Implementation of YOLOv2 from Scratch Using Pytorch.

1. Overview of Network

YOLO V2 (You Only Look Once version 2) is a real-time object detection system that builds upon the original YOLO model by incorporating various architectural and training modifications. The core idea is to predict bounding boxes and class probabilities directly from full images in a single evaluation.

2. Network Architecture

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Backbone Network:

Darknet-19, a convolutional neural network designed for feature extraction. The network contains batchnorm layers and other layers as mentioned in table above.

Fully Convolutional Layers:

- The last average pool is removed and three FCN layers with 1024 channels using 3x3 convolutions were added.Also the network contains a skip connection from the last Maxpool. This will have 26x26 shape so it is sliced in four parts and then concatenated at the end of the backbone and thereafter FCN layers are added.

```
part2 = x1[:, :, :height    2, width    2 :]
part3 = x1[:, :, height    2 :, :width    2]
part4 = x1[:, :, height    2 :, width    2 :]
residual = torch.cat((part1, part2, part3, part4), dim=1)
```

- At the end, one 1x1 convolutional layer to reduce the number of output channels to the required number ofclasses. The network will now have a shape of `B*(N*(C+5))x13x13` . This is permuted and then view is changed to have `Bx13x13xNx(C+5)` . Where B: Batch size, N: No of anchor boxes, C: No of classes.

Output Layer:

Number of output channels = number of anchor boxes * (4 bounding box coordinates + number of classes +object confidence score)

Output shape = BxSxSxNx(5+C)

Shapes

The input shape is 416x416 and at the endo of FCN layers it will get reduced to 13x13 feature map along width and height dimension and there will be channel dimension accordingly based on the no of classes in the dataset.

3. Implementation

The YOLO V2 network was implemented using PyTorch. The key components include the backbone network (Darknet-19) and, data preprocessing pipelines, custom loss function implementation, implementation of mean average precision and finally the output layer and post processing of prediction using non max suppression.

- **Anchor boxes:** The anchor boxes need to be determined by running k-means clustering on the original dataset to determine the appropriate anchor boxes.

- **Backbone Network Implementation (Darknet-19)**: The backbone network is designed to extract features from input images using a series of convolutional layers.

- **Custom Loss function implementation**: The custom loss function is implemented and consists of four major parts. Loss for object, loss for no object, Loss for bounding box coordinates, class loss. Three loss functions nn.CrossEntropy for class loss, nn.BCEWithLogitsLoss, and nn.MSELoss were used.

4. Dataset and Training

African Wildlife Dataset: A dataset from Kaggle containing images four African wildlife species was used. Link: African Wildlife Dataset

Data Preprocessing:

Load and preprocess the dataset using custom dataloaders. This included formatting targets into appropriatetensor shape and assigning a bounding box to a particular anchor box. The target tensor shape was SxSxNx(C+5) - The targets were build like this. For the width and height of bounding box, bw/pw and bh/ph was calculated in the target. For center (x, y) , sig(tx) and sig(ty) was calculate and placed on the target. Thereafter IOU calculation was performed to match the appropriate anchor box to the bounding box. The matched anchor box will now contain above calculated parameters with objectness score 1. Other anchor boxes will have zero values if they were not matched to any bounding box.

