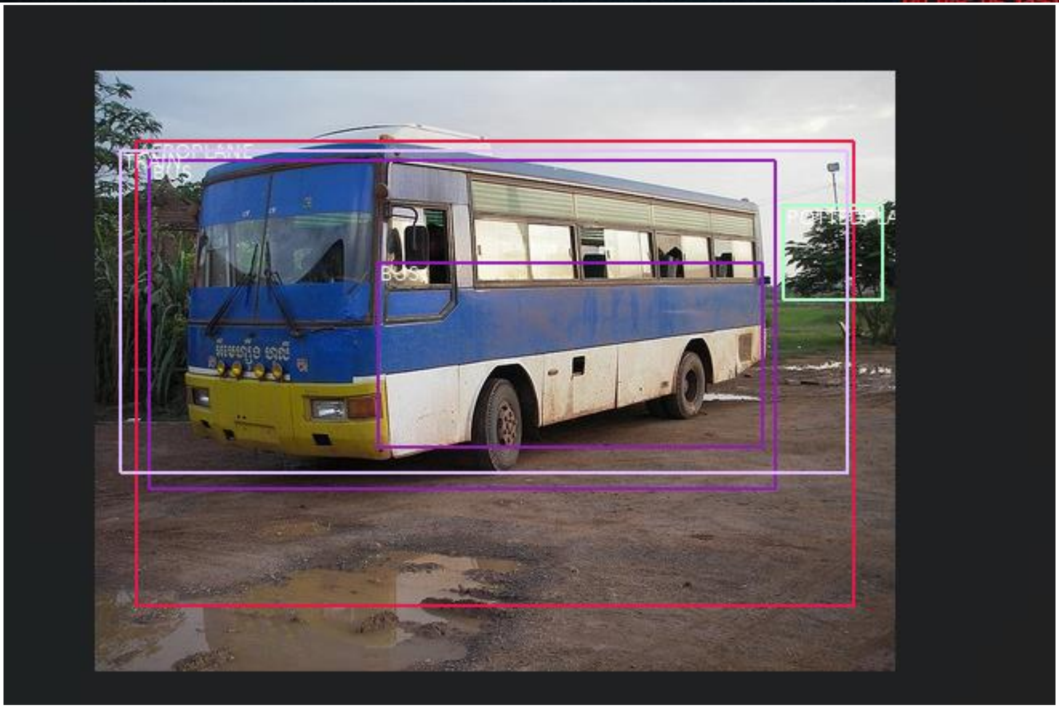


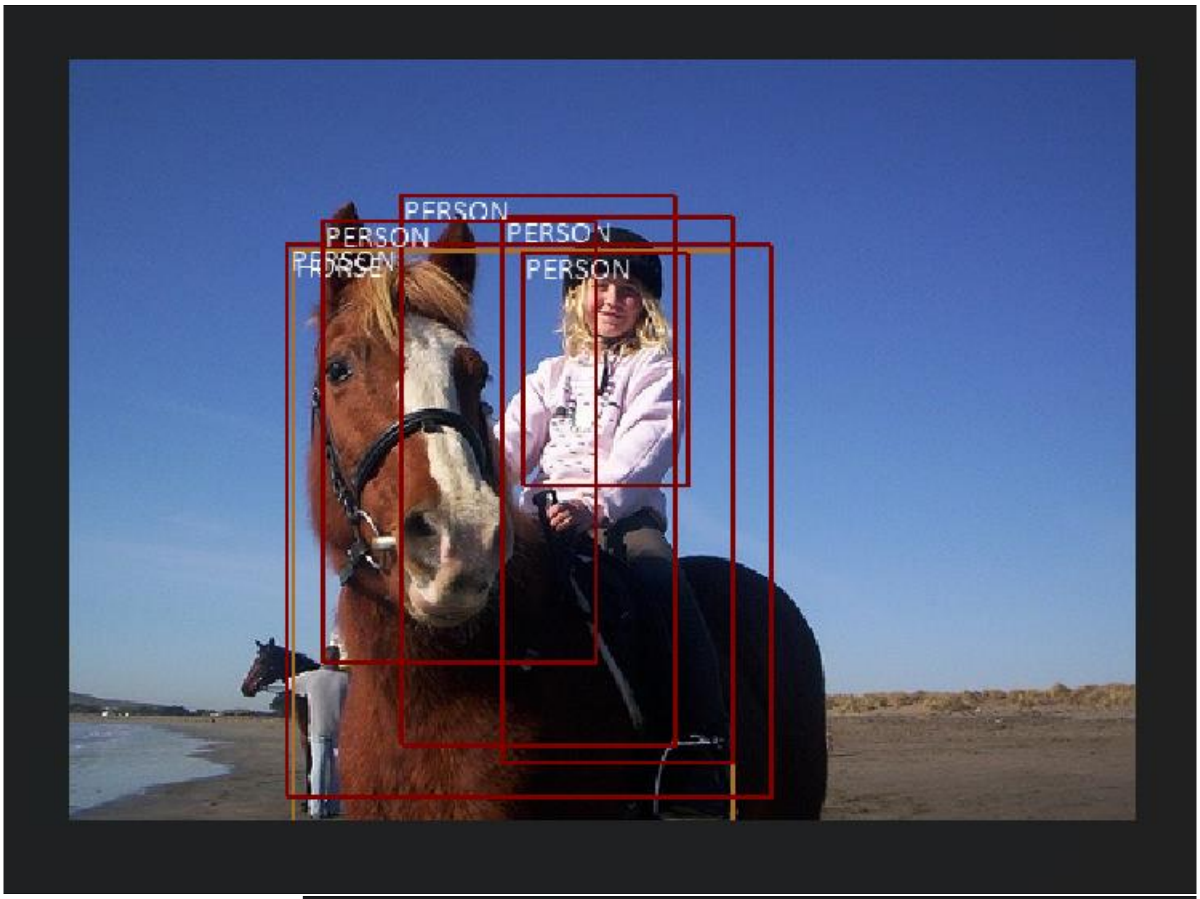
```
  warnings.warn(warning.format(ret))                       site-packages\torch\nn\_reduction.py:42: Use
Epoch: [0][0/2069]  Batch Time 31.734 (31.734)  Data Time 23.046 (23.046)   Loss 24.6783 (24.6783)
Epoch: [0][200/2069]    Batch Time 5.104 (8.563)    Data Time 0.000 (0.119) Loss 5.9864 (9.9819)
Epoch: [0][400/2069]    Batch Time 7.241 (6.819)    Data Time 0.007 (0.061) Loss 5.9980 (8.1865)
Epoch: [0][600/2069]    Batch Time 8.893 (13.969)   Data Time 0.027 (0.044) Loss 5.4388 (7.5030)
Epoch: [0][800/2069]    Batch Time 4.762 (11.936)   Data Time 0.016 (0.034) Loss 6.3551 (7.1072)
Epoch: [0][1000/2069]   Batch Time 3.951 (10.754)   Data Time 0.003 (0.028) Loss 5.7263 (6.8388)
Epoch: [0][1200/2069]   Batch Time 7.669 (10.211)   Data Time 0.010 (0.024) Loss 5.4676 (6.6320)
Epoch: [0][1400/2069]   Batch Time 4.219 (9.719)    Data Time 0.004 (0.021) Loss 4.9466 (6.4797)
Epoch: [0][1600/2069]   Batch Time 7.465 (9.263)    Data Time 0.000 (0.019) Loss 5.1885 (6.3504)
Epoch: [0][1800/2069]   Batch Time 8.208 (9.053)    Data Time 0.012 (0.017) Loss 5.0665 (6.2368)
Epoch: [0][2000/2069]   Batch Time 7.543 (8.878)    Data Time 0.008 (0.016) Loss 4.6560 (6.1324)
 Epoch: [1][0/2069]     Batch Time 29.712 (29.712)  Data Time 21.962 (21.962)   Loss 4.4658 (4.4658)
poch: [1][200/2069]     Batch Time 7.659 (7.131)    Data Time 0.005 (0.114) Loss 5.0332 (5.0949)
poch: [1][400/2069]     Batch Time 7.429 (7.473)    Data Time 0.007 (0.059) Loss 4.0491 (4.9965)
```


