iiit srishti task 3

task

In this phase, you will learn to interpret results from various graphs and other output. You should refer to learning resources on the internet to interpret the results. Here is your task. (a) https://docs.ultralytics.com/datasets/detect/african-wildlife/#dataset-yaml This is a four-class classification dataset, containing animals usually seen in african continent. You will have to execute the code, and run it for a limited number - say anywhere between 10 to 25 epochs. At the end of the training/validation/testing phases, you will have to search your local folder for results that have been generated. (b) The major task for the week is to interpret each of the graphs generated and explain it. For example, if a confusion matrix is generated, you will have to interpret the diagram and explain what the figures/color codes etc actually mean. Best wishes, keep enjoying...

to do

interpret results from various graphs and other outputs.

firstly, execute the code for a limited number of epochs- going through the entire dataset once

secondly, interpret nd explain diagrams and graphs

code execution

https://docs.ultralytics.com/datasets/detect/african-wildlife/#dataset-structure

A YAML (Yet Another Markup Language) file defines the dataset configuration, including paths, classes, and other pertinent details. For the African wildlife dataset, the `african-wildlife.yaml` file is located at https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/datasets/african-wildlife.yaml.

To train a YOLO11n model on the African wildlife dataset for 100 epochs with an image size of 640, use the provided code samples. For a comprehensive list of available parameters, refer to the model's Training page.

train:

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolo11n.pt")  # load a pretrained model (recommended for training)

# Train the model
results = model.train(data="african-wildlife.yaml", epochs=100, imgsz=640)
```

```
# Start training from a pretrained *.pt model
yolo detect train data=african-wildlife.yaml model=yolo11n.pt epochs=100 imgsz=6
```

infer:

```
from ultralytics import YOLO

# Load a model
model = YOLO("path/to/best.pt")  # load a brain-tumor fine-tuned model

# Inference using the model
results = model.predict("https://ultralytics.com/assets/african-wildlife-sample.jp
g")
```

```
# Start prediction with a finetuned *.pt model
yolo detect predict model='path/to/best.pt' imgsz=640 source="https://ultralytic
s.com/assets/african-wildlife-sample.jpg"
```

```
C:\Users\madha\iiitsrishti\task3>C:\Users\madha\iiitsrishti\task3\yolo8venv\Scri
pts\activate
```

```
(yolo8venv) C:\Users\madha\iiitsrishti\task3>curl -L -o african-wildlife.zip http
s://github.com/ultralytics/assets/releases/download/v0.0.0/african-wildlife.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
100  100M  100  100M    0     0  3856k      0  0:00:26  0:00:26 --:--:-- 4167k
```

`curl` is a command-line tool used to transfer data from or to a server using supported protocols like http, https, ftp, etc. The `-L` flag tells curl to follow redirects if the URL provided has been moved to a new location. The `-o` flag specifies the name of the output file, in this case `african-wildlife.zip`, which means the downloaded file will be saved with this name.

in the output:

- `% total` represents the total size of the file to be downloaded, in percentage.
- `% received` shows how much of the file has been downloaded so far, as a percentage.
- `% xferd` (transferred) usually matches `% received`, showing the percentage of data transferred.
- `average speed dload` shows the average download speed in bytes or kilobytes per second.
- `time total` is the estimated total time for the download to complete.
- `time spent` indicates the time spent on the download so far.
- `time left` shows the remaining time until the download is expected to finish.
- `current speed` is the real-time download speed at the current moment.

command to train n cli:

```
(yolo8venv) C:\Users\madha\iiitsrishti\task3>yolo task=detect mode=train model
=yolov8n.pt data=african-wildlife.yaml epochs=20 imgsz=640
```

output:

```
Ultralytics 8.3.155  Python-3.12.2 torch-2.6.0+cpu CPU (Intel Core(TM) i5-10210
U 1.60GHz)
engine\trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=
randaugment, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=No
ne, close_mosaic=10, cls=0.5, conf=None, copy_paste=0.0, copy_paste_mode=fl
ip, cos_lr=False, cutmix=0.0, data=african-wildlife.yaml, degrees=0.0, determini
stic=True, device=cpu, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed
=None, epochs=20, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=t
orchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v
=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr
0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yo
lov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train3, nbs=6
4, nms=False, opset=None, optimize=False, optimizer=auto, overlap_mask=Tru
e, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profil
```

```
e=False, project=None, rect=False, resume=False, retina_masks=False, save=Tr
ue, save_conf=False, save_crop=False, save_dir=runs\detect\train3, save_frame
s=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, s
hear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True,
simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, ta
sk=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=Tr
ue, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, war
mup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None
Overriding model.yaml nc=80 with nc=4

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, Tru
e]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3,
2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, T
rue]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3,
2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2,
True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256,
3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1,
True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256,
5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2,
'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2,
'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3,
2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3,
2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256,
1]
 22        [15, 18, 21]  1    752092  ultralytics.nn.modules.head.Detect           [4, [64,
128, 256]]
Model summary: 129 layers, 3,011,628 parameters, 3,011,612 gradients, 8.2 GFLO
Ps

Transferred 319/355 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
train: Fast image access  (ping: 0.10.1 ms, read: 5.81.5 MB/s, size: 54.6 KB)
train: Scanning C:\Users\madha\iiitsrishti\task3\african-wildlife\train\labels... 105
2 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1052/1052 [00:02<0
0:00, 442.2
train: New cache created: C:\Users\madha\iiitsrishti\task3\african-wildlife\train\la
bels.cache
val: Fast image access  (ping: 0.10.1 ms, read: 5.24.0 MB/s, size: 42.2 KB)
val: Scanning C:\Users\madha\iiitsrishti\task3\african-wildlife\valid\labels... 225 i
mages, 0 backgrounds, 0 corrupt: 100%|██████████| 225/225 [00:00<00:0
0, 470.96it/s
val: New cache created: C:\Users\madha\iiitsrishti\task3\african-wildlife\valid\lab
els.cache
Plotting labels to runs\detect\train3\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' an
d determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.00125, momentum=0.9) with parameter groups 57 weigh
t(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\train3
Starting training for 20 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/20         0G     0.8387      2.239      1.215         41        640: 100%|████████
██| 66/66 [10:28<00:00,  9.53s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:36<00:00,  4.53s/it]
                   all        225        379      0.689      0.557      0.673      0.511
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/20         0G     0.9144      1.503      1.241         50        640: 100%|██████████
██| 66/66 [10:48<00:00,  9.83s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:55<00:00,  6.95s/it]
                   all        225        379      0.681       0.59      0.653      0.414

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/20         0G     0.9653      1.448      1.265         34        640: 100%|█████████
███| 66/66 [11:31<00:00, 10.48s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:31<00:00,  3.88s/it]
                   all        225        379      0.562      0.604      0.631      0.435

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/20         0G     0.9524      1.329       1.25         46        640: 100%|██████████
██| 66/66 [08:50<00:00,  8.04s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:31<00:00,  3.90s/it]
                   all        225        379      0.663      0.694      0.729      0.513

