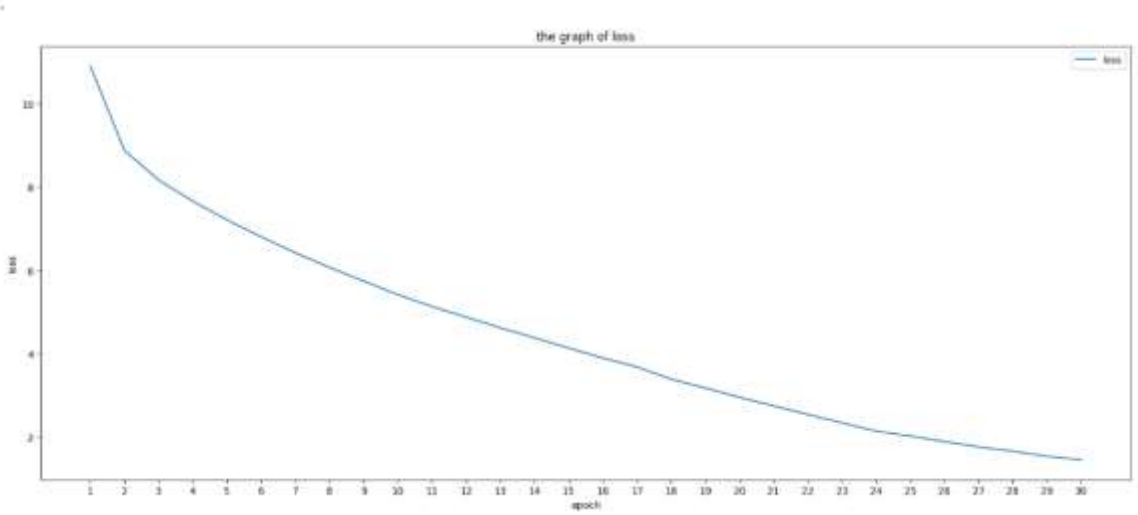
ACV HW5 REPORT

Object detection with YOLO V3

YOLO V3 is an object detection algorithm known for its speed and accuracy. It utilizes a single-stage detection approach, features a Darknet-53 backbone, employs anchor boxes, and predicts bounding boxes at multiple scales. YOLO V3 achieves a balance between speed and accuracy, making it suitable for real-time applications[1].

Task 1: Running the provided notebook on Kaggle

After setting up our Kaggle account and notebook, we did our best to understand the provided starter code and then ran it for 30 epochs while saving loss in the variable. After finishing training we then plotted the loss diagram showing how loss varied as epochs passed by. The loss diagram is shown below



The graph above shows that loss decreased with epochs i.e as the number of epochs elapsed increased the loss of the model reduced this is because as the epochs pass by the model learns more and hence its prediction capability increases. This is a good model since the loss diagram kept the downward or decreasing trend.

Task 2**: Explaining some sections from the notebook**

a. def iou(box1, box2, is pred = True)

a.1 explanation

This function, denoted as iou (Intersection over Union), serves to compute the Intersection over Union score between two bounding boxes, a crucial metric utilized in object detection tasks. It operates on two input bounding boxes, box1, and box2, represented as tensors with shape [..., 4], where the final dimension holds the bounding box coordinates and dimensions ([x, y, width, height]). Additionally, it takes a boolean flag is_pred, indicating whether the provided boxes represent predictions or ground truth labels. In its implementation, when

is_pred is True, the function calculates the IoU score based on the bounding box coordinates and dimensions. It determines the coordinates of the top-left and bottom-right corners of both boxes, computes the intersection and union areas, and then evaluates the IoU score using a formula that prevents division by zero. Conversely, when is_pred is False, the function computes the IoU score using only the width and height of the bounding boxes. The function returns the computed IoU score as a tensor with the same shape as the input tensors. This iou function holds significance in object detection applications, particularly within models like YOLO, where it plays a crucial role during both training and evaluation phases by quantitatively measuring the accuracy of predicted bounding boxes compared to ground truth annotations. Higher IoU scores indicate better alignment between predicted and ground truth bounding boxes, thereby aiding in model training and evaluation processes.

The role played by this function in object detection using YOLO.

