YOLOv1 – From paper to implementation

Dataset:

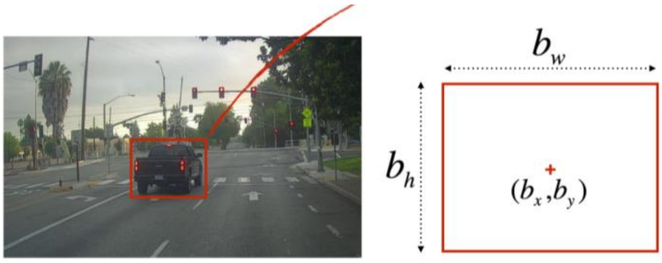
We take dog cat data for classification and localization through YOLOv1.

Image: 

Label: **0 0.51315 0.715 0.3120300 0.205**

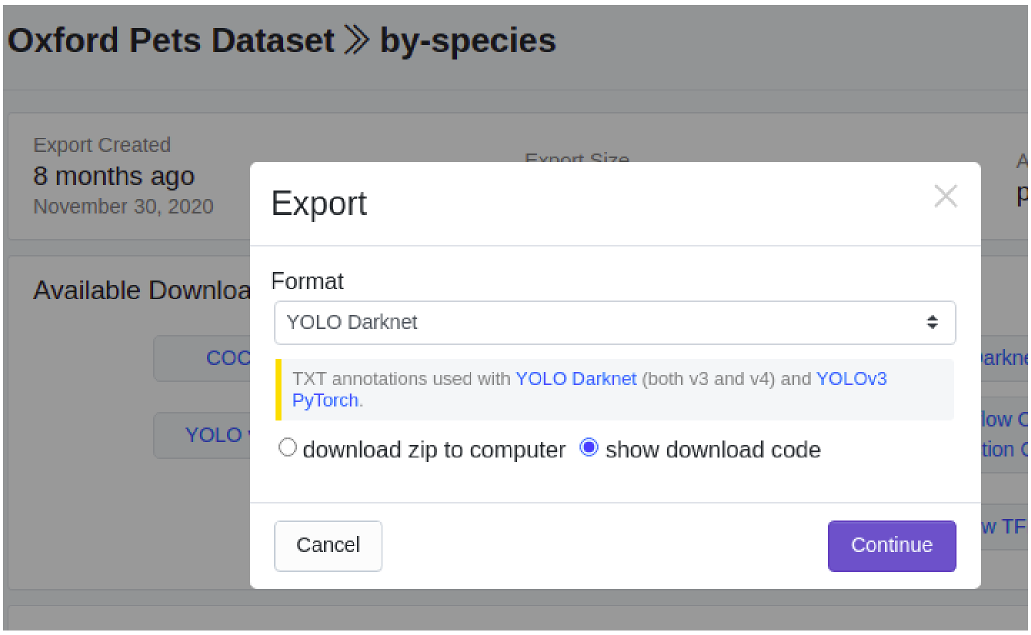
[class, centroid-x, centroid-y, height, width] normalized with image size.

$$y = (p_c, b_x, b_y, b_h, b_w)$$



Download the dataset:

Go to the link: https://public.roboflow.com/object-detection/oxford-pets/2



Export the dataset of image and labels (in YOLO Darknet format), you will get an url if show download code is selected.

Now, download the code by executing following command:

```
"""Download Cat-Dog dataset"""
!curl -L ---paste dataset url here--- > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

The dataset will be downloaded as follows:

```
Streaming output truncated to the last 5000 lines.
 extracting: train/Sphynx_205_jpg.rf.f7c5a41130f3916deed35f974c5d1df8.txt
 extracting: train/_darknet.labels
 extracting: train/american_bulldog_102_jpg.rf.f5d69d3ce15f380432ee6418448c144a.jpg
 extracting: train/american_bulldog_102_jpg.rf.f5d69d3ce15f380432ee6418448c144a.txt
 extracting: train/american_bulldog_104_jpg.rf.bd70dd5517696c5dfac1440ef69b0a7a.jpg
 extracting: train/american_bulldog_104_jpg.rf.bd70dd5517696c5dfac1440ef69b0a7a.txt
 extracting: train/american_bulldog_105_jpg.rf.7ff447c13447606deeab8329f180fe49.jpg
 extracting: train/american_bulldog_105_jpg.rf.7ff447c13447606deeab8329f180fe49.txt
 extracting: train/american_bulldog_106_jpg.rf.d5fdd6a7aab0505920adc58d2efa1793.jpg
 extracting: train/american_bulldog_106_jpg.rf.d5fdd6a7aab0505920adc58d2efa1793.txt
 extracting: train/american_bulldog_108_jpg.rf.5a524854d95e3c261abae43faa6b2e0f.jpg
 extracting: train/american_bulldog_108_jpg.rf.5a524854d95e3c261abae43faa6b2e0f.txt
 extracting: train/american_bulldog_109_jpg.rf.c1d13dacdd6dfe7ee814a989f6fe3802.jpg
 extracting: train/american_bulldog_109_jpg.rf.c1d13dacdd6dfe7ee814a989f6fe3802.txt
 extracting: train/american_bulldog_110_jpg.rf.eea553ad66c84b53f405346311e84789.jpg
 extracting: train/american_bulldog_110_jpg.rf.eea553ad66c84b53f405346311e84789.txt
 extracting: train/american_bulldog_112_jpg.rf.b77a8201aec6eed208e2c988edee59fd.jpg
 extracting: train/american_bulldog_112_jpg.rf.b77a8201aec6eed208e2c988edee59fd.txt
 extracting: train/american_bulldog_113_jpg.rf.fbb760ba752231d4069a532342934743.jpg
 extracting: train/american_bulldog_113_jpg.rf.fbb760ba752231d4069a532342934743.txt
 extracting: train/american_bulldog_114_jpg.rf.af79f2d403da4b0b60c8d99581a5dd0c.jpg
 extracting: train/american_bulldog_114_jpg.rf.af79f2d403da4b0b60c8d99581a5dd0c.txt
 extracting: train/american_bulldog_119_jpg.rf.401188c694c1ded9f154a12e27135c7c.jpg
 extracting: train/american_bulldog_119_jpg.rf.401188c694c1ded9f154a12e27135c7c.txt
 extracting: train/american_bulldog_11_jpg.rf.303a2fa998d45fd9c815635808910266.jpg
```

Dataset contains 3 folders: train, valid, test

Each folder contains an image file, corresponding label file as txt file and a _darknet.labels file containing class names.

Let, us see what inside _darknet.labels:

```
%cat /content/train/_darknet.labels
```

Output:

```
cat
dog
```

Let, us now print information about dataset:

```
import os
train_dir = '/content/train'
test_dir = '/content/test'

train_images = [file for file in os.listdir(train_dir) if file.endswith(".jpg")]
train_labels = [ file[:-4]+ ".txt"  for file in train_images]

test_images = [file for file in os.listdir(test_dir) if file.endswith(".jpg")]
test_labels = [ file[:-4]+ ".txt"  for file in test_images]

print(f"Train data contains {len(train_images)} images and {len(train_labels)} labels")
print(f"Test data contains {len(test_images)} images and {len(test_labels)} labels")
```

This will print

```
Train data contains 2576 images and 2576 labels
Test data contains 368 images and 368 labels
```

So, in total, we have 2576 training images and 368 as test images.

Importing the libraries:

```python
import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torch.optim as optim
import torchvision.transforms.functional as FT
from torch.utils.data import DataLoader
from tqdm import tqdm
#from tqdm.auto import tqdm
import pandas as pd
import os
import PIL
import skimage
from skimage import io
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches
seed = 123
import cv2
torch.manual_seed(seed)
from collections import Counter
from time import sleep
import random
from google.colab.patches import cv2_imshow
```

LoadData class:

torch.utils.data.Dataset is an abstract class representing a dataset. Your custom dataset should inherit Dataset and override the following methods:

- __len__ so that len(dataset) returns the size of the dataset.

- __getitem__ to support indexing such that dataset[i] can be used to get ith sample

```python
class LoadData(torch.utils.data.Dataset):

    def __init__(self, file_dir, S=7, B=2, C=2, transform=None):
        self.file_dir=file_dir
        self.images_list=[image for image in os.listdir(self.file_dir) if image.endswith(".jpg")]
        self.labels_list=[ file[:-4]+ ".txt"  for file in self.images_list]
        self.S=S
        self.B=B
        self.C=C
        self.transform=transform

    def __len__(self):
        return len(self.images_list)
```

This will take the **images_list** from file_dir and **label list** from file_dir

Images_list looks like :

```
'boxer_100_jpg.rf.d650a1af1c386eaf4867fbbe0191f7a8.jpg',
'Sphynx_176_jpg.rf.2adb609b619a1485234acbc02e22f448.jpg',
'Bengal_10_jpg.rf.91f5d7a0f514ca99e1649890373d5b59.jpg',
'havanese_176_jpg.rf.abe2aa6a585df33335590d73efc10421.jpg',
'pug_155_jpg.rf.127666dbee8ee6a0c0723e973c10764f.jpg',
'newfoundland_138_jpg.rf.d3e1ff91897eda0cfb58bd5384b50658.jpg',
'boxer_147_jpg.rf.c1e78f402c4fe62a94b0d1e9b281f15c.jpg',
'leonberger_117_jpg.rf.a6f4d07b07a5ed8b5da7bd923a8dd1e8.jpg',
'american_bulldog_136_jpg.rf.f46b72ef8e913313a00edb00c553a288.jpg',
'Russian_Blue_172_jpg.rf.90269b8cb3cdc8aca29a02dd881ab80a.jpg',
'saint_bernard_124_jpg.rf.49012c0c37941b2c8f9df7187a587bc7.jpg',
'english_setter_100_jpg.rf.f638a7a67a6fb63d69c9204eeef80f69.jpg',
'British_Shorthair_204_jpg.rf.62162168d856ef38f094f778b9891e62.jpg',
'Maine_Coon_175_jpg.rf.07f99b21970f7124053828b71ce70e52.jpg',
'yorkshire_terrier_169_jpg.rf.442a3381315f971c50375a3ad91ca83b.jpg',
'scottish_terrier_117_jpg.rf.c7f7818248eaf4dece960e36d22c5928.jpg',
```

To get the length of images_list

Labels_list looks like

```
['great_pyrenees_169_jpg.rf.08856fb507d2ab55765730f6c2929e9a.txt',
 'Russian_Blue_129_jpg.rf.480ec40fc0c15ac67fb05cd940fae9e3.txt',
 'Russian_Blue_19_jpg.rf.ca796d2467a9acbb449ab3f478dc1091.txt',
 'Siamese_188_jpg.rf.80c12b40c628e06134d7d812c3b21822.txt',
 'Bombay_129_jpg.rf.25d30229f720ea1d43784ab2a1777300.txt',
 'great_pyrenees_127_jpg.rf.0b8f8dbc5800f60b1343edb1e692b289.txt',
 'samoyed_164_jpg.rf.95fc9d6227d2dc0a2b50b35d3a6589ef.txt',
 'german_shorthaired_104_jpg.rf.ce195332795822f24203190ade98d0bf.txt',
 'samoyed_184_jpg.rf.ad7a990b67037646ae21689024861b91.txt',
 'Sphynx_145_jpg.rf.eb7c03e87ca840cbde304ba9a00be258.txt',
 'newfoundland_162_jpg.rf.a3df3bb0c01a66d009dad26ff1d16a27.txt',
 'Persian_107_jpg.rf.2fc449ee83fd39c0ffdfd2dba47bef7b.txt',
 'Sphynx_179_jpg.rf.67ec2c69fee9f4b4b1b9d67e1098def4.txt',
 'Abyssinian_152_jpg.rf.c77147f874da04a0c52070358e0adc6f.txt',
 'boxer_152_jpg.rf.038f5689c280dc5dbfc3d2d394ce81d5.txt',
 'Bengal_110_jpg.rf.ddda14b775ad25f6a578ab182e48eed5.txt',
```

_getitem__ is to get the item from any index idx

```
def __getitem__(self, idx):
    label_path= os.path.join(self.file_dir,self.labels_list[idx])
    label_file = open(label_path, 'r')