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/20         0G     0.9348      1.272      1.251         47        640: 100%|██████████
██| 66/66 [08:32<00:00,  7.77s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:31<00:00,  3.89s/it]
                   all        225        379      0.828      0.695      0.814       0.57

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       6/20         0G     0.9135      1.154      1.223         55        640: 100%|██████████
██| 66/66 [08:31<00:00,  7.74s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.71s/it]
                   all        225        379      0.837      0.778      0.865      0.631

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       7/20         0G     0.8803      1.099       1.21         49        640: 100%|██████████
██| 66/66 [08:32<00:00,  7.77s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.72s/it]
                   all        225        379      0.816      0.733      0.832      0.601
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       8/20         0G      0.894      1.053      1.222         45        640: 100%|████████
██| 66/66 [08:27<00:00,  7.69s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.69s/it]
                   all        225        379      0.866      0.831      0.899      0.681

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       9/20         0G     0.8089     0.9635      1.178         68        640: 100%|███████
███| 66/66 [08:36<00:00,  7.82s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.72s/it]
                   all        225        379      0.826      0.784      0.873       0.66

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/20         0G     0.8182     0.9147      1.173         47        640: 100%|███████
███| 66/66 [08:27<00:00,  7.70s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:30<00:00,  3.75s/it]
                   all        225        379      0.851      0.821      0.883      0.671
Closing dataloader mosaic

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      11/20         0G     0.7577     0.9254      1.146         18        640: 100%|███████
███| 66/66 [08:20<00:00,  7.58s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.72s/it]
                   all        225        379      0.901      0.872      0.912        0.7

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      12/20         0G     0.7047     0.7922      1.116         17        640: 100%|████████
██| 66/66 [08:24<00:00,  7.65s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:30<00:00,  3.75s/it]
                   all        225        379      0.949      0.819      0.928      0.726

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      13/20         0G     0.6915      0.735      1.108         22        640: 100%|████████
██| 66/66 [08:20<00:00,  7.59s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
```

```
0%|██████████| 8/8 [00:29<00:00,  3.75s/it]
           all       225       379     0.921     0.854     0.927     0.732

   Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   14/20        0G     0.6593     0.6925      1.086         20        640: 100%|██████
████| 66/66 [08:19<00:00,  7.57s/it]
          Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.74s/it]
           all       225       379     0.897      0.85      0.93      0.73

   Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   15/20        0G     0.6377     0.6469      1.058         22        640: 100%|██████
████| 66/66 [08:19<00:00,  7.57s/it]
          Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.71s/it]
           all       225       379     0.889     0.856     0.929     0.726

   Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   16/20        0G     0.6062     0.6191      1.043         20        640: 100%|███████
███| 66/66 [08:27<00:00,  7.69s/it]
          Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:30<00:00,  3.77s/it]
           all       225       379     0.889     0.892     0.941     0.746

   Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   17/20        0G     0.5957     0.5675      1.029         22        640: 100%|███████
███| 66/66 [08:23<00:00,  7.63s/it]
          Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.71s/it]
           all       225       379     0.937     0.866     0.939     0.758

   Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   18/20        0G     0.5711     0.5503      1.013         22        640: 100%|███████
███| 66/66 [08:22<00:00,  7.61s/it]
          Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:29<00:00,  3.68s/it]
           all       225       379      0.94     0.888     0.954     0.771

   Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   19/20        0G     0.5467     0.5178     0.9956         20        640: 100%|██████
████| 66/66 [08:21<00:00,  7.61s/it]
```

```
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:30<00:00,  3.78s/it]
                   all        225        379      0.935      0.889      0.948      0.779

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      20/20         0G      0.526     0.4887      0.979         20        640: 100%|██████
████| 66/66 [08:22<00:00,  7.62s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:30<00:00,  3.76s/it]
                   all        225        379      0.911      0.904      0.948      0.783

20 epochs completed in 3.121 hours.
Optimizer stripped from runs\detect\train3\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train3\weights\best.pt, 6.2MB

Validating runs\detect\train3\weights\best.pt...
Ultralytics 8.3.155  Python-3.12.2 torch-2.6.0+cpu CPU (Intel Core(TM) i5-10210
U 1.60GHz)
Model summary (fused): 72 layers, 3,006,428 parameters, 0 gradients, 8.1 GFLO
Ps
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 10
0%|██████████| 8/8 [00:25<00:00,  3.18s/it]
                   all        225        379      0.909      0.904      0.948      0.783
               buffalo         62         89      0.869      0.876      0.931      0.773
              elephant         53         91      0.865       0.89      0.934      0.758
                 rhino         55         85      0.975      0.953      0.965       0.82
                 zebra         59        114      0.927      0.897       0.96      0.782
Speed: 3.2ms preprocess, 98.0ms inference, 0.0ms loss, 0.9ms postprocess per
image
Results saved to runs\detect\train3
 Learn more at https://docs.ultralytics.com/modes/train