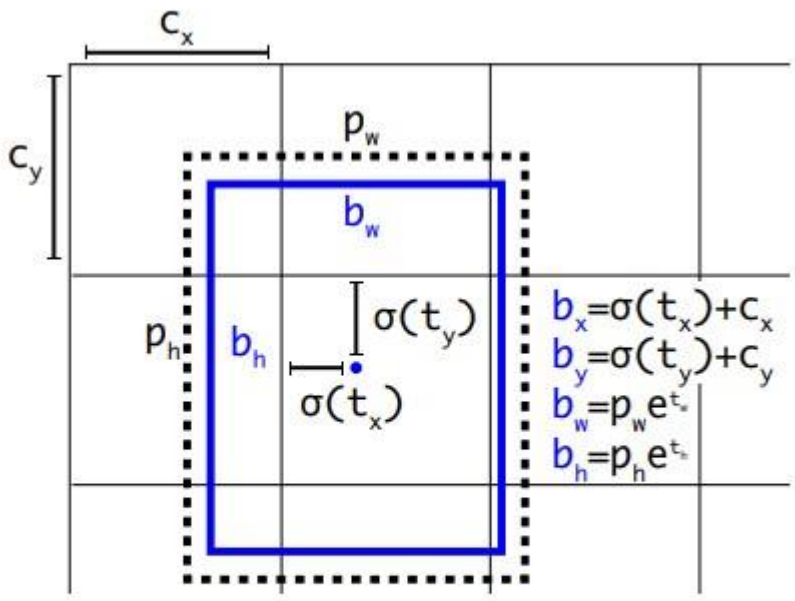


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

Training Loop:

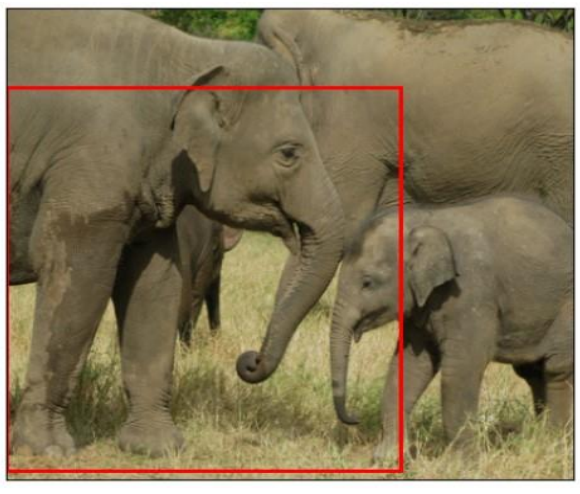
The training loop involves feeding the images through the model, computing the loss using the YOLO loss function, and updating the model weights using an optimizer. Also mAP was implemented in the training loop to evaluate the performance of the model.

The predictions of the model are than processed and the float values are converted into pixel values (for bounding box) and the logits(for objectness score and class prob) were normalized between 0-1 using sigmoid and softmax.

Non-max suppression was used to suppress the highly overlapping bounding boxes. Bounding box predictions and class probabilities are visualized on test images, showcasing the detected objects with bounding boxes.

5. Results

```
Epoch 100/100
----------



train Loss: 1.8434



val Loss: 1.9731
Mean Average Precision: 0.20104852318763733

Model with Best mAP: tensor(0.3027)
Training complete in 127m 42s
```



6. Summary of the Work

Implemented the YOLO V2 object detection model from scratch using PyTorch.

Utilized the Darknet-19 architecture as the backbone for feature extraction.

Processed and trained on the African Wildlife Dataset from Kaggle.

Implemented custom data loaders, data transformations, and the YOLO loss function.

Trained the model and tested it with post-processing steps like non-max suppression.

7. References

1. YOLO V2 Paper
2. YOLO V2 Video Tutorial Series
3. Custom Dataloaders and Transforms in PyTorch
4. Bounding Box Visualization in PyTorch
5. African Wildlife Dataset

Additional Notes:

- For proper training addition experiments are needed on loss function lambda values and the data preprocessing transformations.
- Three layers with 1024 channels using 3x3 convolutions.
- One 1x1 convolutional layer with the number of output channels required.
- For the African Wildlife Dataset (4 classes), the number of output channels is 9 times the number of anchorboxes.
- **Bounding Box Visualization:** Utilize PyTorch's visualization utilities for displaying bounding boxes.
- The nn.CrossEntropy takes .long() vales if the targets are not one hot encoded.(eg: 1,2,3, targets like this.) Other wise float values if the targets are one hot encoded.