    #read labels
    boxes=[]
    for line in label_file.readlines():
        box = list(map(float,line.split()))
        boxes.append(box)

    boxes = torch.tensor(boxes)
    #read image
    img_path = os.path.join(self.file_dir, self.images_list[idx])
    image = Image.open(img_path)
    image = image.convert("RGB")

    if self.transform:
        # image = self.transform(image)
        image, boxes = self.transform(image, boxes)

    #convert center (x,y) w.r.t cell and write x,y,w,h , objectness and class label in label_matrix
    label_matrix = torch.zeros((self.S, self.S, self.C + 5 * self.B))

    for box in boxes:
        class_label, x_img, y_img, w, h = box
        class_label=int(class_label)

        i, j =  int(self.S * y_img) , int(self.S * x_img)
        x_cell, y_cell = self.S * x_img - j , self.S * y_img - i

        if label_matrix[i, j, self.C] == 0:
            # Set that there exists an object
            label_matrix[i, j, self.C] = 1
            # write box coordinates
            box_coordinates = torch.tensor([x_cell, y_cell, w, h])
            label_matrix[i, j, self.C + 1 : self.C + 5 ] = box_coordinates
            #Set class probability as 1
            label_matrix[i, j, class_label] = 1

    return image, label_matrix
```

```
label_path
```

```
'/content/train/Russian_Blue_129_jpg.rf.480ec40fc0c15ac67fb05cd940fae9e3.txt'
```

```
boxes
```

```
[[0.0, 0.497, 0.3498452012383901, 0.286, 0.4520123839009288]]
```

```
img_path
```

```
'/content/train/Russian_Blue_129_jpg.rf.480ec40fc0c15ac67fb05cd940fae9e3.jpg'
```

This will return image and its label_matrix of size S,S,C+5*B. For S=7, B=2, this will 7,7,12

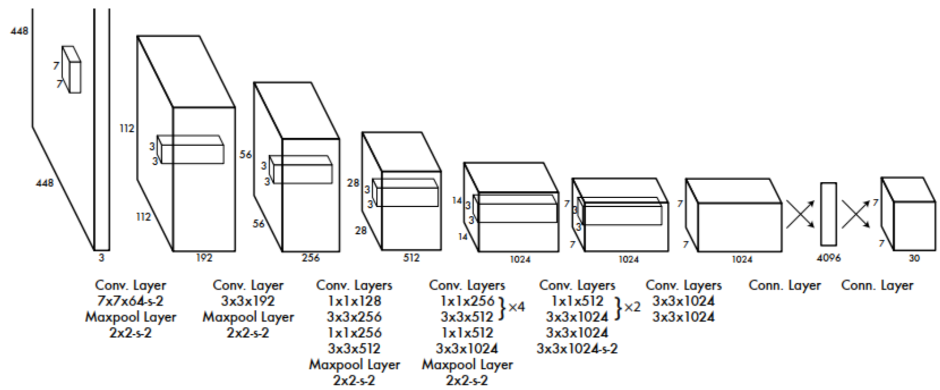
Network Architecture: We will use this in future while defining different parameters of layers

Read more about how to write custom dataloader in pytorch:

https://pytorch.org/tutorials/recipes/recipes/custom_dataset_transforms_loader.html

Defining Network Architecture:

```
architecture_config = [
    #Tuple: (kernel_size, number of filters, strides, padding)
    (7, 64, 2, 3),
    #"M" = Max Pool Layer
    "M",
    (3, 192, 1, 1),
    "M",
    (1, 128, 1, 0),
    (3, 256, 1, 1),
    (1, 256, 1, 0),
    (3, 512, 1, 1),
    "M",
    #List: [(tuple), (tuple), how many times to repeat]
    [(1, 256, 1, 0), (3, 512, 1, 1), 4],
    (1, 512, 1, 0),
    (3, 1024, 1, 1),
    "M",
    [(1, 512, 1, 0), (3, 1024, 1, 1), 2],
    (3, 1024, 1, 1),
    (3, 1024, 2, 1),
    (3, 1024, 1, 1),
    (3, 1024, 1, 1),
    #Doesnt include fc layers
]
```



CNNBlock: This class comprises of convolution layer, Batch normalization, and activation function applied.

forward: this function applies convolution, batch normalization with leaky relu activation to input x i.e., it acts as forward pass.

```
class CNNBlock(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(CNNBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
        self.batchnorm = nn.BatchNorm2d(out_channels)
        self.leakyrelu = nn.LeakyReLU(0.1)

    def forward(self, x):
        return self.leakyrelu(self.batchnorm(self.conv(x)))
```

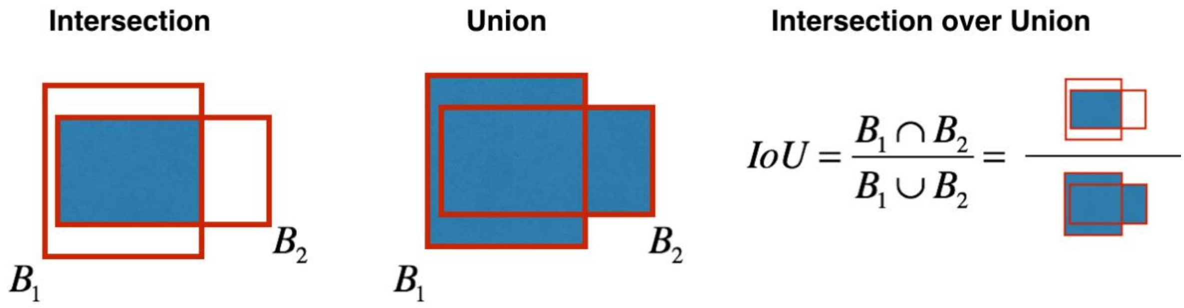
Network formation of YOLOv1:

```
class YoloV1(nn.Module):
    def __init__(self, in_channels=3, **kwargs):
        super(YoloV1, self).__init__()
        self.architecture = architecture_config
        self.in_channels = in_channels
        self.darknet = self._create_conv_layers(self.architecture)
        self.fcs = self._create_fcs(**kwargs)

    def forward(self, x):
        x = self.darknet(x)
        return self.fcs(torch.flatten(x, start_dim=1))