HORSE
HORSE
HORSE
HORSE
HORSE


SHEEP
BIRD

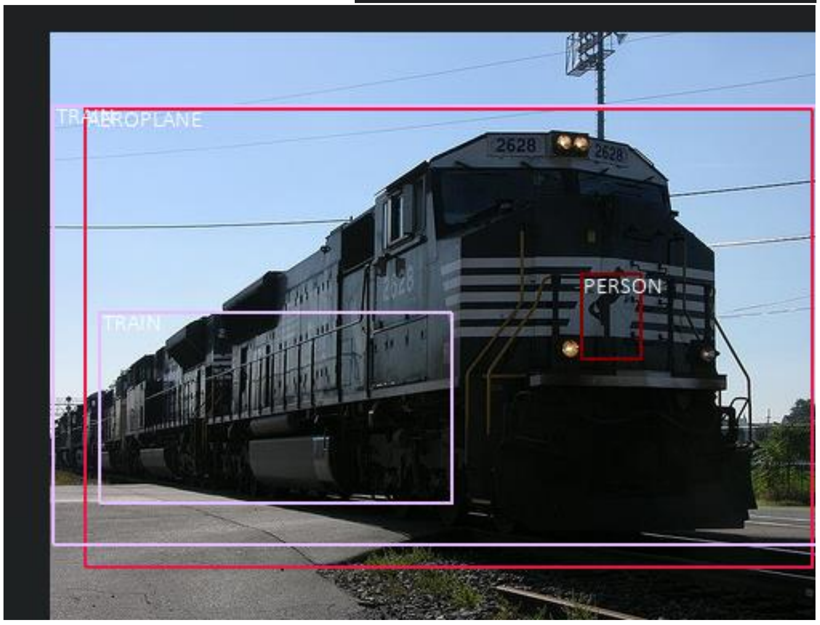

CAR
CAR
CAR
CAR
CAR
CAR

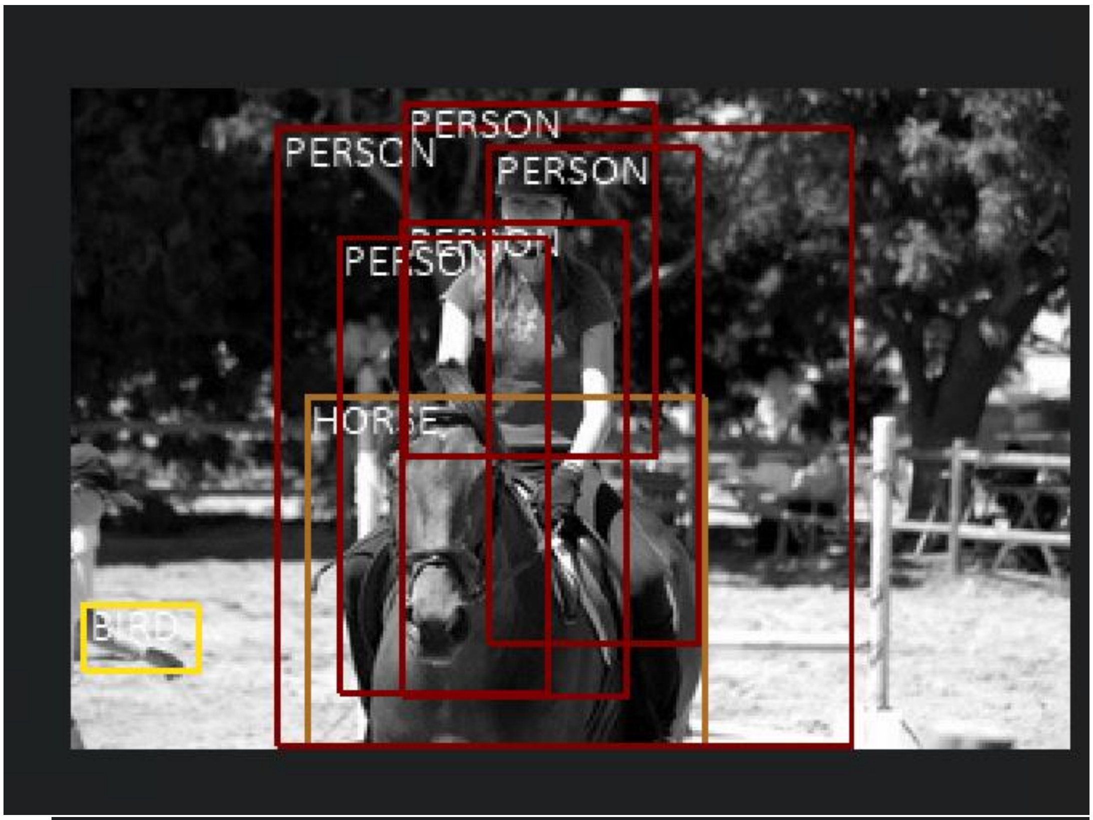

AEROPLANE
TRAIN
BUS
BUS

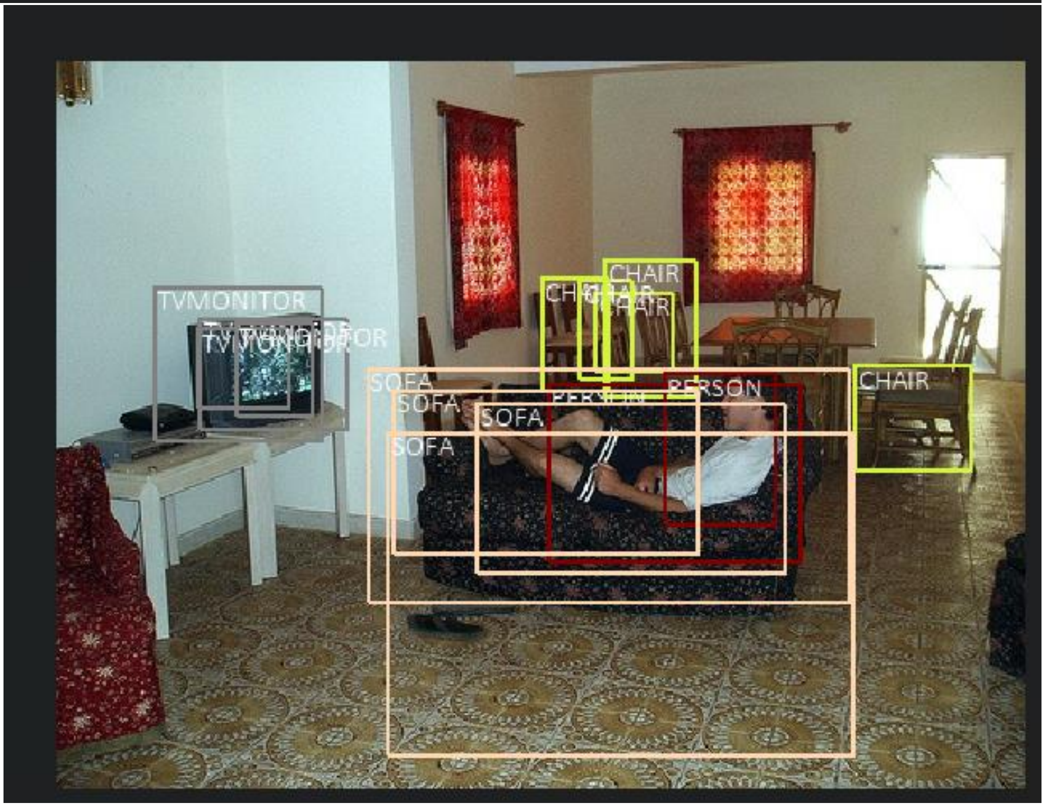

PERSON
PERSON
PERSON
PERSON
HORSE
PERSON