(yolo8venv) C:\Users\madha\iiitsrishti\task3>
```

this log shows the details of training a yolov8 object detection model using ultralytics library on a dataset called african wildlife. the system runs on python 3.12.2 and torch 2.6.0 using cpu on an intel core i5 processor.

the trainer settings list many options controlling how the training happens. for example, `batch=16` means the model processes 16 images at once before updating weights. `epochs=20` means the training runs for 20 full passes over the dataset. `device=cpu` means the training is done using the computer's central processor, not a gpu. options like `augment=False` mean no extra data augmentation was applied during training, except `auto_augment=randaugment`, which is a kind of automatic data augmentation. the `data=african-wildlife.yaml` specifies the dataset configuration file, and `model=yolov8n.pt` indicates the starting model weights being used.

the model summary shows the neural network architecture. it has 129 layers and about 3 million parameters. parameters are values the model learns during training. layers are different parts of the neural network that process the image data step by step. the model uses various modules like convolutional layers (conv), concatenation layers (concat), upsampling, and special blocks called c2f and sppf to extract features from images. these layers help the model understand patterns in images to detect objects.

the training process starts by loading images and labels from specified folders. the dataset has 1052 training images and 225 validation images. it creates cache files for faster access next time.

the optimizer used is adamw, a popular algorithm for adjusting model parameters to reduce loss. it automatically decides the best learning rate (lr0) and momentum to speed up convergence. learning rate controls how big each update step is, while momentum helps smooth the updates.

each epoch shows the training progress. losses like `box_loss`, `cls_loss`, and `dfl_loss` represent different errors the model tries to minimize. `box_loss` measures how well the model predicts the location of bounding boxes around objects, `cls_loss` measures how well it predicts the correct class labels, and `dfl_loss` relates to the distribution of bounding box coordinates.

after each epoch, the model is evaluated on the validation set. metrics shown include precision (p), recall (r), mAP50, and mAP50-95. precision tells how many of the predicted boxes were correct, recall tells how many actual objects were found, and mAP (mean average precision) summarizes detection accuracy at different thresholds. the numbers improving over epochs mean the model is learning to detect objects better.

the log also shows some training parameters like image size (640×640), number of worker threads (0 here), and saving the results in a folder named `runs\detect\train3`.

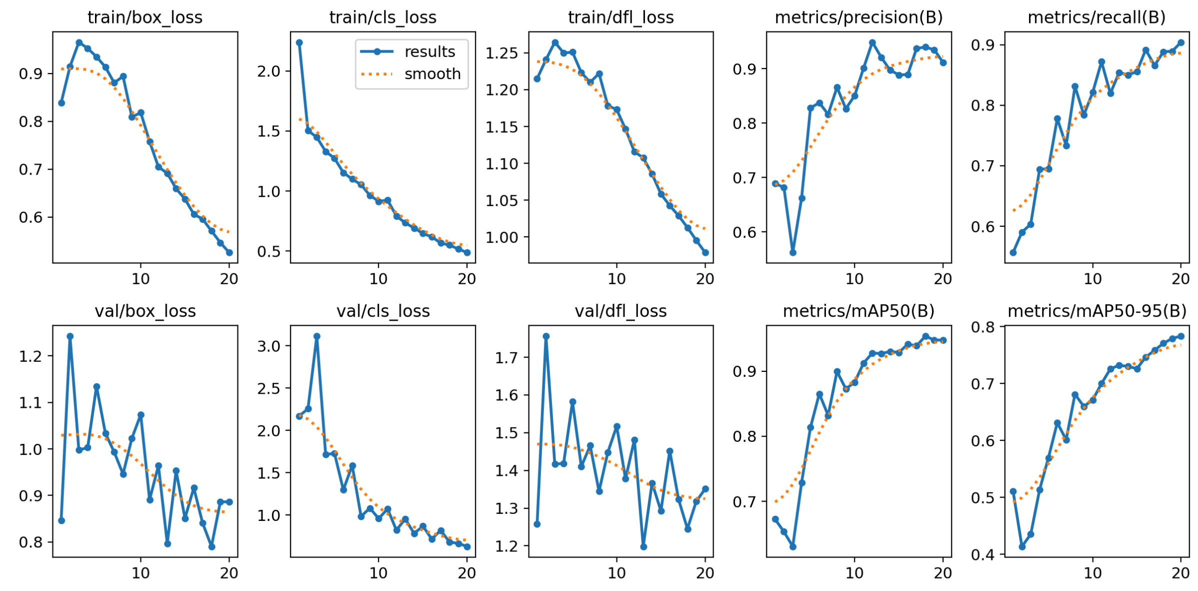
overall, this is a detailed output showing the setup, the model structure, and the step-by-step training progress on the african wildlife detection task. it shows how the model learns to detect animals by minimizing errors and improving performance metrics through multiple training cycles.

b. analysis

1. results.csv

```
epoch,time,train/box_loss,train/cls_loss,train/dfl_loss,metrics/precision(B),metrics/recall(B),metrics/mAP50(B),metrics/mAP50-95(B),val/box_loss,val/cls_loss,val/dfl_loss,lr/pg0,lr/pg1,lr/pg2
1,665.275,0.83866,2.23897,1.21502,0.68923,0.55662,0.67261,0.51105,0.84591,2.16964,1.25816,0.000410354,0.000410354,0.000410354
2,1370.73,0.91435,1.50318,1.24079,0.68146,0.59026,0.65324,0.41392,1.24261,2.25556,1.75619,0.000786083,0.000786083,0.000786083
3,2095.05,0.96533,1.44777,1.26458,0.56208,0.60355,0.63065,0.4353,0.99761,3.1155,1.41635,0.00112056,0.00112056,0.00112056
4,2657.75,0.95235,1.32894,1.2499,0.66263,0.694,0.72892,0.51346,1.00297,1.71369,1.41883,0.00106437,0.00106437,0.00106437
5,3202.22,0.93476,1.27245,1.25098,0.82771,0.69521,0.81362,0.56966,1.13488,1.72844,1.58215,0.0010025,0.0010025,0.0010025
6,3743.48,0.91346,1.15358,1.22322,0.83743,0.77793,0.86522,0.63115,1.034,1.29551,1.41008,0.000940625,0.000940625,0.000940625
7,4286.48,0.88035,1.09921,1.20998,0.81598,0.73336,0.83195,0.6012,0.99337,1.58628,1.46587,0.00087875,0.00087875,0.00087875
8,4824.24,0.89402,1.05348,1.22206,0.86584,0.83117,0.89943,0.68109,0.94593,0.98271,1.34427,0.000816875,0.000816875,0.000816875
9,5370.75,0.80889,0.96352,1.17832,0.82643,0.78386,0.87273,0.65975,1.02295,1.08075,1.44809,0.000755,0.000755,0.000755
10,5909.04,0.81825,0.91465,1.17316,0.85055,0.82098,0.88293,0.67109,1.07392,0.95602,1.51729,0.000693125,0.000693125,0.000693125
11,6439.26,0.75771,0.92541,1.1464,0.90075,0.87175,0.91223,0.69987,0.89065,1.0721,1.37767,0.00063125,0.00063125,0.00063125
12,6974.69,0.70469,0.79216,1.11578,0.94876,0.8194,0.92784,0.72566,0.96447,0.81974,1.48167,0.000569375,0.000569375,0.000569375
13,7505.68,0.69153,0.73505,1.10756,0.92097,0.85411,0.92696,0.73213,0.79686,0.95243,1.19826,0.0005075,0.0005075,0.0005075
14,8035.75,0.65934,0.69246,1.0856,0.89732,0.84981,0.93016,0.72995,0.95306,0.78092,1.36679,0.000445625,0.000445625,0.000445625
15,8565.36,0.63767,0.64694,1.05841,0.88851,0.85551,0.92866,0.72607,0.85114,0.86689,1.29254,0.00038375,0.00038375,0.00038375
16,9103.32,0.60619,0.61907,1.0427,0.88913,0.89163,0.94138,0.74627,0.91626,0.71415,1.45199,0.000321875,0.000321875,0.000321875
17,9637.17,0.59575,0.56748,1.02894,0.93737,0.86556,0.9389,0.75829,0.84072,0.8179,1.32279,0.00026,0.00026,0.00026
18,10169.2,0.5711,0.55031,1.01294,0.93959,0.88826,0.95382,0.77058,0.79046,0.68036,1.24514,0.000198125,0.000198125,0.000198125
```