This iou function is crucial in object detection tasks, particularly in models like YOLO (You Only Look Once). In YOLO, during both training and evaluation, the model predicts bounding boxes for objects in an image. After obtaining these predictions, the IoU score is used to measure the accuracy of predicted bounding boxes compared to ground truth annotations. Higher IoU scores indicate better alignment between predicted and ground truth boxes, facilitating model training and evaluation by providing a quantitative measure of detection accuracy. This function enables YOLO to assess the quality of its predictions and refine its parameters accordingly to improve detection performance.

b. def nms(bboxes, iou threshold, threshold)

b.1 explanation

The nms function implements Non-Maximum Suppression (NMS), a crucial technique in object detection aimed at removing redundant bounding boxes. Initially, it filters out bounding boxes with confidence scores below a specified threshold to eliminate unreliable predictions. Then, the remaining boxes are sorted based on their confidence scores in descending order, prioritizing higher confidence predictions. Subsequently, an empty list, bboxes_nms, is initialized to store the final bounding boxes after suppression. The NMS algorithm iterates through the sorted list of boxes, selecting the first box and comparing it with the remaining boxes to check for overlap using the Intersection over Union (IoU) metric. If the IoU is below a predefined threshold or if the confidence score of the current box is lower than the first box, the current box is added to the bboxes_nms list, avoiding duplicate additions. Finally, the function returns the refined list of bounding boxes after non-maximum suppression. In the context of object detection, particularly with YOLO, this function is instrumental in refining bounding box predictions and removing redundant boxes, ultimately enhancing the accuracy and efficiency of the detection system.

b.2 the role of nms function in object detection using yolo

In YOLO-based object detection, the nms function is vital for post-processing predicted bounding boxes. After YOLO predicts bounding boxes for various objects in an image, nms

filters out low-confidence detections and then performs non-maximum suppression to remove redundant boxes. This ensures that only the most confident and non-overlapping bounding boxes are retained, improving the accuracy and efficiency of the object detection system.

c. class YOLOv3(nn.Module)

c.1 explanation

The YOLOv3 class represents the YOLOv3 (You Only Look Once version 3) object detection model, implemented as a PyTorch module. This class is initialized with parameters for input channels and the number of classes for detection, which default to 3 channels for RGB images and 21 classes, respectively. The class inherits from nn.Module, the base class for neural network modules in PyTorch, and defines a series of layers comprising the YOLOv3 architecture using convolutional neural network (CNN) blocks, residual blocks, upsampling layers, and scale prediction layers. These layers are stored in a nn.ModuleList named self.layers. The forward method performs the forward pass through the model, iteratively passing input data through each layer. During this process, it collects outputs from scale prediction layers and maintains route connections, which are used for skip connections and concatenation during upsampling. Finally, the method returns a list of outputs containing predictions for objects at different scales. Overall, the YOLOv3 class encapsulates the architecture and functionality of the YOLOv3 object detection model, providing methods for performing object detection tasks on input images.

How this class YOLOv3 is used in object detection by using YOLO

The YOLOv3 class serves as the backbone architecture for object detection using the YOLO (You Only Look Once) methodology. Its primary role lies in orchestrating the complex series of operations involved in processing input images and generating predictions for objects within them. This class encapsulates the entire YOLOv3 architecture, comprising various types of layers such as convolutional neural network (CNN) blocks, residual blocks, upsampling layers, and scale prediction layers. Through its forward method, the class conducts a forward pass on input images, sequentially passing them through each layer in the architecture. During this process, the model extracts relevant features from the input image and makes predictions about the presence, location, and class of objects. Additionally, the class facilitates post-processing steps such as non-maximum suppression and confidence thresholding to refine the predictions and filter out redundant detections. Overall, the YOLOv3 class is instrumental in performing efficient and accurate object detection tasks, making it a crucial component in a wide range of applications including surveillance, autonomous driving, and object tracking.

d. class Scale Prediction(nn.Module)

d.1 explanation

The ScalePrediction class is designed to handle the prediction of bounding boxes and class probabilities at multiple scales in an object detection system, particularly in the context of YOLO (You Only Look Once) architecture. It inherits from the nn.Module class and consists of layers defined within its constructor (__init__). In detail, the class comprises a pred submodule, which is a sequential module consisting of convolutional layers followed by batch normalization and Leaky ReLU

activation. This submodule is responsible for processing input features to produce predictions. The number of output channels for the first convolutional layer is chosen to increase the network's capacity. The final convolutional layer outputs predictions for bounding boxes and class probabilities, with the number of output channels determined by (num_classes + 5) * 3, where num_classes represents the number of object classes, and each prediction consists of coordinates (x, y, width, height) for bounding boxes along with objectness score and class probabilities. In the forward pass method, the input tensor x is passed through the pred module to obtain raw predictions, which are then reshaped to match the desired output format for YOLO, organizing predictions into a grid structure across multiple scales with dimensions (batch_size, 3, grid_size, grid_size, num_classes + 5). This class essentially facilitates the process of predicting bounding boxes and class probabilities at different scales within the YOLO object detection framework, enabling efficient detection of objects with varying sizes and aspect ratios by leveraging convolutional layers and reshaping operations.