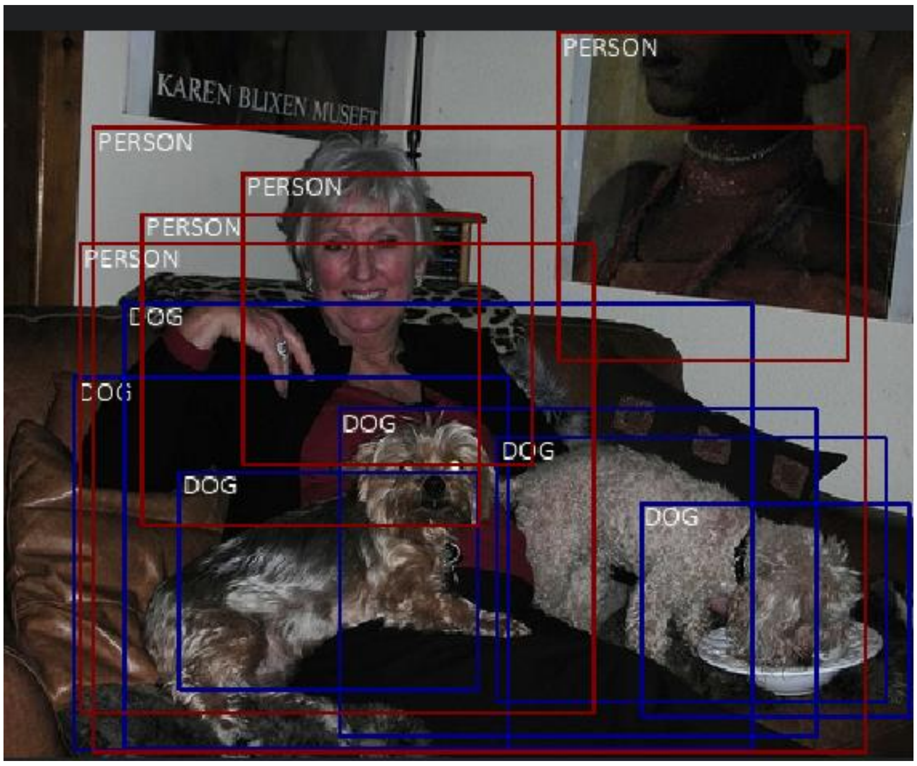
PERSON
POTTEDPLANT
SHEEP
BOTTLE

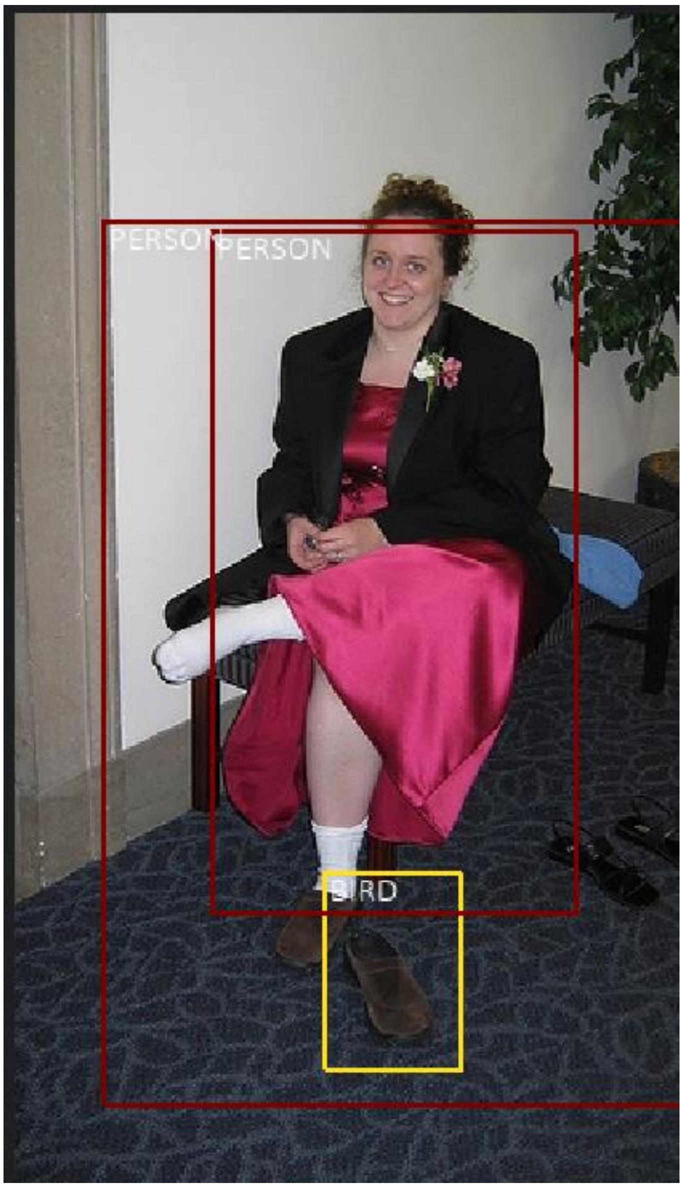
MOTORBIKE

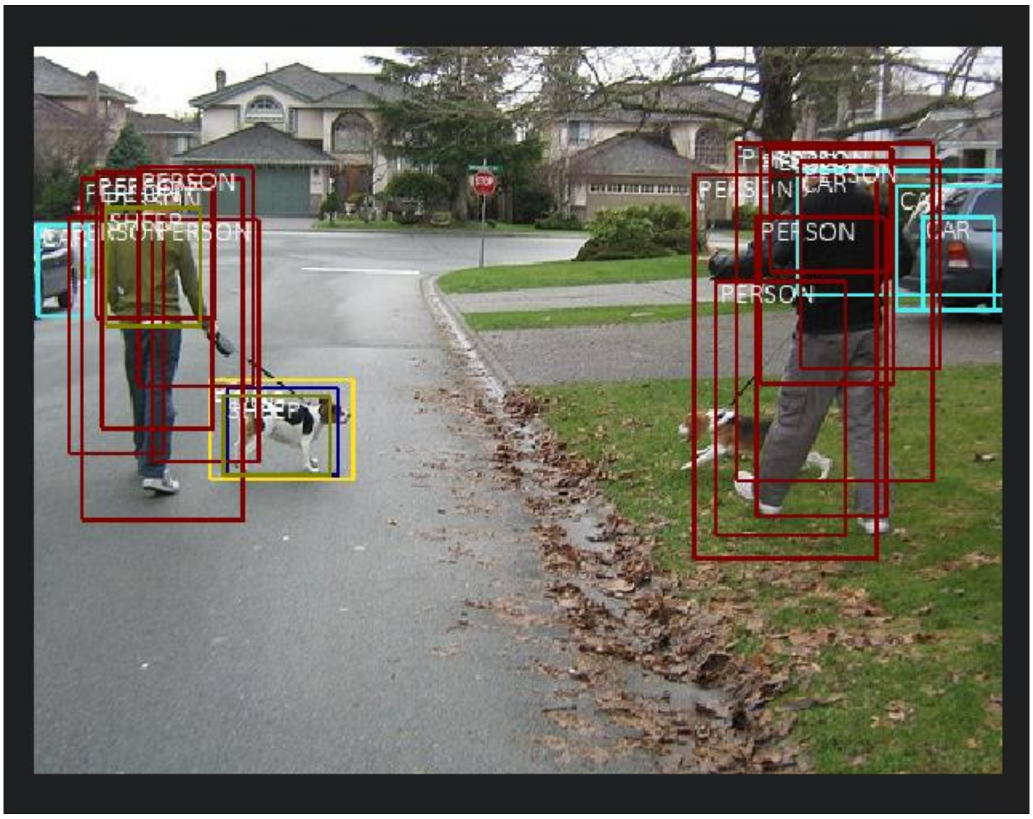

TRAIN
AEROPLANE
PERSON
TRAIN
2628


PERSON
PERSON
PERSON
PERSON
PERSON
HORSE
BIRD


TVMONITOR
TVMONITOR