```
19,10701.8,0.54669,0.51777,0.99556,0.9348,0.88919,0.94761,0.77914,0.88616,0.6
6277,1.31809,0.00013625,0.00013625,0.00013625
20,11234.9,0.526,0.48865,0.97899,0.91122,0.90413,0.94788,0.78324,0.88661,0.
62688,1.35081,7.4375e-05,7.4375e-05,7.4375e-05
```



i tracked the training progress over 20 epochs and saw that my losses steadily decreased while precision, recall, and mAP improved, showing my model was learning better. the learning rate gradually dropped, helping the model stabilize and refine its performance. overall, the metrics suggest my model got more accurate and confident with each epoch.

2. confusion matrix

it's called a **confusion matrix** because it shows where the model gets confused between different classes. basically, it highlights the mistakes the model makes—like when it confuses one class for another.

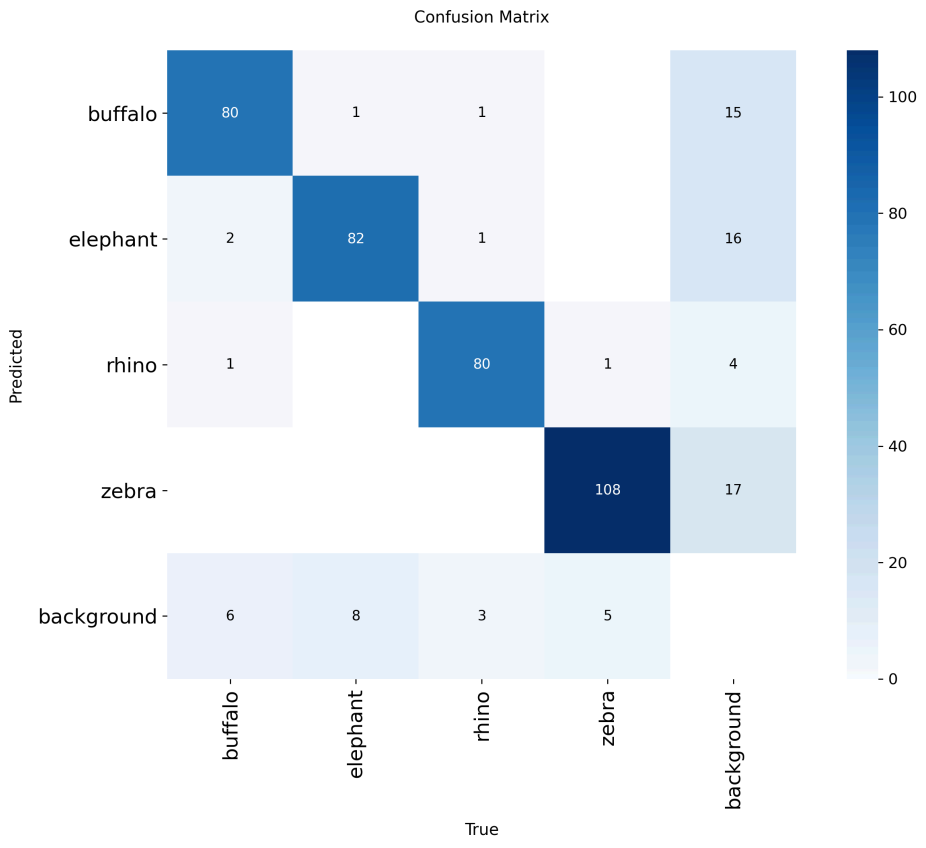
the matrix lays out these confusions clearly by comparing the actual class vs. the predicted class, making it easier to see exactly how often and where the model mixes things up. that's why the term "confusion" fits — it maps the errors or "confusions" in classification.

it is a table that shows the number of correct and incorrect predictions made by the model compared to the actual results. the matrix is organized with rows representing the actual classes and columns representing the predicted classes.

it helps to understand how well the model is doing in terms of:

- true positives (tp): cases where the model correctly predicted the positive class.

- true negatives (tn): cases where the model correctly predicted the negative class.
- false positives (fp): cases where the model incorrectly predicted the positive class when it was actually negative.
- false negatives (fn): cases where the model incorrectly predicted the negative class when it was actually positive.



i observre that the highest correct prediction count is for the zebra class (108), showing the model is very confident and accurate identifying zebras.

buffalo, elephant, and rhino have very similar correct predictions (80, 82, 80 respectively), which means the model performs consistently across these classes.

the model confuses background most often as elephant (8 times) and buffalo (6 times), which suggests these two animals might have visual features that overlap with background or the background class has patterns similar to these animals.