The role played by this class in object detection by YOLO

The ScalePrediction class in YOLO plays a crucial role in predicting bounding boxes and class probabilities across multiple scales. By leveraging convolutional layers and reshaping operations, it efficiently processes input features to generate predictions organized in a grid structure. This enables YOLO to detect objects of various sizes and aspect ratios, facilitating robust object detection in real-world scenarios**.**

e. ANCHORS

Anchors in object detection, particularly in YOLO (You Only Look Once) models, are predefined bounding boxes of different shapes and sizes that are placed at strategic positions across the input image. These anchor boxes serve as reference points for the model to predict bounding box coordinates and dimensions relative to these anchors. By providing anchor boxes at multiple scales and aspect ratios, YOLO can effectively detect objects of various sizes and shapes within an image. The provided anchor boxes, organized into lists corresponding to different feature maps, are scaled between 0 and 1 and are essential for generating accurate predictions during both training and inference phases. Each set of anchor boxes is associated with a specific feature map, allowing the model to focus on objects at different spatial resolutions. Adjusting anchor box configurations can influence the model's ability to detect objects of interest accurately.

The role played by anchors in object detection by using yolo

In YOLO (You Only Look Once) object detection, anchors play a fundamental role in defining the set of predefined bounding boxes used for detecting objects of interest. Anchors are predetermined shapes and sizes assigned to different regions of the input image. These anchor boxes are typically placed at various locations across the image and cover a range of aspect ratios and scales corresponding to different object sizes and shapes. During training, YOLO uses these anchor boxes to predict bounding boxes for objects. The model predicts offsets and confidences for each anchor box, which are then adjusted based on the anchor box's position and size. By using anchors, YOLO can efficiently detect objects of different scales and aspect ratios in a single forward pass, making the detection process faster and more accurate. Additionally, anchors help in handling overlapping objects and mitigating the computational complexity associated with exhaustive search methods. Overall, anchors serve as reference templates that guide the model's predictions and enable it to localize and classify objects effectively in real-world images**.**

f. what happens in the training loop?

The training loop manages the process of training a neural network model for object detection. It initializes a progress bar to visualize the training progress and creates an empty list to store the calculated losses during training. Iterating over the batches of training data, each batch is moved to the GPU, and model predictions are obtained by passing the input batch through the model. Subsequently, losses are computed for each scale using the defined loss function, model outputs, ground truth labels, and scaled anchors. These losses are appended to the list. Gradients are then reset, and backpropagation is performed to compute gradients of the loss with respect to model parameters. These gradients are used to update the model parameters via an optimization step. The scaler is updated to prepare for the next iteration, and the progress bar is updated with the mean loss calculated from accumulated losses. Finally, the mean loss is returned after the loop completes, signifying the end of one training epoch. This loop encapsulates the core training process, iteratively optimizing the model parameters to minimize the loss and improve the model's performance.

Task3: Adding an extra class

we added one extra class called “face” to the pre-existing 20 classes in the dataset. We annotated the provided workshop data in the homework material, we annotated face in the dataset by using CVAT software in YOLO format[2]. The annotations and their corresponding images were then uploaded to Kaggle. Then from them a csv file was made which link an image with its label.

YOLO data format

The YOLO (You Only Look Once) format is a specific data format used for annotating objects in images or videos for object detection tasks, particularly when training and deploying YOLO-based[3] object detection models.

Structure of yolo format

1. Bounding box information: each annotation include information about the bounding box of a detected object. This is generally made up of 4 values:

   - The x-coordinate of the center of the bounding box.
   - The y-coordinate of the center of the bounding box.
   - The width of the bounding box.
   - The height of the bounding box.
2. Class label: the yolo format include a value specifying class label and it is usually the starting value.