CHAIR
CHAIR
CHAIR
CHAIR
CHAIR
SOFA
SOFA
SOFA
SOFA
PERSON
PERSON


KAREN BLIXEN MUSEET
PERSON
PERSON
PERSON
PERSON
PERSON
DOG
DOG
DOG
DOG
DOG
DOG

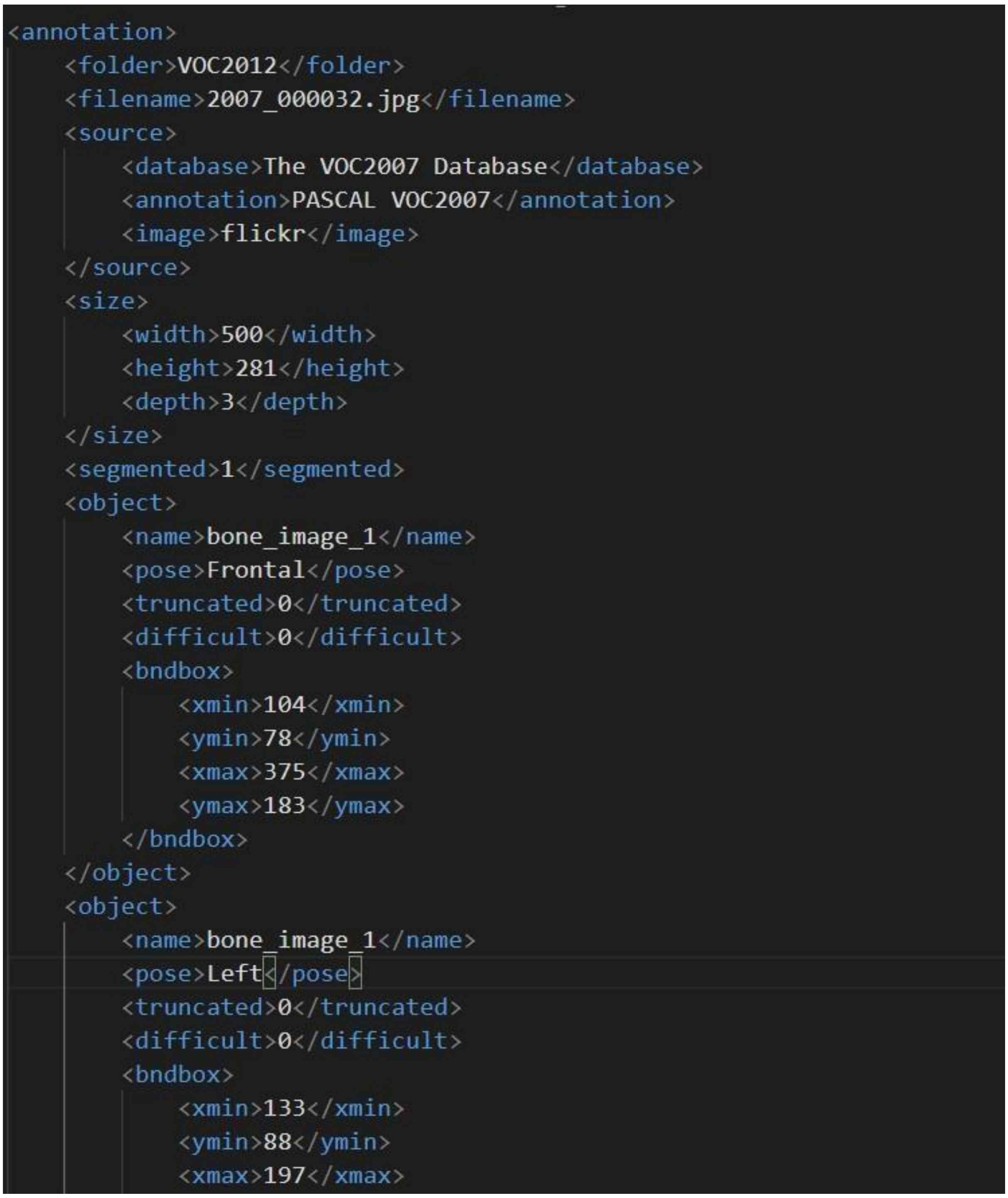

PERSON
PERSON
BIRD



We Can see in the images that identification of object is not that particular or we can say its not that accurate but the fact is that it’s not get trained fully as It has total 2000 epochs and its only trained for 2 epochs so if we allow it to get trained it to 2000 epoch accuracy will be greater than 95 percent .