elephant is mistakenly predicted as backgroud 16 times, which is quite high. this could mean the model struggles to detect elephants clearly in some cases, possibly due to similar textures or occlusion.

buffalo is confused as background 15 times, almost as oftrn as elephants, hinting that the model may be mixing these large animals with background noise in challenging images.

rhino is rarely confuded with other animal classes (only 1 for buffalo and 1 for elephant), which suggests the model has learned distinct features for rhino well.

there are very few off-diagonal confusions between anumal classes, meaning the model mostly confuses animals with background rather than with other animals.

the backgropnd class predictions are very low (5 times correctly predicted), indicating that the model may not be very good at identifying background alone or that the background class is underrepresented or ambiguous.

the presence of some off-diagonal elements even in predicted vs true background (6 buffalo, 8 elephant, 3 rhino) shows that some background regions might contain partial or confusing features of animals, leading to misclassification.

overall, the matrix reveals the model excels at animal recognition but struggles to distinguish animals from background in many cases, especially forelephants and buffalo. this suggests improvements could be made by focusing on better background separation or using additional features for those tricky cases.

3. normalised confusion matrix

a normalized confusion matrix shows the proportion of correct and incorrect predictions for each class relative to the total true instances of that class.

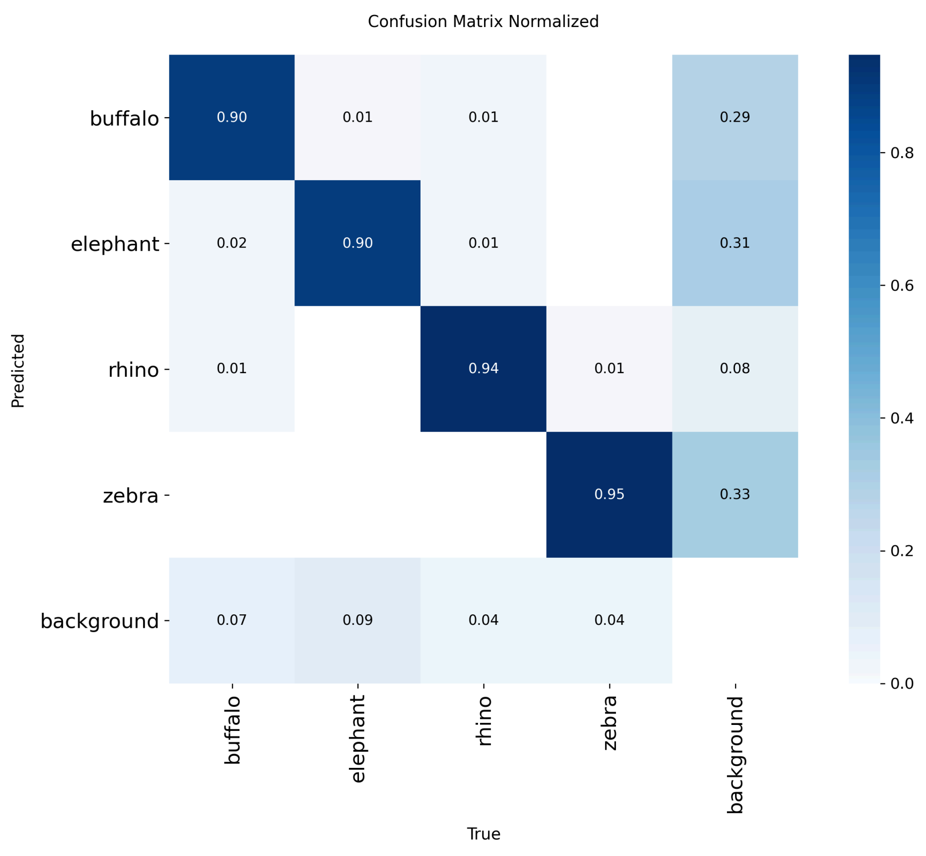
it helps me see not just raw counts but how well the model performs as a percentage, making it easier to compare across classes of different sizes.

each value in the matrix represents the fraction of times a class was predicted as another class, so high values on the diagonal mean good accuracy.

off-diagonal values show where the model confuses one class for another, highlighting weaknesses or overlap between classes.

normalizing the matrix helps me identify patterns like if a class is often mistaken for background or another specific class, regardless of how many examples there are.