    #create darknet
    def _create_conv_layers(self, architecture):
        layers = []
        in_channels = self.in_channels

        for x in architecture:
            if type(x) == tuple:
                layers += [CNNBlock(in_channels, x[1], kernel_size=x[0], stride=x[2], padding=x[3])]
                in_channels = x[1]
            elif type(x) == str:
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            elif type(x) == list:
                conv1 = x[0] #Tuple
                conv2 = x[1] #Tuple
                repeats = x[2] #Int

                for _ in range(repeats):
                    layers += [CNNBlock(in_channels, conv1[1], kernel_size=conv1[0], stride=conv1[2], padding=conv1[3])]
                    layers += [CNNBlock(conv1[1], conv2[1], kernel_size=conv2[0], stride=conv2[2], padding=conv2[3])]
                    in_channels = conv2[1]

        return nn.Sequential(*layers)
```

This is followed by fully connected network

```
#create fully connected layer
def _create_fcs(self, split_size, num_boxes, num_classes):
    S, B, C = split_size, num_boxes, num_classes
    return nn.Sequential(nn.Flatten(),
                         nn.Linear(1024 * S * S, 600),
                         nn.Dropout(0.0),
                         nn.LeakyReLU(0.1),
                         nn.Linear(600, S * S * (C + B * 5)),
                         nn.Sigmoid())
```

Intersection over Union (IoU)



To calculate IoU, coordinates of two boxes are given. We find the intersecting coordinates and find intersecting area. Similarly union area is area of B_1 + B_2 – Intersecting area.

```
""" This function calculates intersection over union
    input parameters: Prediction boxes(boxes_preds) predicted for a batch of input i.e, of size (batch_size,S,S, 4)
                      Ground truth boxes(boxes_gt) for a batch of input i.e, of size (batch_size,S,S,4)
                      centroid_format ='image' or 'cell'
                      if the center of bounding box (x,y) are offset w.r.t cell then centroid_format should be kept equal to 'cell'.Hence, it will
                      be first converted w.r.t image via. function convert_boxes_wrt_image and then passed for IOU calculation.

    output: IOU for all input pairs of prediction and ground truth boxes i.e, of size (batch_size,S,S,1)
    (0,1,2,3) == (x,y,w,h)"""
def IOU(boxes_preds, boxes_gt,  centroid_format='image'):

    #change the box centroid coordinates (x,y) w.r.t image if they are w.r.t cell
    if centroid_format == 'cell':
        boxes_preds = convert_boxes_wrt_image(boxes_preds)
        boxes_gt = convert_boxes_wrt_image(boxes_gt)

    #box-1 left-top(x1,y1) and right-bottom(x2,y2) coordinate points
    b1_x1 = boxes_preds[...,0:1] - boxes_preds[...,2:3]/2    # center_x - w/2
    b1_x2 = boxes_preds[...,0:1] + boxes_preds[...,2:3]/2    # center_x + w/2
    b1_y1 = boxes_preds[...,1:2] - boxes_preds[...,3:4]/2    # center_y - h/2
    b1_y2 = boxes_preds[...,1:2] + boxes_preds[...,3:4]/2    # center_y + h/2

    #box-2 left-top(x1,y1) and right-bottom(x2,y2) coordinate points
    b2_x1 = boxes_gt[...,0:1] - boxes_gt[...,2:3]/2    # center_x - w/2
    b2_x2 = boxes_gt[...,0:1] + boxes_gt[...,2:3]/2    # center_x + w/2
    b2_y1 = boxes_gt[...,1:2] - boxes_gt[...,3:4]/2    # center_y - h/2
    b2_y2 = boxes_gt[...,1:2] + boxes_gt[...,3:4]/2    # center_y + h/2

    #left-top(x1,y1) and right-bottom(x2,y2) coordinate points of intersection box
    x1 = torch.max(b1_x1,b1_x1)
    x2 = torch.min(b1_x2,b2_x2)
    y1 = torch.max(b1_y1,b2_y1)
    y2 = torch.min(b1_y2,b2_y2)

    #.clamp(0) is for the case when they don't intersect. Since when they don't intersect, one of these will be negative so that should become 0
    intersection_area = (x2 - x1).clamp(0) * (y2 - y1).clamp(0)

    #Area of box-1 and box-2
    box1_area = abs((b1_x2 - b1_x1) * (b1_y2 - b1_y1))
    box2_area = abs((b2_x2 - b2_x1) * (b2_y2 - b2_y1))

    iou = intersection_area / (box1_area + box2_area - intersection_area + 1e-6)

    return iou
```

Function: convert_boxes_wrt_image

```
""" convert_boxes_wrt_image , this function takes input boxes whose centroid format are w.r.t cell and convert it w.r.t image
    input:
    batch_boxes (tensor): Predicted or Ground truth boxes for a batch and it is a tensor of size (batch_size,S,S,4)
    S (integer) : Split size for algorithm

    output (tensor): Boxes whose centroid are w.r.t 'image' of size (batch_size, S,S,4)
"""
def convert_boxes_wrt_image(batch_boxes, S=7):

    #cell_indices is 'j' with size (N,S,S,1) and cell_indices.permute(0, 2, 1, 3) is 'i' with size (N,S,S,1)
    batch_size = batch_boxes.shape[0]
    cell_indices = torch.arange(7).repeat(batch_size, 7, 1).unsqueeze(-1).to(DEVICE)      #(N,S,S,1)
    x = 1 / S * (batch_boxes[..., :1] + cell_indices)
    y = 1 / S * (batch_boxes[..., 1:2] + cell_indices.permute(0, 2, 1, 3))
    w_h = batch_boxes[..., 2:4]

    converted_bboxes = torch.cat((x, y, w_h), dim=-1)  #(N,S,S,4)

    return converted_bboxes
```

YOLO

Loss Function:

$$
\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]
+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2
+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2
+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
$$

This loss comprises of:

1. MSE from x,y
2. MSE from square root of height and width
3. Object loss
4. No object loss
5. Class loss

```
class YOLOLoss(nn.Module):
    def __init__(self, S=7, B=2, C=2):
        super(YOLOLoss,self).__init__()
        self.mse = nn.MSELoss(reduction = 'sum')
        self.S = S
        self.B = B
        self.C = C
        self.lambda_noobj = 0.5
        self.lambda_coord = 5
```

In paper, each image is divided into 7X7 grid. Each grid predicts two bounding boxes. IoU of both bounding boxes is calculated with ground truth. Out of two, that bounding box is consider for further calculations whose IoU with ground truth is maximum out of two.