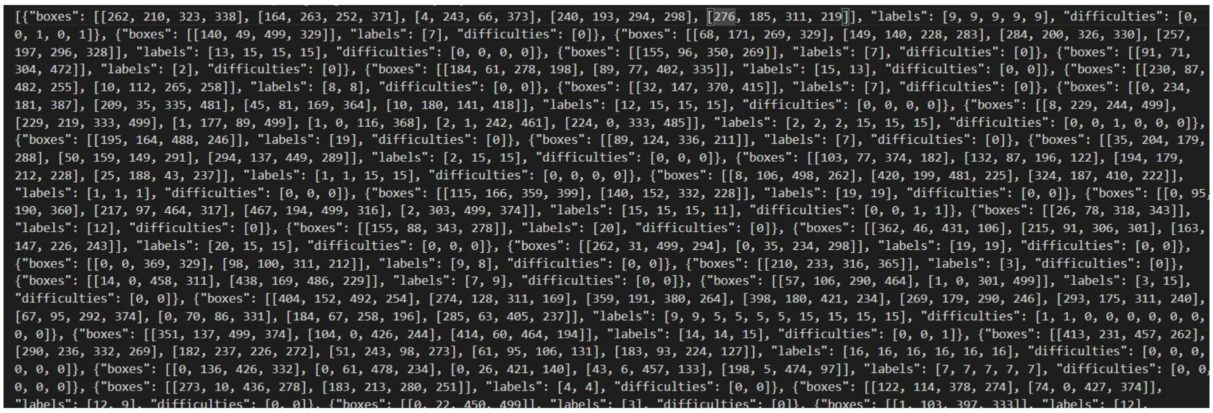
Now further moving to creation of Dataset

I have made the algorithm that cam make bounding boxes around any complex image but now the task comes to make a dataset of bone and use that Algorithm to predict RoI around that .
For that I have saw videos how to use that Intel depth camera and the kind of input my Algorithm is taking is lil complex , I'll explain it line by line :
Input image must have 4 coordinates and a difficulty binary criteria , 0 if it's roi is easy to predict and 1 if it's not easy it will be depend how complex your background.
Here is the snippet of xml file , also I am troubling it how to use the camera like it will give me only one coordinate , I have put another coordinate randomly , so I have to figure out how to use this camera in great way possible .



```
<annotation>
    <folder>VOC2012</folder>
    <filename>2007_000032.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
    </source>
    <size>
        <width>500</width>
        <height>281</height>
        <depth>3</depth>
    </size>
    <segmented>1</segmented>
    <object>
        <name>bone_image_1</name>
        <pose>Frontal</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>104</xmin>
            <ymin>78</ymin>
            <xmax>375</xmax>
            <ymax>183</ymax>
        </bndbox>
    </object>
    <object>
        <name>bone_image_1</name>
        <pose>Left</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>133</xmin>
            <ymin>88</ymin>
            <xmax>197</xmax>
```