overall, it gives a clearer picture of the model’s behavior and errors in a balanced way, helping me focus on improving specific class distinctions.



for buffalo, the recall is 0.90, so 90% of buffaloes were correctly identified. some buffaloes were predicted as elephants (1%), rhinos (1%), and background (7%). for elephants, recall is also 0.90, with some misclassified as buffalo (2%) and background (9%). rhinos have the highest recall at 0.94, with minor misclassifications as elephants (1%), zebras (1%), and background (4%). zebras have the best recall overall at 0.95, with small errors predicting them as rhinos (1%) and background (4%).

the background class has a low recall of 0.29, meaning only 29% of background samples were correctly identified. many background samples were misclassified as elephants (31%), zebras (33%), and rhinos (8%). this suggests the background class is not well separated from animal classes or has features similar to them.

overall, the model performs very well identifying animal classes, especially rhino and zebra, but struggles with the background class. the model also sometimes confuses animals as background, indicating challenges in detecting animals clearly against complex backgrounds. improving background separation and refining features for tricky classes could help.

it is important to consider precision as well, which measures how many predicted positives are correct, by looking at columns. also, the interpretation of background matters — if background means no animals, then many false positives happen, showing the model confuses background with animals. the normalization is likely row-wise, so values show proportions of true class samples predicted as each category.

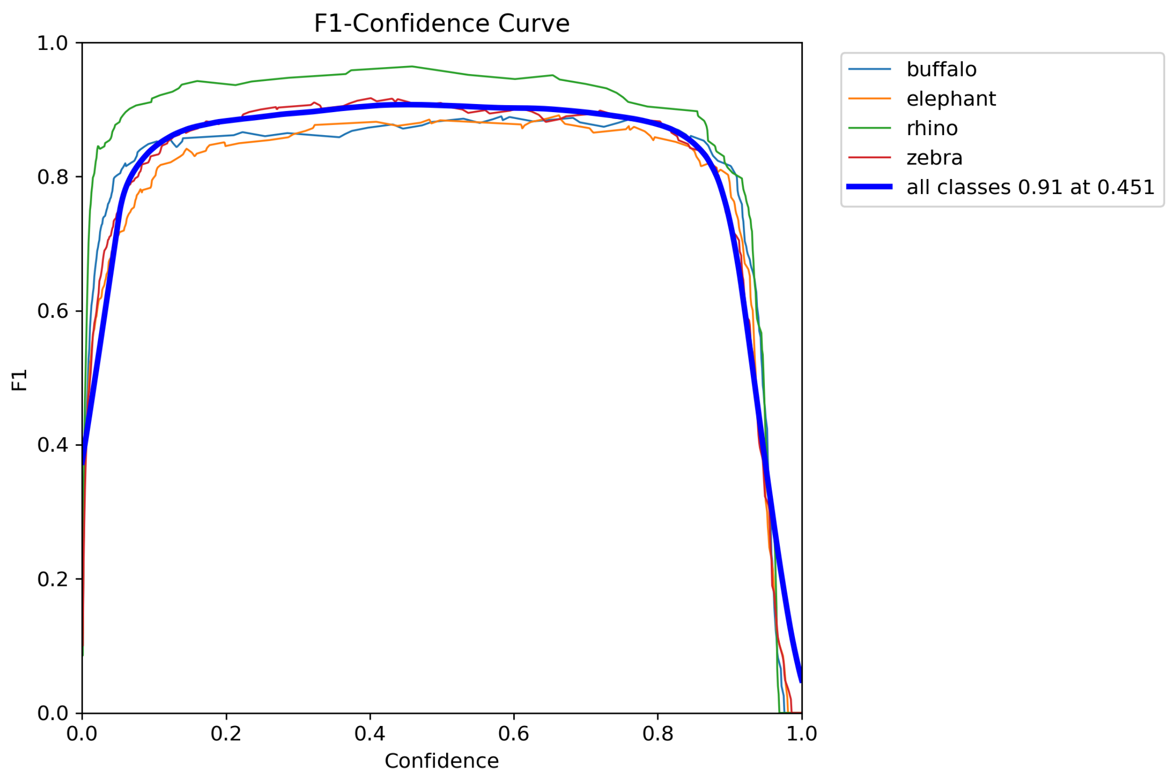
4. f1 curve

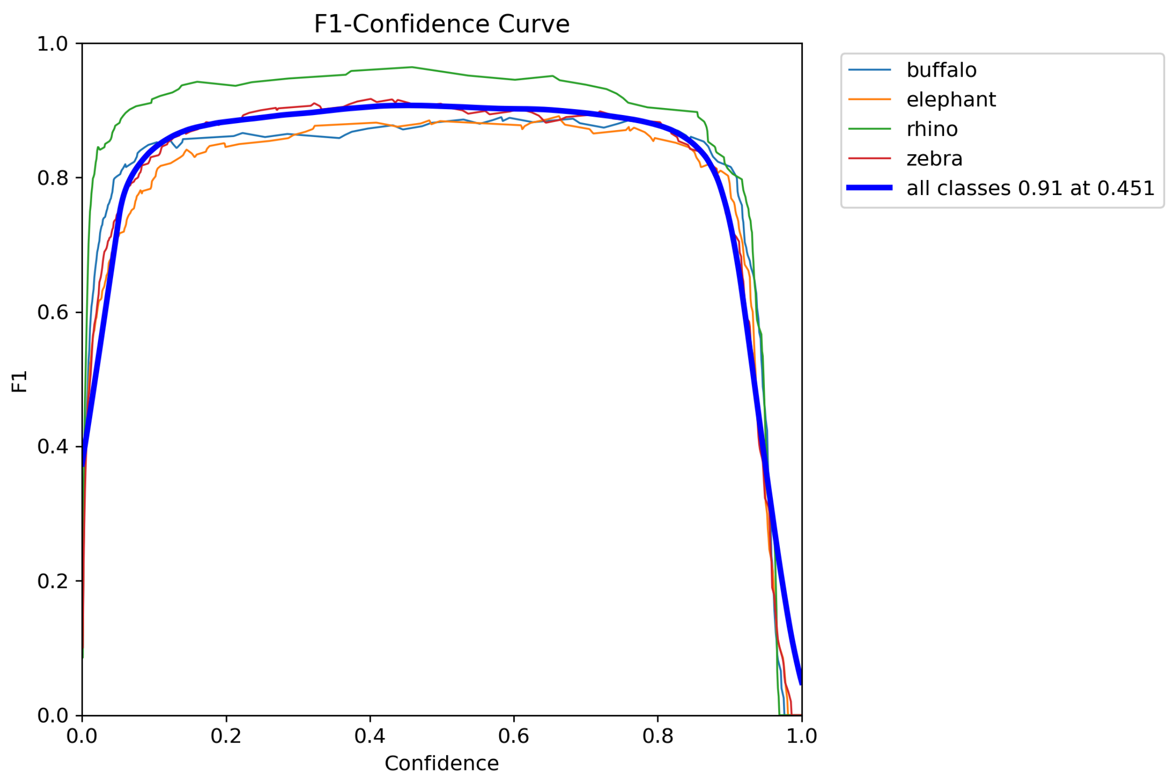
the f1 curve shows how the f1 score changes with different thresholds in a classification model. the f1 score is the harmonic mean of precision and recall, which balances how many positive predictions are correct (precision) and how many actual positives are found (recall).

as the decision threshold changes, the model predicts positive less or more strictly. this affects precision and recall differently, so the f1 score varies too. plotting the f1 score against these thresholds gives the f1 curve.