```python
""" This function the loss
    Input: Predictions of size (batch_size, S,S, 5*B+C)
           Ground truth targets of size (batch_size, S * S* (5*B+C))
    Output: Mean Square Error defined in YOLOv1 paper (hence, a float value)"""

class YOLOLoss(nn.Module):
    def __init__(self, S=7, B=2, C=2):
        super(YOLOLoss,self).__init__()
        self.mse = nn.MSELoss(reduction = 'sum')
        self.S = S
        self.B = B
        self.C = C
        self.lambda_noobj = 0.5
        self.lambda_coord = 5

    def forward(self, predictions, ground_truths):
        #reshape prediction output from (batch_size , 588) to (batch_size, 7,7,12)
        predictions = predictions.reshape(-1, self.S, self.S, 5*self.B+self.C )


        #get the best box i.e, box responsible for prediction out of 2 predicted bounding box
        iou_b1 = IOU(predictions[...,self.C + 1 : self.C + 5], ground_truths[..., self.C + 1 : self.C + 5],centroid_format='cell')  #(N,S,S,1)
        iou_b2 = IOU(predictions[...,self.C + 6 : ], ground_truths[..., self.C + 1 : self.C + 5],centroid_format='cell')  #(N,S,S,1)

        ious = torch.cat([iou_b1.unsqueeze(0), iou_b2.unsqueeze(0)], dim=0)  #(2,N,S,S,1)

        iou_maxes, bestbox = torch.max(ious, dim=0)   #dimension of both iou_maxes and bestbox are (N,S,S,1)

        exists_box = ground_truths[..., self.C : self.C + 1 ] # (N,S,S,1)
```

Continue...

As mentioned in paper, box loss is calculated by MSE of x,y and MSE of square root of w, h

```python
#Calculate box loss

box_pred =exists_box * (
            ( bestbox * predictions[...,self.C + 6 :] + (1- bestbox) * predictions[...,self.C + 1 : self.C + 5])   #(N,S,S,4)
)

box_gt = exists_box * ground_truths[..., self.C + 1 : self.C + 5] #(N,S,S,4)

## take square root of width and height
box_pred[..., 2:4] = torch.sign(box_pred[..., 2:4]) * torch.sqrt(torch.abs(box_pred[..., 2:4] + 1e-6))

box_gt[..., 2:4] = torch.sqrt(box_gt[..., 2:4])

box_loss = self.mse(torch.flatten(box_pred, end_dim=-2),
                    torch.flatten(box_gt, end_dim=-2))      # float
```

Object loss and no object loss is directly calculated

```python
#Calculate object loss

pred_C = bestbox * predictions[..., self.C + 5 : self.C + 6] + (1-bestbox) * predictions[..., self.C : self.C + 1] #(N,S,S,1)
gt_C = ground_truths[..., self.C : self.C + 1] #(N,S,S,1)

object_loss = self.mse(torch.flatten(exists_box * pred_C, end_dim=-2),
                       torch.flatten(exists_box * gt_C, end_dim=-2))     # float

#Calculate no object loss

no_object_loss = self.mse(
    torch.flatten((1 - exists_box) * predictions[..., self.C:self.C + 1], end_dim=-2),     #(N*S*S,1)
    torch.flatten((1 - exists_box) * ground_truths[..., self.C:self.C + 1], end_dim=-2))

no_object_loss += self.mse(
    torch.flatten((1 - exists_box) * predictions[..., self.C + 5:self.C + 6], end_dim=-2),
    torch.flatten((1 - exists_box) * ground_truths[..., self.C:self.C + 1], end_dim=-2)) #no_object_loss is float
```

This is class loss between predicted and ground truth class. Final loss is addition of all the individual losses.

```python
#Class probability loss

class_loss = self.mse(torch.flatten(exists_box * predictions[..., :self.C], end_dim=-2),
                      torch.flatten(exists_box * ground_truths[..., :self.C], end_dim=-2))

#Total loss
loss = ( self.lambda_coord * box_loss  # first two rows in paper
      + object_loss  # third row in paper
      + self.lambda_noobj * no_object_loss  # forth row
      + class_loss)  # fifth row

return loss
```

Train function:

```
def train_fn(train_loader, model, optimizer, loss_fn , epoch):
    mean_loss = []
    with tqdm(train_loader, unit="batch") as loop:
        loop.set_description(f"Epoch {epoch}")
        for batch_idx, (x,y) in enumerate(loop):
            x , y = x.to(DEVICE) , y.to(DEVICE)
            out = model(x)
            loss = loss_fn(out,y)
            mean_loss.append(loss.item())
            optimizer.zero_grad()          #all gradient should be zero in next epoch and should not carry previous gradients
            loss.backward()                #calculates the gradient update for each layer
            optimizer.step()               #update the weights
            sleep(0.1)
        print(f"Mean loss was {sum(mean_loss) / len(mean_loss)}")
```

Compose class for image transformation:

```
class Compose(object):
    def __init__(self, transforms):
        self.transforms = transforms

    def __call__(self, img, bboxes):
        for t in self.transforms:
            img, bboxes = t(img), bboxes

        return img, bboxes


transform = Compose([transforms.Resize((448, 448)), transforms.ToTensor()])
```

Learning rate scheduler:

From the paper:

Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from 10^{-3} to 10^{-2}. If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with 10^{-2} for 75 epochs, then 10^{-3} for 30 epochs, and finally 10^{-4} for 30 epochs.