After this annotation file , there a file name as lable map which will contain all the similar type of object's coordinate as well as difficulty but the camera is giving me 3d coordinates only but as you can see we have to give 4 coordinates giving the random 4th coordinates is not gonna work so thinking about that alternative is also a task that ill tackle down this week .



```
[{"boxes": [[262, 210, 323, 338], [164, 263, 252, 371], [4, 243, 66, 373], [240, 193, 294, 298], [276, 185, 311, 219]], "labels": [9, 9, 9, 9, 9], "difficulties": [0,
0, 1, 0, 1]}, {"boxes": [[140, 49, 499, 329]], "labels": [7], "difficulties": [0]}, {"boxes": [[68, 171, 269, 329], [149, 140, 228, 283], [284, 200, 326, 330], [257,
197, 296, 328]], "labels": [13, 15, 15, 15], "difficulties": [0, 0, 0, 0]}, {"boxes": [[155, 96, 350, 269]], "labels": [7], "difficulties": [0]}, {"boxes": [[91, 71,
304, 472]], "labels": [2], "difficulties": [0]}, {"boxes": [[184, 61, 278, 198], [89, 77, 402, 335]], "labels": [15, 13], "difficulties": [0, 0]}, {"boxes": [[230, 87,
482, 255], [10, 112, 265, 258]], "labels": [8, 8], "difficulties": [0, 0]}, {"boxes": [[32, 147, 370, 415]], "labels": [7], "difficulties": [0]}, {"boxes": [[0, 234,
181, 387], [209, 35, 335, 481], [45, 81, 169, 364], [10, 180, 141, 418]], "labels": [12, 15, 15, 15], "difficulties": [0, 0, 0, 0]}, {"boxes": [[8, 229, 244, 499],
[229, 219, 333, 499], [1, 177, 89, 499], [1, 0, 116, 368], [2, 1, 242, 461], [224, 0, 333, 485]], "labels": [2, 2, 2, 15, 15, 15], "difficulties": [0, 0, 1, 0, 0, 0]},
{"boxes": [[195, 164, 488, 246]], "labels": [19], "difficulties": [0]}, {"boxes": [[89, 124, 336, 211]], "labels": [7], "difficulties": [0]}, {"boxes": [[35, 204, 179,
288], [50, 159, 149, 291], [294, 137, 449, 289]], "labels": [2, 15, 15], "difficulties": [0, 0, 0]}, {"boxes": [[103, 77, 374, 182], [132, 87, 196, 122], [194, 179,
212, 228], [25, 188, 43, 237]], "labels": [1, 1, 15, 15], "difficulties": [0, 0, 0, 0]}, {"boxes": [[8, 106, 498, 262], [420, 199, 481, 225], [324, 187, 410, 222]],
"labels": [1, 1, 1], "difficulties": [0, 0, 0]}, {"boxes": [[115, 166, 359, 399], [140, 152, 332, 228]], "labels": [19, 19], "difficulties": [0, 0]}, {"boxes": [[0, 95,
190, 360], [217, 97, 464, 317], [467, 194, 499, 316], [2, 303, 499, 374]], "labels": [15, 15, 15, 11], "difficulties": [0, 0, 1, 1]}, {"boxes": [[26, 78, 318, 343]],
"labels": [12], "difficulties": [0]}, {"boxes": [[155, 88, 343, 278]], "labels": [20], "difficulties": [0]}, {"boxes": [[362, 46, 431, 106], [215, 91, 306, 301], [163,
147, 226, 243]], "labels": [20, 15, 15], "difficulties": [0, 0, 0]}, {"boxes": [[262, 31, 499, 294], [0, 35, 234, 298]], "labels": [19, 19], "difficulties": [0, 0]},
{"boxes": [[0, 0, 369, 329], [98, 100, 311, 212]], "labels": [9, 8], "difficulties": [0, 0]}, {"boxes": [[210, 233, 316, 365]], "labels": [3], "difficulties": [0]},
{"boxes": [[14, 0, 458, 311], [438, 169, 486, 229]], "labels": [7, 9], "difficulties": [0, 0]}, {"boxes": [[57, 106, 290, 464], [1, 0, 301, 499]], "labels": [3, 15],
"difficulties": [0, 0]}, {"boxes": [[404, 152, 492, 254], [274, 128, 311, 169], [359, 191, 380, 264], [398, 180, 421, 234], [269, 179, 290, 246], [293, 175, 311, 240],
[67, 95, 292, 374], [0, 70, 86, 331], [184, 67, 258, 196], [285, 63, 405, 237]], "labels": [9, 9, 5, 5, 5, 5, 15, 15, 15, 15], "difficulties": [1, 1, 0, 0, 0, 0, 0, 0,
0, 0]}, {"boxes": [[351, 137, 499, 374], [104, 0, 426, 244], [414, 60, 464, 194]], "labels": [14, 14, 15], "difficulties": [0, 0, 1]}, {"boxes": [[413, 231, 457, 262],
[290, 236, 332, 269], [182, 237, 226, 272], [51, 243, 98, 273], [61, 95, 106, 131], [183, 93, 224, 127]], "labels": [16, 16, 16, 16, 16, 16], "difficulties": [0, 0, 0,
0, 0, 0]}, {"boxes": [[0, 136, 426, 332], [0, 61, 478, 234], [0, 26, 421, 140], [43, 6, 457, 133], [198, 5, 474, 97]], "labels": [7, 7, 7, 7, 7], "difficulties": [0, 0,
0, 0, 0]}, {"boxes": [[273, 10, 436, 278], [183, 213, 280, 251]], "labels": [4, 4], "difficulties": [0, 0]}, {"boxes": [[122, 114, 378, 274], [74, 0, 427, 374]],
"labels": [12, 9], "difficulties": [0, 0]}, {"boxes": [[0, 22, 450, 499]], "labels": [3], "difficulties": [0]}, {"boxes": [[1, 103, 397, 333]], "labels": [12],
```