the curve helps find the best threshold where the balance of precision and recall is optimal, giving the highest f1 score. this is useful when both false positives and false negatives are important to minimize.

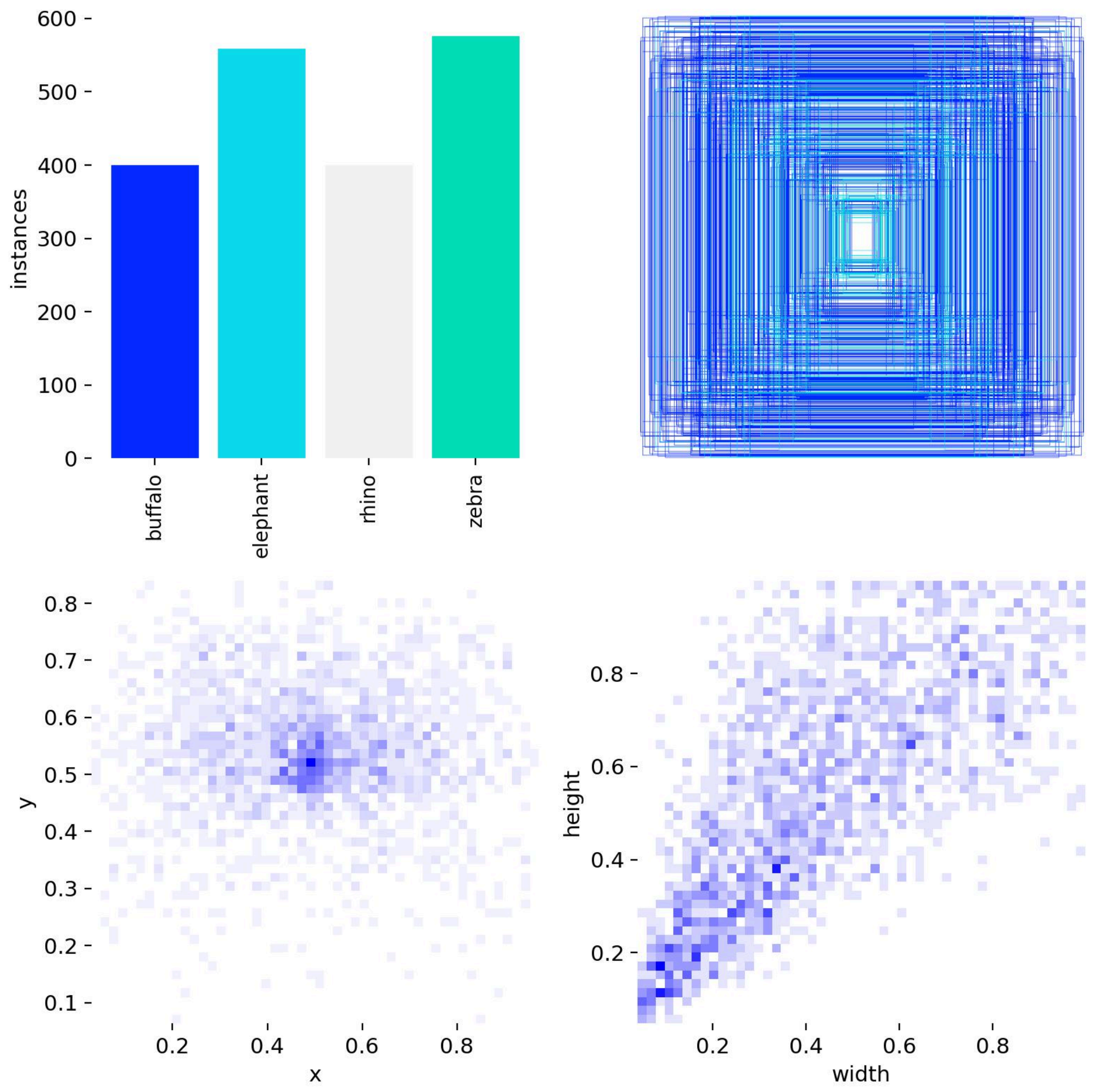
in summary, the f1 curve visualizes how well the model performs at different thresholds, helping select the threshold that gives the best tradeoff between precision and recall.





he graph shows that this model is generally very good at identifying these animals when it has a moderate to high level of confidence in its predictions. the sweet spot for the overall best performance is around 45% confidence, where it achieves an excellent f1 score of 0.91. if the model becomes too uncertain (low confidence) or too certain (very high confidence), its performance tends to drop.

5. labels



top left (bar chart):

i can see that the number of instances is not equal for all classes. zebra and elephant have the most instances — both close to 600. buffalo and rhino have fewer, around 400 each. this tells me that the dataset is slightly imbalanced, with zebra and elephant appearing more often.

top right (overlay of bounding boxes):

this plot looks like a bunch of blue rectangles stacked over each other. these are all the bounding boxes drawn together. i noticed that most of the boxes are centered and there's a dense cluster in the middle. it suggests that the animals are usually in the center of the image, and the boxes are generally square-shaped.

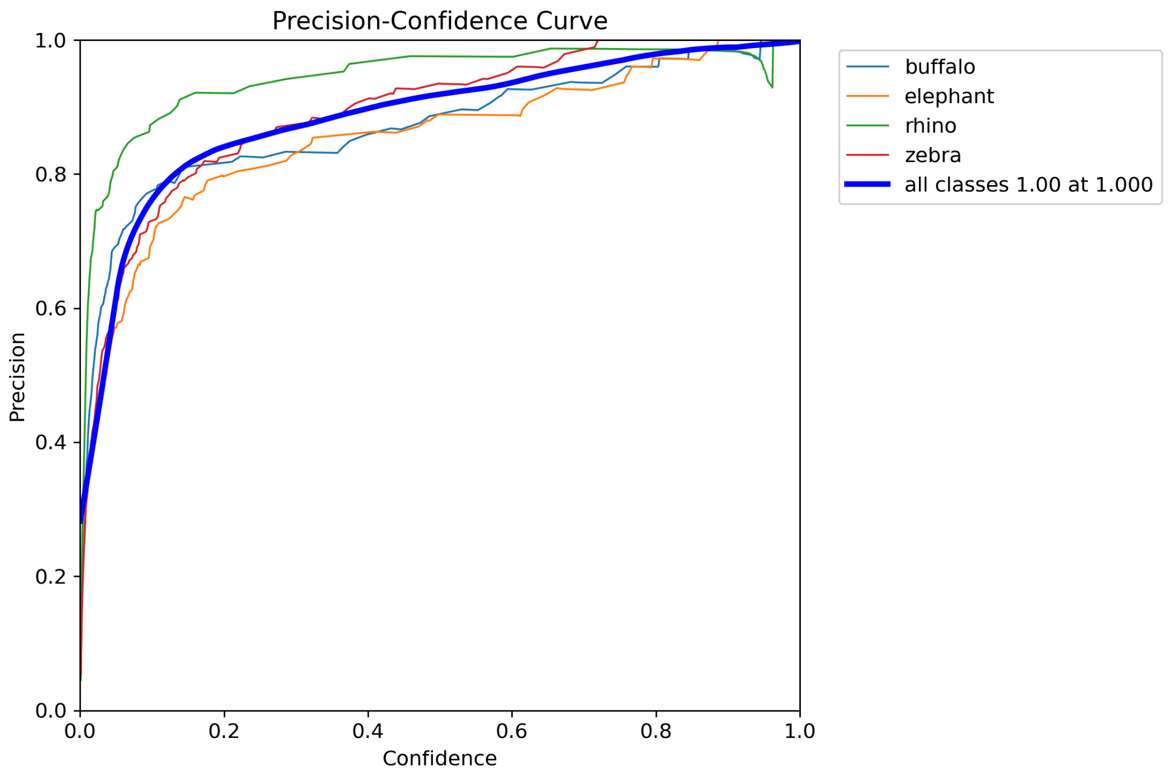
bottom left (x vs y heatmap):

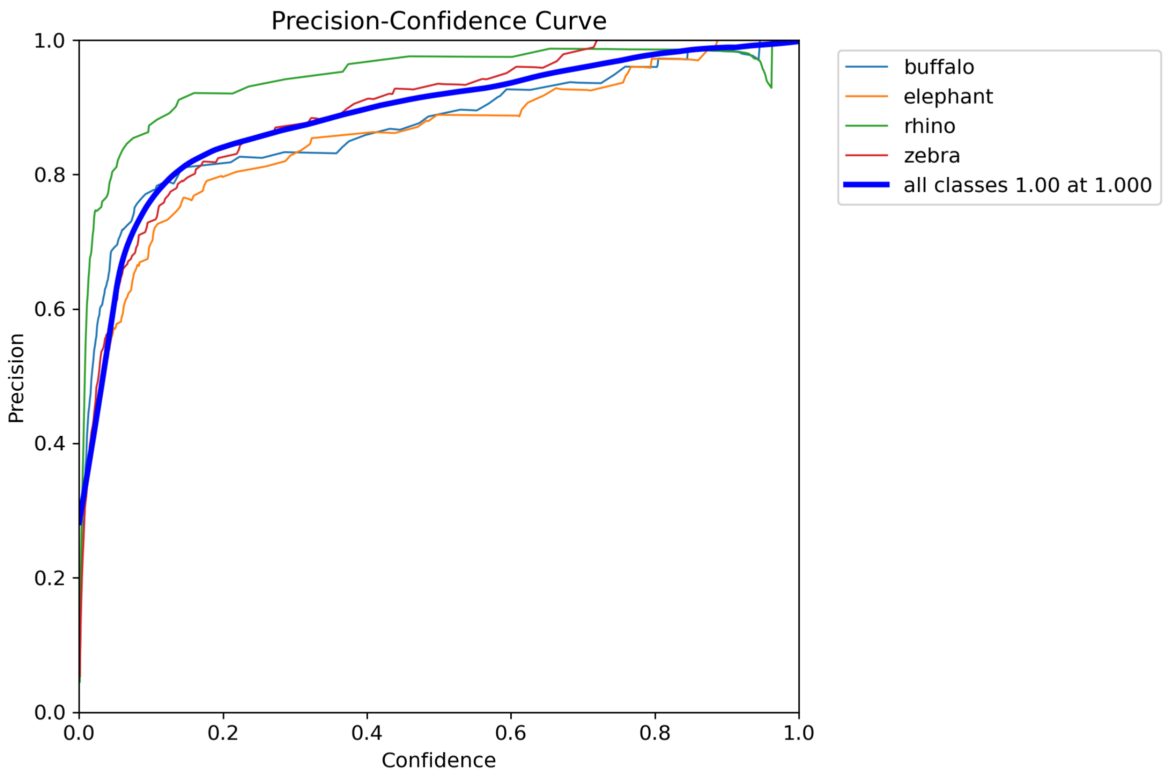
this heatmap shows where the objects are located in terms of x and y positions. i see a dark spot in the center, which means most bounding box centers are near the middle of the image. the density decreases as we move away from the center. again, this confirms that the animals are usually photographed in the center.

bottom right (width vs height heatmap):

this heatmap shows how big the bounding boxes are. i see that most boxes have widths and heights between 0.1 and 0.4. also, there's a diagonal trend, meaning that as width increases, height increases too. so the bounding boxes are keeping a similar shape — maybe because the animals are proportionate.

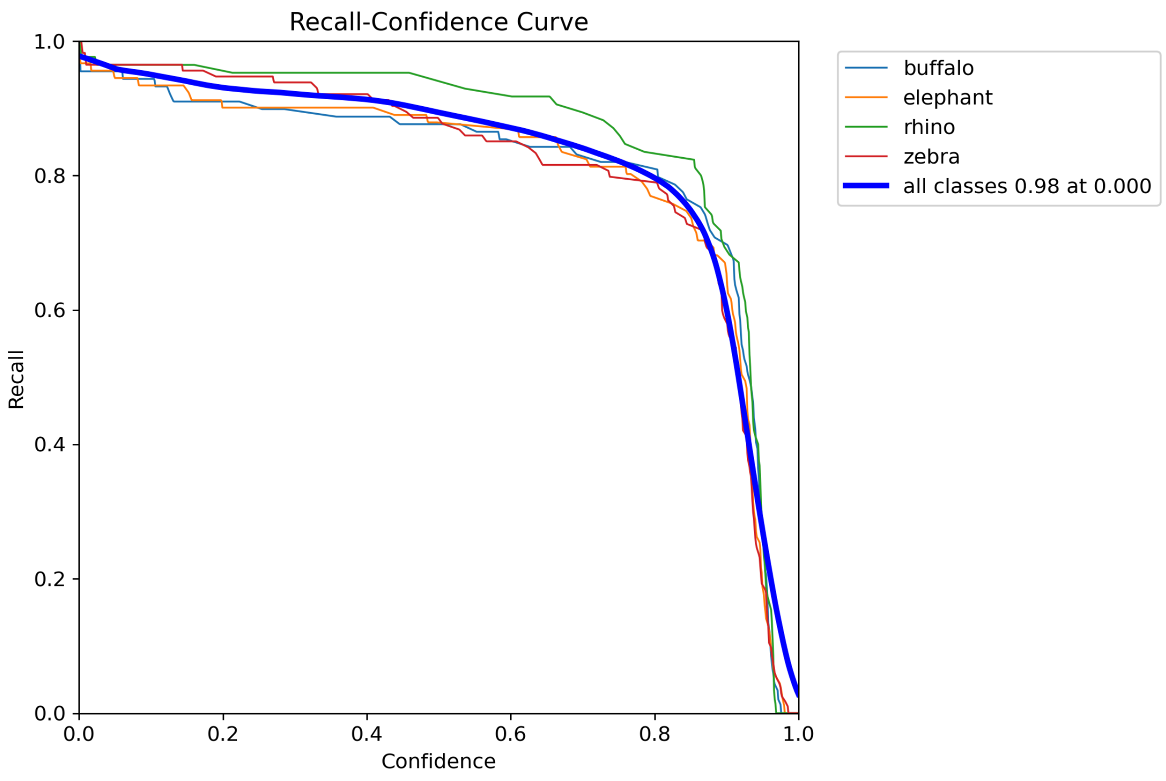
6. P CURVE



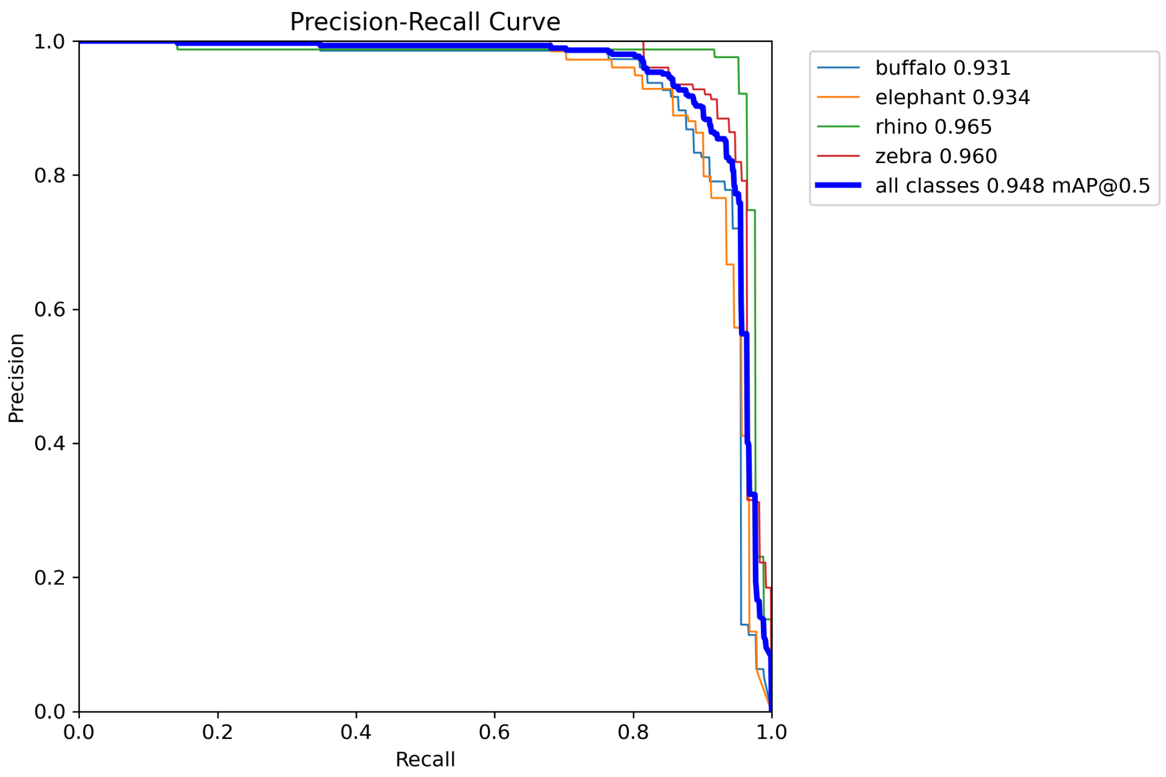


when i looked at this precision-confidence curve, i noticed that the curves start at a low precision value when the confidence is near zero, and then they gradually increase. this happens because at lower confidence levels, the mofel is making more uncertain predictions, so it's likely to include a lot of false positives — that’s wh precision is low. as the confidence threshold increases, the model becomes more selective, only keeping predictions it's more certai about. this reduces false positives, so precision improves. eventually, the curve become almost flat or constant near the top right corner. that’s because at high confidence levels, the predictions are very accurate — the model is almost always right when it makes a prediction with such high confidence. so, the precision reaches close to 1 and stays stable. this trend shows that the model performs well when it’s confident, and that the high-confidence predictions are reliable.

7. pr curve and r curve



in both graphs, the curves are decreasing because of the trade-offs that naturally occur in classification models, especially object detection tasks like this one.



in the **precision-recall curve**, as recall increases, i'm including more predictions that try to catch all the true positives. but when i do that, i'm also pulling in more false positives, which causes precision to drop. so it's a balancing act—recall goes up when i'm less strict, but that same looseness can hurt precision. this inverse relationship is why the curve slopes downward.

in the **recall-confidence curve**, confidence refers to how sure the model is about a prediction. when confidence is high, only the most certain predictions are considered, so recall is lower because i'm ignoring many true positives with lower confidence. as i lower the confidence threshold, i allow more predictions, which increases recall—but i also start including less reliable results. since the x-axis is confidence and i'm moving from high to low confidence, recall goes up, but the line appears to go down when plotted this way because high-confidence predictions cover fewer true positives.

both curves reflect how the model behaves under different thresholds—either for detection confidence or for balancing between precision and recall. the "all classes" thick blue line gives a smoothed average of performance across categories, which helps understand overall behavior.