```
LEARNING_RATE = 2e-5
#DEVICE = "cuda" if torch.cuda.is_available else "cpu"
DEVICE = "cpu"
BATCH_SIZE = 16 # 64 in original paper but resource exhausted error otherwise.
WEIGHT_DECAY = 0
EPOCHS = 2
NUM_WORKERS = 2
PIN_MEMORY = True
LOAD_MODEL = False
LOAD_MODEL_FILE = "model.pth"
```

Putting everything together:

```
def main():
    model = YoloV1(split_size=7, num_boxes=2, num_classes=2).to(DEVICE)
    optimizer = optim.Adam(
        model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY
    )
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer, factor=0.1, patience=3, mode='max', verbose=True)
    loss_fn = YOLOLoss()

    if LOAD_MODEL:
        load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

    train_dataset = LoadData(file_dir= '/content/train',transform=transform)

    test_dataset = LoadData(file_dir='/content/test',transform=transform)

    train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        drop_last=False,
    )

    test_loader = DataLoader(
        dataset=test_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        drop_last=False,
    )
```

```
for epoch in range(EPOCHS):
    train_fn(train_loader, model, optimizer, loss_fn)

    #pred_boxes, target_boxes = get_bboxes(
   #     train_loader, model, iou_threshold=0.5, threshold=0.4
   # )

    #mean_avg_prec = mean_average_precision(
    #    pred_boxes, target_boxes, iou_threshold=0.5, box_format="midpoint"
    #)
    #print(f"Train mAP: {mean_avg_prec}")

    #scheduler.step(mean_avg_prec)

checkpoint = {
        "state_dict": model.state_dict(),
        "optimizer": optimizer.state_dict(),
}
save_checkpoint(checkpoint, filename=LOAD_MODEL_FILE)
```

```
if __name__ == "__main__":
    main()
```

Getting the best detections out of all detection bounding boxes predicted by the model via. Non-maximum suppression:

The function for Non-maximum suppression will be given input with specific format as list containing bounding box information as [class_int, prob ,x_img ,y_img, w_img, h_img] w.r.t to each image in a batch.

Hence, the following function tensor_to_boxes converts the prediction tensor of size (N,S,S,5*B+C) to list of dimension (N,S*S,6) after converting the box centroid w.r.t image width and height.

```
"""Input: label_matrix of size (batch_size, S,S, 5*B+C)
  Output type-list : Contains elements as list. Each list has values as: [class_int, prob ,x_img ,y_img, w_img, h_img]
"""

def tensor_to_boxes(in_tensor ,S=7 ,B=2 ,C=2):
    bboxes1 = in_tensor[..., C + 1:C + 5]
    bboxes2 = in_tensor[..., C + 6: ]

    #get best boxes
    scores = torch.cat((in_tensor[..., C:C + 1].unsqueeze(0), in_tensor[..., C + 5:C + 6].unsqueeze(0)), dim=0)
    max_scores, best_box = torch.max(scores, dim=0)  #both are of size (N,S,S,1)
    best_boxes = bboxes1 * (1 - best_box) + best_box * bboxes2  #(N,S,S,4)

    converted_bboxes = convert_boxes_wrt_image(best_boxes)  #(N,S,S,4)
    predicted_class = in_tensor[..., :C].argmax(-1).unsqueeze(-1)  #(N,S,S,1)

    converted_preds = torch.cat((predicted_class, max_scores, converted_bboxes), dim=-1)  #(N,S,S,6)

    converted_preds = converted_preds.reshape(-1, S * S, 6)
    converted_preds[..., 0] = converted_preds[..., 0].long()
    all_bboxes = []

    for ex_idx in range(converted_preds.shape[0]):
        bboxes = []

        for bbox_idx in range(S * S):   #S*S = total no. of cells in SxS grid
            bboxes.append([x.item() for x in converted_preds[ex_idx, bbox_idx, :]])
        all_bboxes.append(bboxes)

    return all_bboxes  # (N,49,6)
```

```
"""
    Does Non Max Suppression given bboxes
    Parameters:
        bboxes (list): list of lists containing all bboxes with each bboxes
        specified as [class_int, prob_score, x_img, y_img, x_img, y_img]
        iou_threshold (float): threshold where predicted bboxes is correct
        threshold (float): threshold to remove predicted bboxes (independent of IoU)
        box_format (str): "midpoint" or "corners" used to specify bboxes
    Returns:
        list: bboxes after performing NMS given a specific IoU threshold
    """

def non_max_suppression(bboxes, iou_threshold, threshold):

    assert type(bboxes) == list

    bboxes = [box for box in bboxes if box[1] > threshold]
    bboxes = sorted(bboxes, key=lambda x: x[1], reverse=True)
    bboxes_after_nms = []

    while bboxes:
        chosen_box = bboxes.pop(0)

        #following loop discards those box whose class is same as chosen_box and iou w.r.t chosen box is > iou_threshold
        bboxes = [
            box
            for box in bboxes
            if box[0] != chosen_box[0]                #if the class is not same
            or IOU(torch.tensor(chosen_box[2:]),torch.tensor(box[2:])) < iou_threshold
        ]

        bboxes_after_nms.append(chosen_box)

    return bboxes_after_nms
```

Following is the function that performs **NMS** . When we give input as bboxes which is list containing all bounding box list of format [class_int, prob ,x_img ,y_img, w_img, h_img], corresponds to single image. We get output as list which contain only best bounding boxes i.e., 1 bounding box corresponds to each predicted class, if the confidence is greater than threshold.

Model Evalaution via. mAP:

The function to calculate mAP takes input of list containing list of all bounding box Information in the format [train_idx, class_int, prob, x_img, y_img, w_img, h_img]. mAP is mean of Average Precision(AP) over all classes.

AP = area under precision-recall curve for a class.

Hence, to get the boxes in this format (to give input to mAP) at every epoch of training or while testing we use get_bboxes function.

get_bboxes functions takes train_loader/test_loader as input, if while training we are using it set train=True.

It sets the model to evaluation mode, predicts the output and gives the bounding boxes required for calculating mAP both for predicted as well as ground truth bounding boxes.