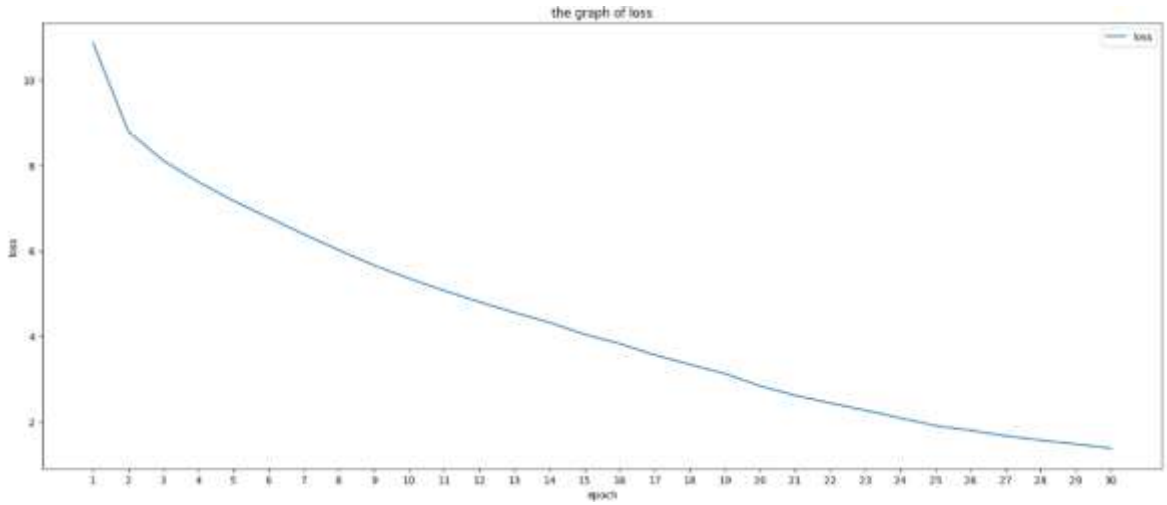
3. Image dimensions: the YOLO format often incorporates the dimensions of the image in which the objects were detected. These dimensions help in scaling the bounding box coordinates accurately.
4. Normalization: bounding box coordinates are typically normalized between 0 and 1. Normalization ensures consistency across images of different sizes which is a important during training and inference.
5. File structure a yolo format is often a text file

Merits of using the yolo format

1. **Standardization**: YOLO format ensures uniform representation of object detection annotations.
2. **Ease of Use**: Structured format simplifies parsing and processing of annotation data.
3. **Compatibility**: Works seamlessly with popular deep learning frameworks like TensorFlow and PyTorch.
4. **Efficiency**: Optimizes data storage and retrieval, especially for large datasets.
5. **Scalability**: Supports annotations for multiple objects within a single image.

Face detection

The resulting labels from annotations and their corresponding images were uploaded on Kaggle and they were merged with pre existing images and labels to make a unified dataset. This dataset was used to train the model with reference to the provided starter code and the resulting loss diagram was found as below.



This loss diagram kind of looks like the loss diagram that we had before adding more data and class but it you look closely you can find that apart from the fact that loss is continually decreasing as the epochs pass by, you can observe that compared with the graph that we had before here for each epoch the resulting loss is less than the loss that was obtained before. Therefore the final loss for this diagram is less than the final loss that was obtained before adding the face class. This means adding face label has made the prediction of the model a bit more accurate.

Face detection in action

Here the purpose of this section was to use web camera to implement face detection. We found that Kaggle notebook can not be able to access the our local webcam thus we used local editor(vs code to access the camera). We downloaded the checkpoint from Kaggle notebook and we used most of the codes as the code that we were using in kaggle notebook and we used loaded the checkpoint so it did not require us to train from scratch. The camera was able to detect face as it can be visualized in following video

Improving performance

To improve performance it is advisable to remove data for person because person bounding box are interfering with face sometime it is returning person when the face is expected to be the outcome. In additional the data we have for face are few, we tried to increase them by using additional dataset. At first I used the 7 Gb annotated images dataset that I obtained from Kaggle but I did not be able to implement this since I failed to merge the data formats(the existing one and one from kaggle) I tried to annotated my images from my local computer the same error happened but it could be handled if I had enough time to deal with this project. Thus as I sum up, the expected method to improve the performance of this model is to remove person class and increase the images for face class.

REFERENCES

[1] ‘YOLO Algorithm for Object Detection Explained [+Examples]’. Accessed: Apr. 02, 2024. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection, https://www.v7labs.com/blog/yolo-object-detection
[2] ‘CVAT’. Accessed: Apr. 02, 2024. [Online]. Available: https://www.cvat.ai/post/yolo
[3] ‘CVAT’. Accessed: Apr. 02, 2024. [Online]. Available: https://www.cvat.ai/

{Citation}