```
""" This function get_bboxes takes dataset (train/test), model, iou_threshold, threshold as input.
    It sets the model to evaluation mode and predicts the output for given dataset and convert the prediction tensor to
    list of box information as [train_idx, class_int, prob , x_img, y_img, w_img, h_img  ] and outputs it for further use in
    model evaluation for mAP.
"""
def get_bboxes(loader,model,iou_threshold, threshold,pred_format="cells",device="cuda", S=7, B=2, C=2 , train = True):
    all_pred_boxes = []
    all_true_boxes = []
    # make sure model is in eval before get bboxes
    model.eval()
    train_idx = 0

    for batch_idx, (x, labels) in enumerate(loader):
        x = x.to(device)
        labels = labels.to(device)

        with torch.no_grad():
            predictions = model(x)

        batch_size = x.shape[0]
        true_bboxes = tensor_to_boxes(labels)  #converts tensor to list of boxes
        predictions = predictions.reshape(-1, S, S, 5*B + C )  #(N,588) --> (N,S,S,5*B+C) because tensor_to_boxes expects this size of i/p
        bboxes = tensor_to_boxes(predictions)

        for idx in range(batch_size):
            nms_boxes = non_max_suppression(
                bboxes[idx],
                iou_threshold=iou_threshold,
                threshold=threshold
                )

            for nms_box in nms_boxes:
                all_pred_boxes.append([train_idx] + nms_box)

            for box in true_bboxes[idx]:
                # many will get converted to 0 pred
                if box[1] > threshold:
                    all_true_boxes.append([train_idx] + box)

            train_idx += 1

    if train == True:
        model.train()

    return all_pred_boxes, all_true_boxes
```

```python
def mean_average_precision(pred_boxes, true_boxes, iou_threshold=0.5, num_classes=2):

    average_precisions = []   #to store AP corresponding to all classes
    epsilon = 1e-6            # used for numerical stability

    for c in range(num_classes):
        #take all detection bbox and ground truth bbox belonging to class c
        detections=[det for det in pred_boxes if det[1] == c]
        ground_truths=[grt for grt in true_boxes if grt[1] == c]

        #Initialize a dictionary with (key,value) as (train_idx, tensor of zeros of size number of ground truth
        #bbox for image at train_idx). This is to keep track whether detection w.r.t this ground truth bbox is
        #covered. Because we can have only one TP detection bbox w.r.t one ground truth bbox.

        ##foll will give dict with key=train_idx and value = no. of times train_idx appeared in ground_truths
        #Example: amount_bboxes = {0:3, 1:5}
        amount_bboxes = Counter([gt[0] for gt in ground_truths])
        #Now change to tensor of zeros-Example:ammount_bboxes = {0:torch.tensor[0,0,0], 1:torch.tensor[0,0,0,0,0]}
        for key, val in amount_bboxes.items():
            amount_bboxes[key] = torch.zeros(val)

        # sort by box probabilities which is index 2
        detections.sort(key=lambda x: x[2], reverse=True)
        TP = torch.zeros((len(detections)))        #initialize a tensor of zeros of size (len(detections))
        FP = torch.zeros((len(detections)))
        total_true_bboxes = len(ground_truths)

        # If none exists for this class then we can safely skip
        if total_true_bboxes == 0:
            continue
```

Continue...

```python
        # If none exists for this class then we can safely skip
        if total_true_bboxes == 0:
            continue

        for detection_idx, detection in enumerate(detections):
            #get all ground truth bbox belonging to same image(train_idx) to with detection belongs to
            ground_truth_img = [bbox for bbox in ground_truths if bbox[0] == detection[0]]

            #from all these ground truth bbox of this image get the best bbox that detection is representing
            best_iou = 0
            for idx, gt in enumerate(ground_truth_img):
                iou = IOU(torch.tensor(detection[3:]),torch.tensor(gt[3:]))
                if iou > best_iou:
                    best_iou = iou
                    best_gt_idx = idx

            #check whether best_iou is > iou_threshold and
            #this ground truth bbox is not covered before if yes this detection is TP
            if best_iou > iou_threshold:
                if  amount_bboxes[detection[0]][best_gt_idx]==0:
                    TP[detection_idx] = 1
                    amount_bboxes[detection[0]][best_gt_idx] = 1
                else:
                    FP[detection_idx] = 1
            else:
                FP[detection_idx] = 1

        TP_cumsum = torch.cumsum(TP, dim=0)
        FP_cumsum = torch.cumsum(FP, dim=0)
        recalls = TP_cumsum / (total_true_bboxes + epsilon)
        precisions = torch.divide(TP_cumsum, (TP_cumsum + FP_cumsum + epsilon))
        precisions = torch.cat((torch.tensor([1]), precisions))
        recalls = torch.cat((torch.tensor([0]), recalls))
        # torch.trapz for numerical integration
        average_precisions.append(torch.trapz(precisions, recalls))

    return sum(average_precisions) / len(average_precisions)
```

```python
def main():
    model = YoloV1(split_size=7, num_boxes=2, num_classes=2).to(DEVICE)
    optimizer = optim.Adam(
        model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY
    )
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer, factor=0.1, patience=3, mode='max', verbose=True)
    loss_fn = YOLOLoss()

    if LOAD_MODEL:
        load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

    train_dataset = LoadData(file_dir= '/content/train',transform=transform)

    test_dataset = LoadData(file_dir='/content/test',transform=transform)

    train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        drop_last=False,
    )

    test_loader = DataLoader(
        dataset=test_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        drop_last=False,
    )

    for epoch in range(EPOCHS):

        train_fn(train_loader, model, optimizer, loss_fn, epoch)

        pred_boxes, target_boxes = get_bboxes(
            train_loader, model, iou_threshold=0.5, threshold=0.4, device=DEVICE )

        mean_avg_prec = mean_average_precision(
            pred_boxes, target_boxes, iou_threshold=0.5)
        print(f"Train mAP: {mean_avg_prec}")

        scheduler.step(mean_avg_prec)

    checkpoint = {
            "state_dict": model.state_dict(),
            "optimizer": optimizer.state_dict(),
    }
    save_checkpoint(checkpoint, filename=LOAD_MODEL_FILE)
```

Hence, the complete loop of training will be now modified as:

```python
if __name__ == "__main__":
    main()
```

Train the model....

```
Epoch 0: 100%|██████████| 161/161 [00:57<00:00,  2.79batch/s]
Mean loss was 44.052795576012656
Epoch 1:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.02153470367193222
Epoch 1: 100%|██████████| 161/161 [00:57<00:00,  2.78batch/s]
Mean loss was 32.11251536067228
Epoch 2:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.07187370955944061
Epoch 2: 100%|██████████| 161/161 [00:57<00:00,  2.79batch/s]
Mean loss was 27.565581007773833
Epoch 3:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.25845736265182495
Epoch 3: 100%|██████████| 161/161 [00:57<00:00,  2.79batch/s]
Mean loss was 23.268739048738656
Epoch 4:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.3366401195526123
Epoch 4: 100%|██████████| 161/161 [00:57<00:00,  2.79batch/s]
Mean loss was 18.86909055413667
Epoch 5:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.45115986466407776
Epoch 5: 100%|██████████| 161/161 [00:57<00:00,  2.80batch/s]
Mean loss was 15.66784277613859
Epoch 6:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.5672346353530884
Epoch 6: 100%|██████████| 161/161 [00:57<00:00,  2.79batch/s]
Mean loss was 13.040108834734614
Epoch 7:   0%|          | 0/161 [00:00<?, ?batch/s]Train mAP: 0.7270433306694031
Epoch 7: 100%|██████████| 161/161 [00:57<00:00,  2.79batch/s]
Mean loss was 10.389480792217373
```

You may want to save the model for further use...

Model evaluation on Test dataset:

```
model = YoloV1(split_size=7, num_boxes=2, num_classes=2).to(DEVICE)
#LOAD_MODEL_FILE = '/content/model.pth'
LOAD_MODEL_FILE = '/content/drive/MyDrive/SpanIdea/YOLO_from_scratch/weights/model_22.7_mAP.pth'

optimizer = optim.Adam(
        model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY
    )
load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

model.eval()
#Get mAP on test data:

test_dataset = LoadData(file_dir='/content/test',transform=transform)

test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=False, drop_last=False)

pred_boxes, target_boxes = get_bboxes(test_loader, model, iou_threshold=0.5, threshold=0.4, device=DEVICE ,
                                      train = False)

mean_avg_prec = mean_average_precision(pred_boxes, target_boxes, iou_threshold=0.5)
print(f"Test mAP: {mean_avg_prec}")
```

Visualize the results by drawing bounding box:

We can draw bounding box for our test image as follows:

```
#Get detection for input image
S=7
B=2
C=2
img_path = '/content/test/Abyssinian_131_jpg.rf.e8acfb60e4d01529586b9d81930b35a2.jpg'
image = Image.open(img_path)
image = image.convert("RGB")
image = image.resize((448,448))
tran = transforms.ToTensor()
x = tran(image)
x = x.unsqueeze(0)
predictions = model(x.to(DEVICE))
predictions = predictions.reshape(-1, S, S, 5*B + C )
bboxes = tensor_to_boxes(predictions)

batch_size = len(bboxes)
for idx in range(batch_size):
    nms_boxes = non_max_suppression(
                bboxes[idx],
                iou_threshold=0.5,
                threshold=0.2
                )

img = cv2.imread(img_path)
width , height, ch = img.shape
for box in nms_boxes:
    x1 = int(width* (box[2] - box[4]/2))
    x2 = int(width* (box[2] + box[4]/2))
    y1 = int(height* (box[3] - box[5]/2))
    y2 = int(height* (box[3] + box[5]/2))
    label = class_names[int(box[0])]
    confidence = float(box[1])
    shape = [x1,y1,x2,y2]
    #draw bounding box and write confidence and text
    cv2.rectangle(img, (x1, y1), (x2, y2), colors[label], 2)
    cv2.putText(img, "{} [{:.2f}]".format(label, float(confidence)),
                (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                colors[label], 2)
cv2_imshow(img)
```

In the ‘output.jpg’ we can see the predicted bounding boxes.

Following function **class_colors** assign random unique colors to all classes**:**

```
def class_colors(names):
    """
    Create a dict with one random BGR color for each
    class name
    """
    return {name: (
        random.randint(0, 255),
        random.randint(0, 255),
        random.randint(0, 255)) for name in names}


class_names=['cat','dog']
colors=class_colors(class_names)
```

Results:

This YOLOv1 model was trained for following parameters :

```
LEARNING_RATE = 2e-5
DEVICE = "cuda" if torch.cuda.is_available else "cpu"
BATCH_SIZE = 16  #16 # 64 in original paper but resource exhausted error otherwise.
WEIGHT_DECAY = 0
EPOCHS = 20
NUM_WORKERS = 2
PIN_MEMORY = True
LOAD_MODEL = False
LOAD_MODEL_FILE = "model.pth"
```

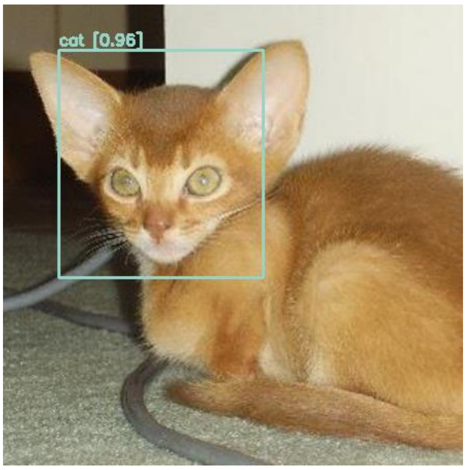
The results are as follows:

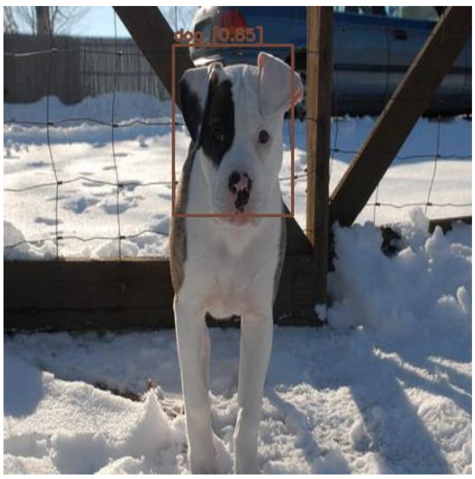
Training mAP: 0.8269129991531372

Test mAP: 0.22777032852172852

Some examples of predictions are as follows:

Correct predictions:





Incorrect predictions: