Projet Réalité mixte et vision 3D

Fine Tuning YOLOv8n-pose on AP-10K

Work made by : Fedi Koubaa and Jihen Fennani RT4

In this notebook, we aim to adapt the YOLOv8n-pose model, originally designed for human pose estimation (HPE), to accurately detect and estimate animal poses across various species using the AP-10K dataset.

The primary objective of this work is to explore the generalization capabilities of the YOLOv8n-pose model to different animal families. By fine-tuning the YOLOv8n-pose model on the AP-10K dataset, we aim to assess its ability to accurately detect and estimate poses across a diverse range of animal species.

```
!pip install -q tensorboard ultralytics
```

```
import os
import time
import json
import requests
from zipfile import ZipFile
import tarfile
from shutil import copyfile
from dataclasses import dataclass, field

import yaml
import glob

import random
import numpy as np
import pandas as pd
import cv2

from ultralytics import YOLO

import matplotlib.pyplot as plt
```

Data Preperation and EDA

```
!gdown 1-FNNGcdtAQRehYYkGY1y4wzFNg4iWNad
Downloading...
From (original): https://drive.google.com/uc?id=1-FNNGcdtAQRehYYkGY1y4wzFNg4iWNad
```

```
From (redirected): https://drive.google.com/uc?id=1-FNNGcdtAQRehYYkGY1y4wzFNg4iWNad&confirm=t&uuid=7443623e-d111-4d8a-b2f3-501fad482353
To: /home/studio-lab-user/PFA/ap-10k.zip
100%|██████████████████████████████████████| 3.40G/3.40G [01:03<00:00, 54.0MB/s]
```

```
!unzip ap-10k.zip
```

```
ls ap-10k/
annotations/   data/
```

```
ls ap-10k/annotations
ap10k-test-split1.json  ap10k-train-split1.json  ap10k-val-split1.json
ap10k-test-split2.json  ap10k-train-split2.json  ap10k-val-split2.json
ap10k-test-split3.json  ap10k-train-split3.json  ap10k-val-split3.json
```

```
cat ap-10k/annotations/ap10k-test-split1.json
```

```python
import json
import os

def merge_json_files(file_path):
    merged_data = {
        'images': [],
        'annotations': [],
        'categories': []
        }
    with open(file_path, 'r') as file:
      data = json.load(file)
      merged_data['images'].extend(data.get('images', []))
      merged_data['annotations'].extend(data.get('annotations', []))
      merged_data['categories'].extend(data.get('categories', []))
    return merged_data

base_folder = 'ap-10k/annotations/'

i=1 # split 1

test_file = os.path.join(base_folder, f'ap10k-test-split{i}.json')
train_file = os.path.join(base_folder, f'ap10k-train-split{i}.json')
val_file = os.path.join(base_folder, f'ap10k-val-split{i}.json')

# Fusion des fichiers
```

```
test_data = merge_json_files(test_file)
train_data = merge_json_files(train_file)
val_data = merge_json_files(val_file)
```

```
for key, value in train_data.items():
    print(key)
images
annotations
categories
```

```
i=1
print(train_data['annotations'][i])
print(train_data['images'][i])
print(train_data['categories'][i])
print(len(train_data['annotations'][i]['keypoints'])/3)
print(len(train_data['annotations'][1]['keypoints'])/train_data['annot
ations'][1]['num_keypoints']+1)

{'id': 273, 'image_id': 178, 'category_id': 1, 'bbox': [155, 214, 463,
450], 'area': 208350, 'iscrowd': 0, 'num_keypoints': 13, 'keypoints':
[204, 312, 2, 0, 0, 0, 163, 353, 2, 290, 391, 2, 0, 0, 0, 363, 469, 2,
376, 536, 2, 402, 630, 2, 278, 479, 2, 255, 542, 2, 225, 628, 2, 0, 0,
0, 541, 512, 2, 525, 604, 2, 0, 0, 0, 582, 482, 2, 592, 561, 2]}
{'license': 1, 'id': 178, 'file_name': '000000000178.jpg', 'width':
1024, 'height': 678, 'background': 1}
{'id': 2, 'name': 'argali sheep', 'supercategory': 'Bovidae',
'keypoints': ['left_eye', 'right_eye', 'nose', 'neck', 'root_of_tail',
'left_shoulder', 'left_elbow', 'left_front_paw', 'right_shoulder',
'right_elbow', 'right_front_paw', 'left_hip', 'left_knee',
'left_back_paw', 'right_hip', 'right_knee', 'right_back_paw'],
'skeleton': [[1, 2], [1, 3], [2, 3], [3, 4], [4, 5], [4, 6], [6, 7],
[7, 8], [4, 9], [9, 10], [10, 11], [5, 12], [12, 13], [13, 14], [5,
15], [15, 16], [16, 17]]}
17.0
4.923076923076923
```

```
print(len(test_data['images']))
print(len(test_data['annotations']))
print(len(test_data['categories']))
1997
2634
54
```

```
print(len(train_data['images']))
print(len(train_data['annotations']))
print(len(train_data['categories']))
7023
9122
```

54

```
# Afficher toutes les images décompressées
import os
from PIL import Image
import matplotlib.pyplot as plt

# Répertoire contenant les images décompressées
image_directory = 'ap-10k/data'
images = os.listdir(image_directory)

# S'assurer que la sortie ne soit pas trop volumineuse pour un
affichage pratique
print(f"Total images: {len(images)}")
max_images_to_show = 20  # Vous pouvez ajuster ce nombre

# Afficher les images
for i, image_name in enumerate(images):
    if i >= max_images_to_show:
        print("Affichage limité aux premières 20 images.")
        break
    image_path = os.path.join(image_directory, image_name)
    img = Image.open(image_path)
    plt.figure()
    plt.imshow(img)
    plt.axis('off')
    plt.title(image_name)
    plt.show()
```

```
Total images: 10015
```

We will maintain the following directory

structure for YOLOv8 dataset:

```
animal-pose-data
├── train
│   ├── images (6773 files)
│   └── labels (6773 files)
└── valid
    ├── images (1703 files)
    └── labels (1703 files)
```

```
DATA_DIR = "animal-pose-data"

TRAIN_DIR           = f"train"
TRAIN_FOLDER_IMG     = f"images"
TRAIN_FOLDER_LABELS = f"labels"

TRAIN_IMG_PATH   = os.path.join(DATA_DIR, TRAIN_DIR, TRAIN_FOLDER_IMG)
```

```
TRAIN_LABEL_PATH = os.path.join(DATA_DIR, TRAIN_DIR,
TRAIN_FOLDER_LABELS)

VALID_DIR           = f"valid"
VALID_FOLDER_IMG    = f"images"
VALID_FOLDER_LABELS = f"labels"

VALID_IMG_PATH   = os.path.join(DATA_DIR, VALID_DIR, VALID_FOLDER_IMG)
VALID_LABEL_PATH = os.path.join(DATA_DIR, VALID_DIR,
VALID_FOLDER_LABELS)

os.makedirs(TRAIN_IMG_PATH, exist_ok=True)
os.makedirs(TRAIN_LABEL_PATH, exist_ok=True)
os.makedirs(VALID_IMG_PATH, exist_ok=True)
os.makedirs(VALID_LABEL_PATH, exist_ok=True)
```

3.1 Copy Image files

```
# for data in train_data['images']:
#     img_file = data["file_name"]
#     filename = img_file.split("/")[-1]
#     copyfile(os.path.join("ap-10k/data/", img_file),
#              os.path.join(TRAIN_IMG_PATH, filename))


# for data in test_data['images']:
#     img_file = data["file_name"]
#     filename = img_file.split("/")[-1]
#     copyfile(os.path.join("ap-10k/data/", img_file),
#              os.path.join(VALID_IMG_PATH, filename))
```

3.2 Create YOLO Annotatio3.2 Create YOLO Annotation TXT FILES

Our final task for data preparation is to create the boxes and the keypoint annotations in accordance with Ultralytics' YOLO. Since we will deal with a single class (i.e., animal), we set the class index to `0`.

```
CLASS_ID = 0
```

The function `create_yolo_boxes_kpts` performs the following tasks:

- Modifies visibility indicators for keypoints (setting the visibilities for labeled keypoints to 2).
- Normalizes the coordinates of both bounding boxes and keypoints relative to the image dimensions.
- Converts bounding boxes to $[x_{center},\ ,y_{center},\ width,\ height]$ in normalized form.

```python
def create_yolo_txt_files(data, LABEL_PATH):
    for item in data['annotations']:
        # print(item)
        image_id = item['image_id']
        image_data = next((img for img in data['images'] if img['id']
== image_id), None)
        # print(image_data)
        if not image_data:
          print(f'image of id {image_id} not found')
          continue

        img_width, img_height = image_data['width'],
image_data['height']
        x_min, y_min, bbox_width, bbox_height = item['bbox']

        # Conversion en coordonnées centrées et normalisées
        x_center = round((x_min + bbox_width / 2) / img_width,5)
        y_center = round((y_min + bbox_height / 2) / img_height,5)
        width_norm = round(bbox_width / img_width,5)
        height_norm = round(bbox_height / img_height,5)

        # Extract keypoints
        keypoints = item['keypoints']
        num_keypoints = item['num_keypoints']
        keypoints_visibilities = [2 if keypoints[i+2] > 0 else 0 for i
in range(0, len(keypoints), 3)]
        landmark_kpts = [(keypoints[i], keypoints[i+1],
keypoints_visibilities[i//3]) for i in range(0, len(keypoints), 3)]
        # print("keypoints",keypoints)
        # print("landmark_kpts",landmark_kpts)
        # print("len landmark_kpts",len(landmark_kpts))
        # Format keypoints into YOLO format
        kpts_yolo = np.array(landmark_kpts, dtype=np.float32) /
np.array([img_width, img_height,1])

        # Append keypoints to the YOLO annotation string
        kpts_flattend = [round(ele, 5) for ele in
kpts_yolo.flatten().tolist()]
        # print(kpts_flattend)
        # print(len(kpts_flattend))
        TXT_FILE = image_data["file_name"].split(".")[0]+".txt"
        with open(os.path.join(LABEL_PATH, TXT_FILE), "w") as f:
              line = f"{CLASS_ID} {x_center} {y_center} {width_norm}
{height_norm} "
              line+= " ".join(map(str, kpts_flattend))
              f.write(line)
#          break
# create_yolo_txt_files(train_data, TRAIN_LABEL_PATH)
```

We will finally create the `txt` files for YOLO based on the `train_json_data` and `val_json_data` obtained earlier. The function `create_yolo_txt_files` creates the required `txt` annotations in YOLO using the `create_yolo_boxes_kpts` utility function explained above.

```
create_yolo_txt_files(train_data, TRAIN_LABEL_PATH)
create_yolo_txt_files(test_data, VALID_LABEL_PATH)
```

4 Data Visualization

```
train_images = os.listdir(TRAIN_IMG_PATH)
valid_images = os.listdir(VALID_IMG_PATH)


train_labels = os.listdir(TRAIN_LABEL_PATH)
valid_labels = os.listdir(VALID_LABEL_PATH)
print(f"Training images: {len(train_images)}, Validation Images:
{len(valid_images)}")
print(f"Training labels: {len(train_labels)}, Validation Labels:
{len(valid_labels)}")
print(len(train_data['images']))
print(len(test_data['images']))
```

```
Training images: 7023, Validation Images: 1997
Training labels: 7023, Validation Labels: 1997
7023
1997
```

The `draw_landmarks` function is used to annotate the corresponding landmark points on the image using COLORS_RGB_MAP.

```
def draw_landmarks(image, landmarks):

    radius = 5
    # Check if image width is greater than 1000 px.
    # To improve visualization.
    if (image.shape[1] > 1000):
        radius = 8

    for idx, kpt_data in enumerate(landmarks):

        loc_x, loc_y = kpt_data[:2].astype("int").tolist()

        cv2.circle(image,
                   (loc_x, loc_y),
                   radius,
                   color=(0,0,255),
```

```
                    thickness=-1,
                    lineType=cv2.LINE_AA)

    return image
```

The `draw_boxes` function is used to annotate the bounding boxes along with the confidence scores (if passed) on the image.

```
def draw_boxes(image, detections, class_name = "animal", score=None,
color=(0,255,0)):

    font_size = 0.25 + 0.07 * min(image.shape[:2]) / 100
    font_size = max(font_size, 0.5)
    font_size = min(font_size, 0.8)
    text_offset = 3

    thickness = 2
    # Check if image width is greater than 1000 px.
    # To improve visualization.
    if (image.shape[1] > 1000):
        thickness = 10

    xmin, ymin, xmax, ymax = detections[:4].astype("int").tolist()
    conf = round(float(detections[-1]),2)
    cv2.rectangle(image,
                  (xmin, ymin),
                  (xmax, ymax),
                  color=(0,255,0),
                  thickness=thickness,
                  lineType=cv2.LINE_AA)

    display_text = f"{class_name}"

    if score is not None:
        display_text+=f": {score:.2f}"

    (text_width, text_height), _ = cv2.getTextSize(display_text,
                                                   cv2.FONT_HERSHEY_SI
MPLEX,
                                                   font_size, 2)

    cv2.rectangle(image,
                      (xmin, ymin),
                      (xmin + text_width + text_offset, ymin -
text_height - int(15 * font_size)),
                      color=color, thickness=-1)
```

```
    image = cv2.putText(
                    image,
                    display_text,
                    (xmin + text_offset, ymin - int(10 * font_size)),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    font_size,
                    (0, 0, 0),
                    2, lineType=cv2.LINE_AA,
                )

    return image
```

The `visualize_annotations` is used to annotate both the bounding box coordinates and the landmark keypoints on the corresponding image after converting them to absoulute coordinates.

Recall that both the bounding box coordinates and the keypoints were normalized in the range `[0, 1]`. However, to plot them, we need the absolute coordinates.

The conversion mapping from YOLO bboxes to $[x_{min}, y_{min}, x_{max}, y_{max}]$ is pretty straight forward and can be obtained using the following set of equations:

$$x_{min} = \frac{W}{2}(2x_{center} - width)$$

$$y_{min} = \frac{H}{2}(2y_{center} - height)$$

$$x_{max} = x_{min} + width * W$$

$$y_{max} = y_{min} + height * H$$

Similarly, the keypoints can denormalized (to the absolute coordinates) using:

$$x_{abs} = x_{norm} * W$$

$$y_{abs} = y_{norm} * H$$

Here, the `width` and `height` are the box width and height respectively; whereas `W` and `H` are the image width and height respectively.

```
def visualize_annotations(image, box_data, keypoints_data):

    image = image.copy()

    shape_multiplier = np.array(image.shape[:2][::-1]) # (W, H).
    # Final absolute coordinates (xmin, ymin, xmax, ymax).
    denorm_boxes = np.zeros_like(box_data)
```

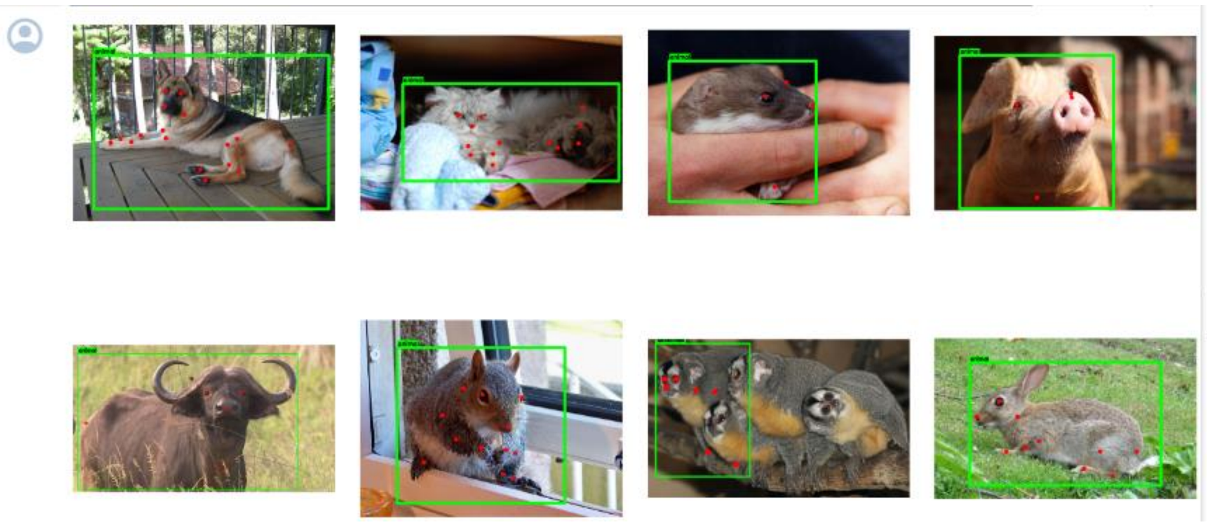
```
    # De-normalize center coordinates from YOLO to (xmin, ymin).
    denorm_boxes[:, :2] = (shape_multiplier/2.) * (2*box_data[:,:2] -
box_data[:,2:])

    # De-normalize width and height from YOLO to (xmax, ymax).
    denorm_boxes[:, 2:] = denorm_boxes[:,:2] +
box_data[:,2:]*shape_multiplier

    for boxes, kpts in zip(denorm_boxes, keypoints_data):
        # De-normalize landmark coordinates.
        kpts[:, :2]*= shape_multiplier
        image = draw_boxes(image, boxes)
        image = draw_landmarks(image, kpts)

    return image
```

The following plot shows a few image samples with their corresponding ground truth annotation. The keypoint annotations are filtered based on their corresponding visibility flag.

addCode
addTexte

```
IMAGE_FILES = os.listdir(TRAIN_IMG_PATH)
NUM_LANDMARKS = 17

num_samples = 8
num_rows = 2
num_cols = num_samples//num_rows

fig, ax = plt.subplots(
        nrows=num_rows,
        ncols=num_cols,
        figsize=(25, 15),
    )

random.seed(45)
random.shuffle(IMAGE_FILES)

for idx, (file, axis) in enumerate(zip(IMAGE_FILES[:num_samples],
ax.flat)):

    image = cv2.imread(os.path.join(TRAIN_IMG_PATH, file))

    # Obtain the txt file for the corresponding image file.
    filename = file.split(".")[0]
    # Split each object instance in separate lists.
```

```
    with open(os.path.join(TRAIN_LABEL_PATH, filename+".txt"), "r") as
file:
        label_data = [x.split() for x in
file.read().strip().splitlines() if len(x)]

    label_data = np.array(label_data, dtype=np.float32)
    # YOLO BBox instances in [x-center, y-center, width, height] in
normalized form.
    box_instances = label_data[:,1:5]
    # Shape: (N, 4), where, N = #instances per-image
    # Kpt instances.
    # Filter keypoints based on visibility.
    instance_kpts = []
    kpts_data = label_data[:,5:].reshape(-1, NUM_LANDMARKS, 3)

    for inst_kpt in kpts_data:
        vis_ids = np.where(inst_kpt[:, -1]>0.)[0]
        vis_kpts = inst_kpt[vis_ids][:,:2]
        vis_kpts = np.concatenate([vis_kpts, np.expand_dims(vis_ids,
axis=-1)], axis=-1)
        instance_kpts.append(vis_kpts)

    image_ann = visualize_annotations(image, box_instances,
instance_kpts)
    axs.imshow(image_ann[...,::-1])
    axis.axis("off")


plt.tight_layout(h_pad=4., w_pad=4.)
plt.show();
```



5 Configurations

5.1 Training Configuration

We shall define the training configuration for fine-tuning in the `TrainingConfig` class.

```
@dataclass(frozen=True)
class TrainingConfig:
    DATASET_YAML:   str = "animal-keypoints.yaml"
    MODEL:          str = "yolov8n-pose.pt"
    EPOCHS:         int = 50
    KPT_SHAPE:    tuple = (17,3)
    PROJECT:        str = "Animal_Keypoints"
    NAME:           str = f"{MODEL.split('.')[0]}_{EPOCHS}_epochs"
    CLASSES_DICT:  dict = field(default_factory = lambda:{0 :
"animal"})
```

5.2 Data Configuration

The `DatasetConfig` class takes in the various hyperparameters related to the data such as the image size and batch size to be used while training, along with the various augmentation probabilities such as Mosaic, horizontal flip, etc.

```
@dataclass(frozen=True)
class DatasetConfig:
    IMAGE_SIZE:    int   = 640
    BATCH_SIZE:    int   = 16
    CLOSE_MOSAIC:  int   = 10
    MOSAIC:        float = 0.4
    FLIP_LR:       float = 0.0 # Turn off horizontal flip.
train_config = TrainingConfig()
data_config = DatasetConfig()
```

Before we start our training, we need to create a `yaml` containing the path to the images and label files. We also need to specify the class names, starting from index=0 and the keypoint shape.

```
current_dir = os.getcwd()
print(current_dir)
data_dict = dict(
                path      = os.path.join(current_dir, DATA_DIR),
                train     = os.path.join(TRAIN_DIR, TRAIN_FOLDER_IMG),
                val       = os.path.join(VALID_DIR, VALID_FOLDER_IMG),
                names     = train_config.CLASSES_DICT,
                kpt_shape = list(train_config.KPT_SHAPE),
               )
```

```
with open(train_config.DATASET_YAML, "w") as config_file:
    yaml.dump(data_dict, config_file)

/home/studio-lab-user/PFA
```

6 Training

```
pose_model = model = YOLO(train_config.MODEL)

pose_model.train(data     = train_config.DATASET_YAML,
            epochs       = train_config.EPOCHS,
            imgsz        = data_config.IMAGE_SIZE,
            batch        = data_config.BATCH_SIZE,
            project      = train_config.PROJECT,
            name         = train_config.NAME,
            close_mosaic = data_config.CLOSE_MOSAIC,
            mosaic       = data_config.MOSAIC,
            fliplr       = data_config.FLIP_LR
           )
```

```
Ultralytics YOLOv8.2.11 🚀 Python-3.10.14 torch-2.0.0.post200 CUDA:0
(Tesla T4, 14931MiB)
engine/trainer: task=pose, mode=train, model=yolov8n-pose.pt,
data=animal-keypoints.yaml, epochs=50, time=None, patience=100,
batch=16, imgsz=640, save=True, save_period=-1, cache=False,
device=None, workers=8, project=Animal_Keypoints, name=yolov8n-
pose_50_epochs3, exist_ok=False, pretrained=True, optimizer=auto,
verbose=True, seed=0, deterministic=True, single_cls=False, rect=False,
cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0,
profile=False, freeze=None, multi_scale=False, overlap_mask=True,
mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False,
save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False,
dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=False,
visualize=False, augment=False, agnostic_nms=False, classes=None,
retina_masks=False, embed=None, show=False, save_frames=False,
save_txt=False, save_conf=False, save_crop=False, show_labels=True,
show_conf=True, show_boxes=True, line_width=None, format=torchscript,
keras=False, optimize=False, int8=False, dynamic=False, simplify=False,
opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937,
weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8,
warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0,
label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4,
degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0,
flipud=0.0, fliplr=0.0, bgr=0.0, mosaic=0.4, mixup=0.0, copy_paste=0.0,
auto_augment=randaugment, erasing=0.4, crop_fraction=1.0, cfg=None,
tracker=botsort.yaml, save_dir=Animal_Keypoints/yolov8n-pose_50_epochs3

                   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, True]
  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, True]
  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   1035934  ultralytics.nn.modules.head.Pose
[1, [17, 3], [64, 128, 256]]
YOLOv8n-pose summary: 250 layers, 3295470 parameters, 3295454
gradients, 9.3 GFLOPs

Transferred 397/397 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir Animal_Keypoints/yolov8n-
pose_50_epochs3', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
```

7 Evaluation

```
ckpt_path  = os.path.join(train_config.PROJECT, train_config.NAME,
"weights", "best.pt")
model_pose = YOLO(ckpt_path)

metrics = model_pose.val()
```

```
Ultralytics YOLOv8.2.11 🚀 Python-3.10.14 torch-2.0.0.post200 CUDA:0 (Tesla T4, 14931MiB)
YOLOv8n-pose summary (fused): 187 layers, 3289964 parameters, 0 gradients, 9.2 GFLOPs
val: Scanning /home/studio-lab-user/PFA/animal-pose-data/valid/labels.cache... 1997 images, 0
backgrounds, 0 corrupt: 100%|██████████| 1997/1997 [00:00<?, ?it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Pose(P          R      mAP50
mAP50-95): 100%|██████████| 125/125 [00:19<00:00,  6.56it/s]
             all       1997       1997      0.877      0.871      0.921      0.696      0.675      0.601      0.545      0.195
Speed: 0.3ms preprocess, 3.5ms inference, 0.0ms loss, 0.7ms postprocess per image
Results saved to runs/pose/val
```

8 Predictions

The `prepare_predictions` function obtains the predicted boxes, confidence scores, and keypoints for the corresponding image.

```
def prepare_predictions(
    image_dir_path,
    image_filename,
    model,
    BOX_IOU_THRESH = 0.55,
    BOX_CONF_THRESH=0.30,
    KPT_CONF_THRESH=0.68):

    image_path = os.path.join(image_dir_path, image_filename)
    image = cv2.imread(image_path).copy()

    results = model.predict(image_path, conf=BOX_CONF_THRESH,
iou=BOX_IOU_THRESH)[0].cpu()

    if not len(results.boxes.xyxy):
        return image

    # Get the predicted boxes, conf scores and keypoints.
    pred_boxes = results.boxes.xyxy.numpy()
    pred_box_conf = results.boxes.conf.numpy()
    pred_kpts_xy = results.keypoints.xy.numpy()
    pred_kpts_conf = results.keypoints.conf.numpy()

    # Draw predicted bounding boxes, conf scores and keypoints on
image.
    for boxes, score, kpts, confs in zip(pred_boxes, pred_box_conf,
pred_kpts_xy, pred_kpts_conf):
```

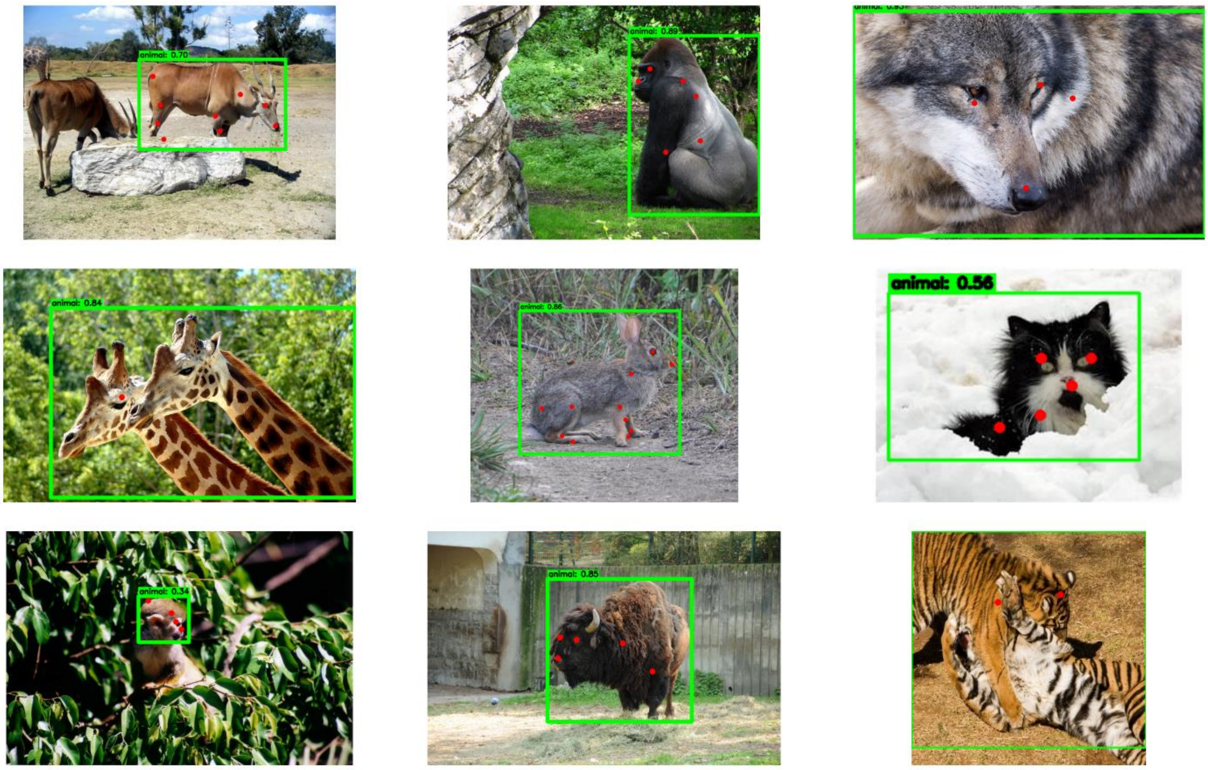
```
        kpts_ids = np.where(confs > KPT_CONF_THRESH)[0]
        filter_kpts = kpts[kpts_ids]
        filter_kpts = np.concatenate([filter_kpts,
np.expand_dims(kpts_ids, axis=-1)], axis=-1)
        image = draw_boxes(image, boxes, score=score)
        image = draw_landmarks(image, filter_kpts)

    return image
```

```
VAL_IMAGE_FILES = os.listdir(VALID_IMG_PATH)

num_samples = 9
num_rows = 3
num_cols = num_samples//num_rows

fig, ax = plt.subplots(
        nrows=num_rows,
        ncols=num_cols,
        figsize=(25, 15),
    )

random.seed(90)
random.shuffle(VAL_IMAGE_FILES)

for idx, (file, axis) in enumerate(zip(VAL_IMAGE_FILES[:num_samples],
ax.flat)):

    image_pred = prepare_predictions(VALID_IMG_PATH, file, model_pose)
    axis.imshow(image_pred[...,::-1])
    axis.axis("off")

plt.tight_layout(h_pad=4., w_pad=4.)
plt.show();
```

```
image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000000151.jpg: 480x640 1 animal, 60.2ms
Speed: 2.4ms preprocess, 60.2ms inference, 1.8ms postprocess per image at
shape (1, 3, 480, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000046263.jpg: 480x640 1 animal, 7.1ms
Speed: 2.3ms preprocess, 7.1ms inference, 1.4ms postprocess per image at
shape (1, 3, 480, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000019254.jpg: 448x640 1 animal, 61.9ms
Speed: 2.4ms preprocess, 61.9ms inference, 1.3ms postprocess per image at
shape (1, 3, 448, 640)
```

YOLOv8n-pose on AP-10K

```
image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000043197.jpg: 448x640 1 animal, 7.0ms
Speed: 2.2ms preprocess, 7.0ms inference, 1.3ms postprocess per image at
shape (1, 3, 448, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000047473.jpg: 576x640 1 animal, 61.0ms
Speed: 2.8ms preprocess, 61.0ms inference, 1.3ms postprocess per image at
shape (1, 3, 576, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000035084.jpg: 512x640 1 animal, 61.1ms
Speed: 2.0ms preprocess, 61.1ms inference, 1.3ms postprocess per image at
shape (1, 3, 512, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000020126.jpg: 448x640 1 animal, 7.7ms
Speed: 2.1ms preprocess, 7.7ms inference, 1.3ms postprocess per image at
shape (1, 3, 448, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000001173.jpg: 448x640 1 animal, 7.0ms
Speed: 2.1ms preprocess, 7.0ms inference, 1.3ms postprocess per image at
shape (1, 3, 448, 640)

image 1/1 /home/studio-lab-user/PFA/animal-pose-
data/valid/images/000000041150.jpg: 640x640 1 animal, 9.2ms
Speed: 2.5ms preprocess, 9.2ms inference, 1.3ms postprocess per image at
shape (1, 3, 640, 640)
```




```python
import os
import matplotlib.pyplot as plt
```

```
def show_training_results(image_folder):
    # List all image files in the folder
    image_files = [file for file in os.listdir(image_folder) if
file.endswith(('.jpg', '.png'))]

    # Plot each image
    for image_file in image_files:
        image_path = os.path.join(image_folder, image_file)
        image = plt.imread(image_path)

        plt.imshow(image)
        plt.title(image_file)
        plt.axis('off')
        plt.show()

# Specify the folder containing the training result images
result_folder = 'runs/pose/val/'

# Show the training result images
show_training_results(result_folder)
```

val_batch0_labels.jpg

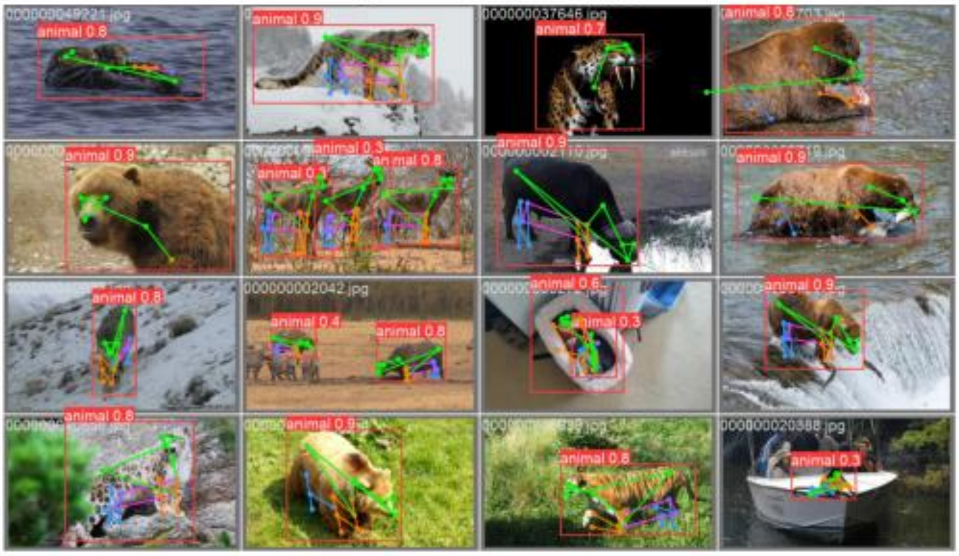
val_batch0_pred.jpg

val_batch1_labels.jpg

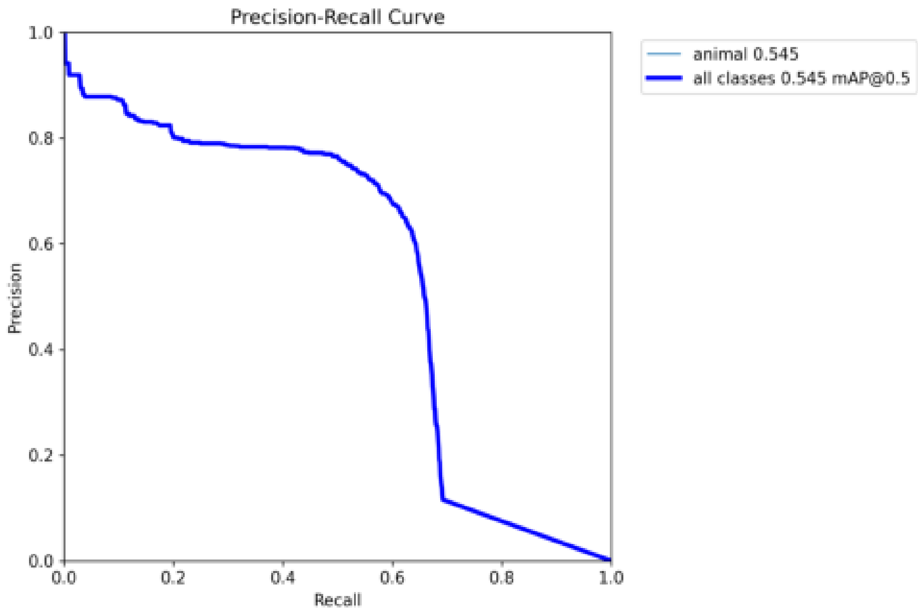
val_batch1_pred.jpg

val_batch2_labels.jpg

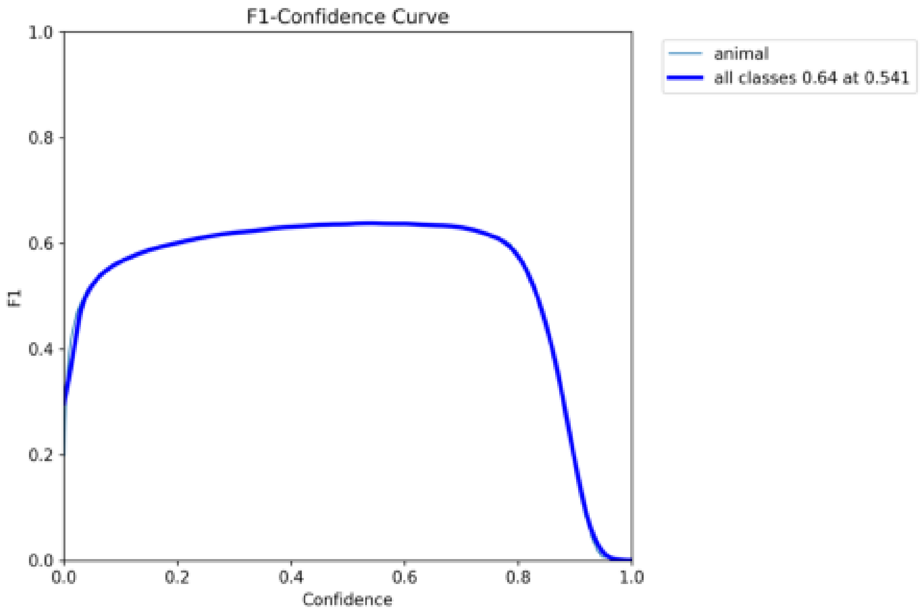
val_batch2_pred.jpg



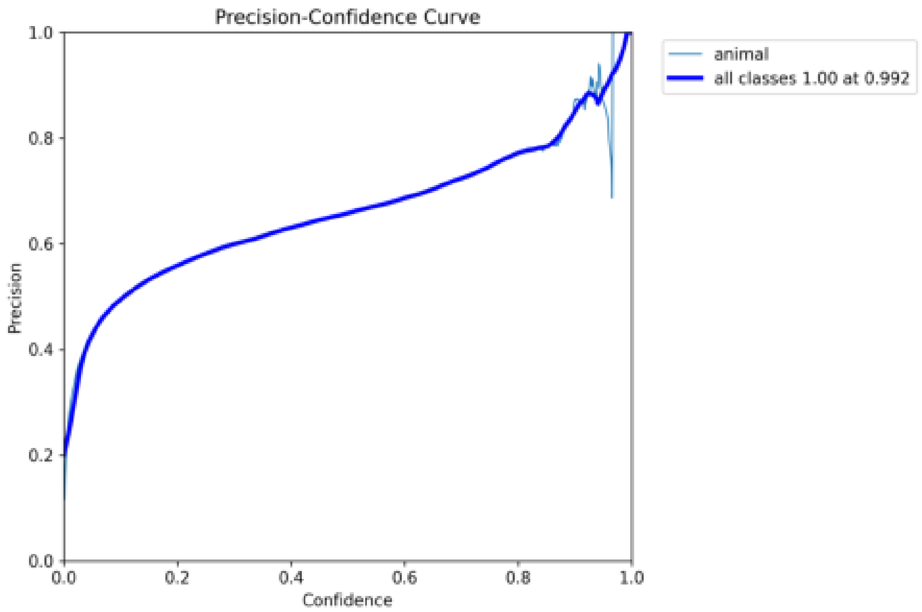
PosePR_curve.png



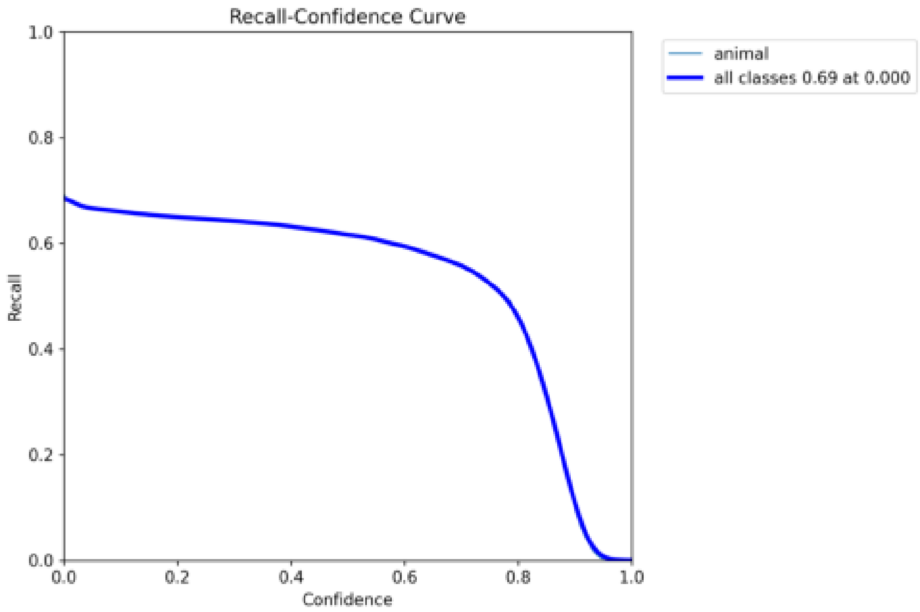
PoseF1_curve.png



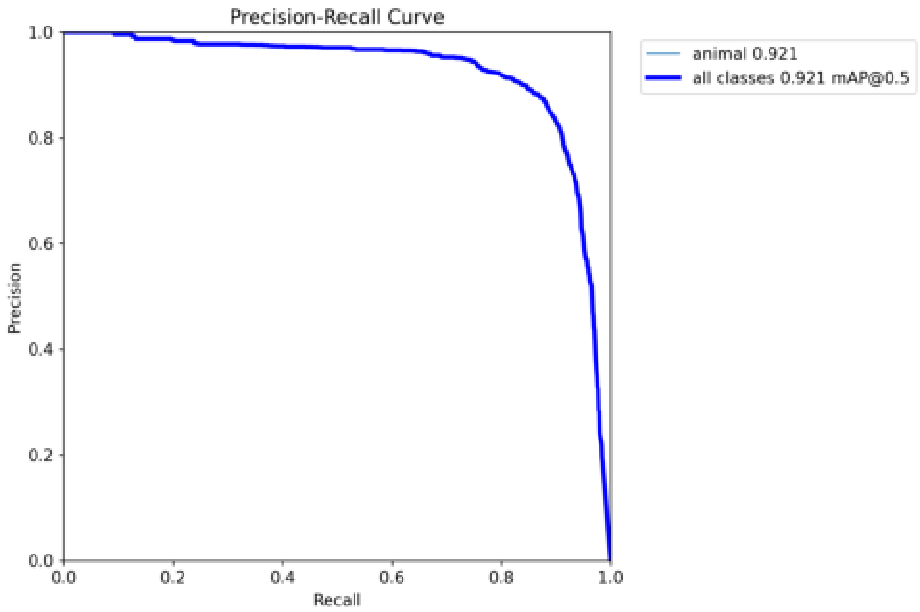
PoseP_curve.png



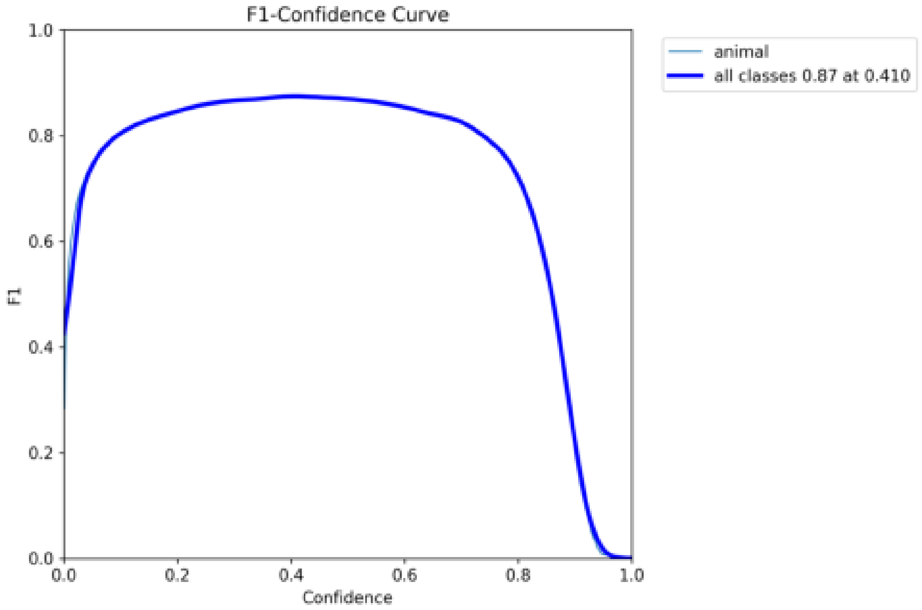
PoseR_curve.png



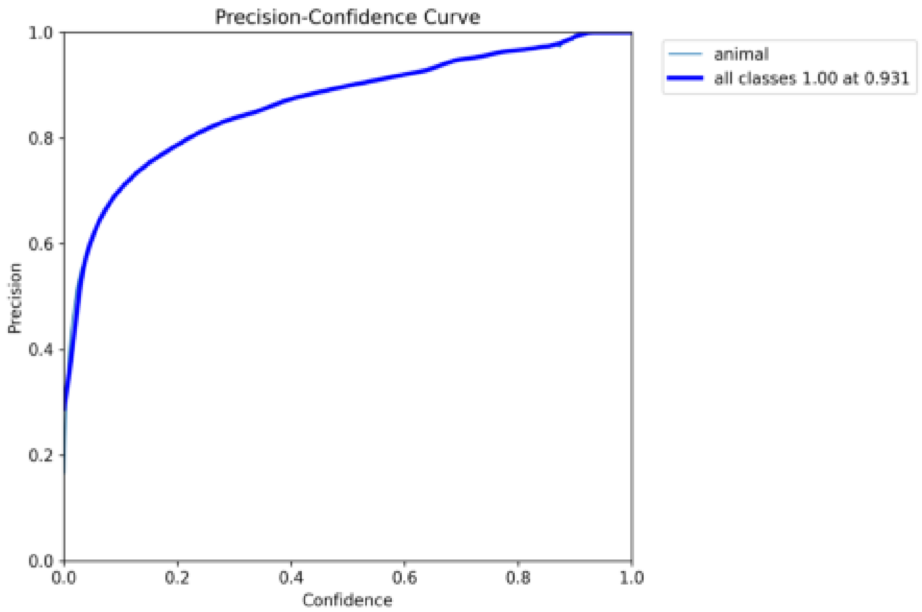
BoxPR_curve.png



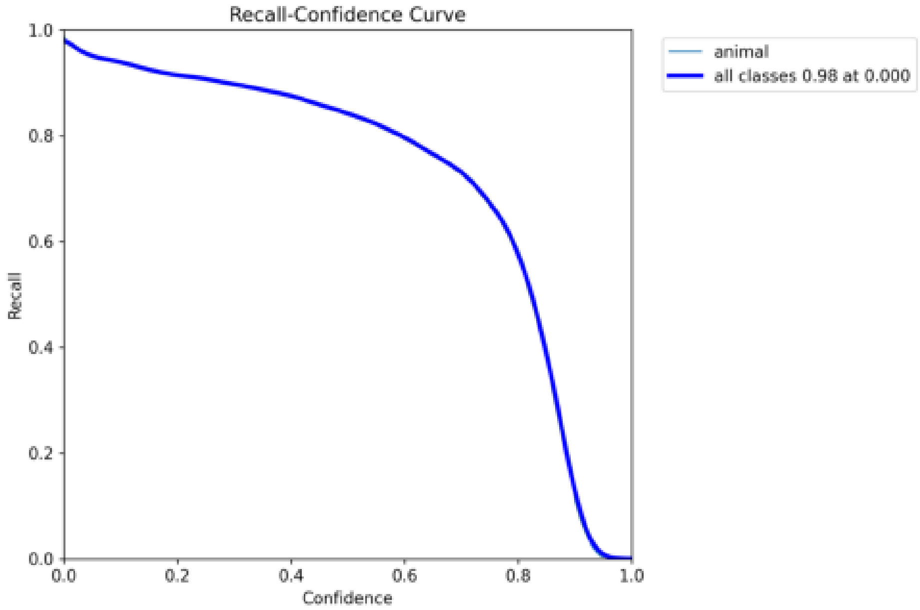
BoxF1_curve.png



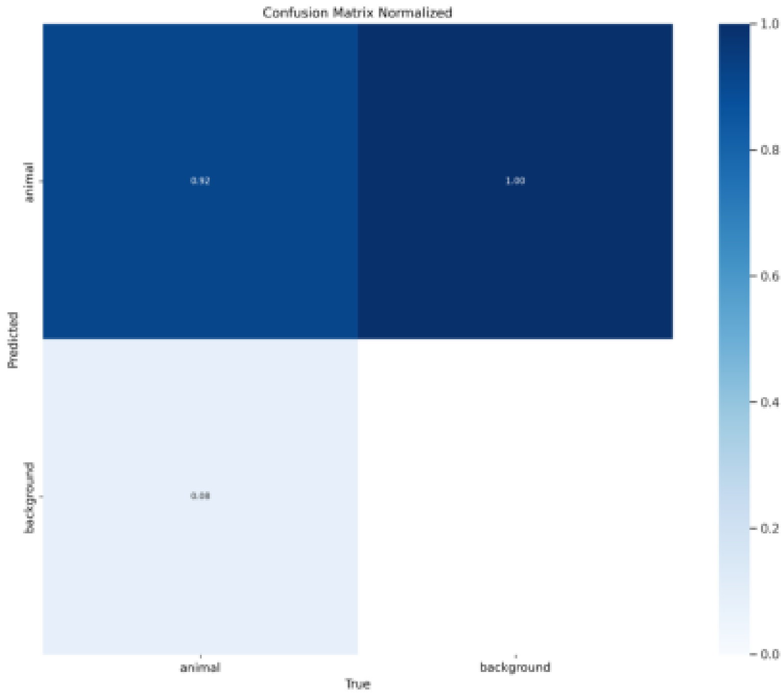
BoxP_curve.png

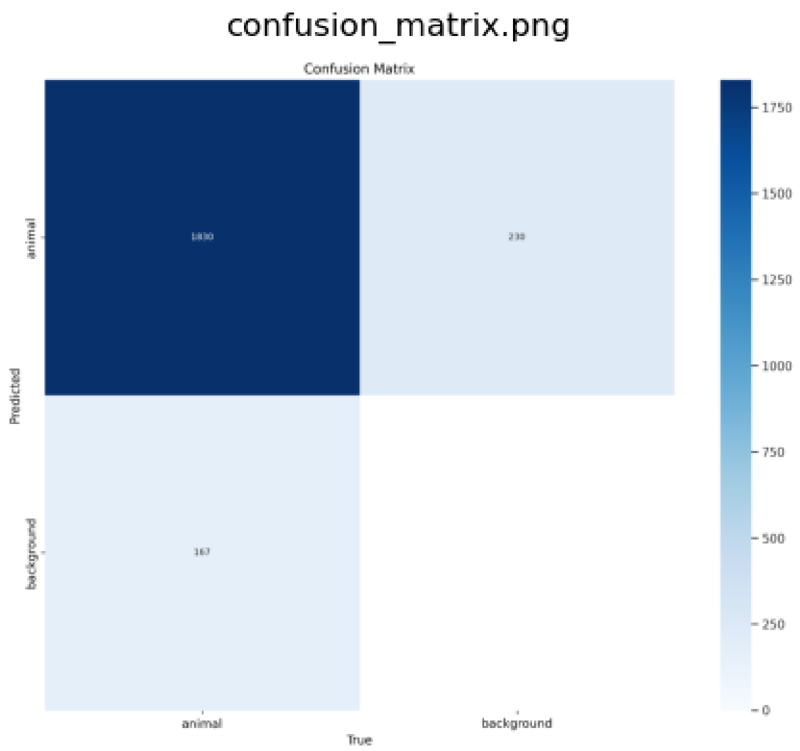


BoxR_curve.png



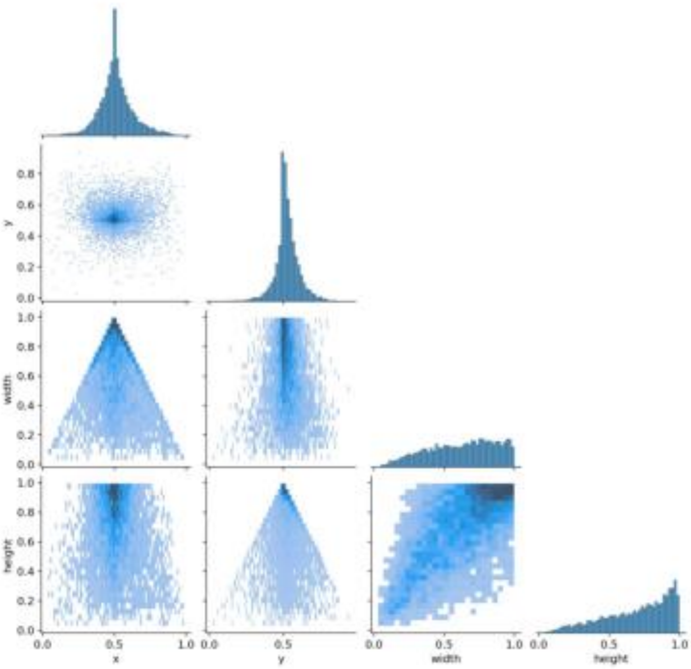
confusion_matrix_normalized.png



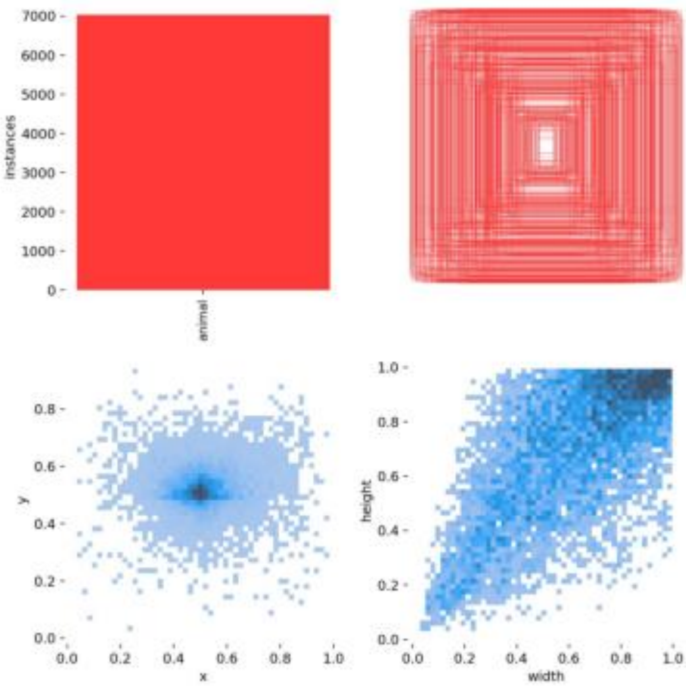



```python
import os
import matplotlib.pyplot as plt

def show_training_results(image_folder):
    # List all image files in the folder
    image_files = [file for file in os.listdir(image_folder) if
file.endswith(('.jpg', '.png'))]

    # Plot each image
    for image_file in image_files:
        image_path = os.path.join(image_folder, image_file)
        image = plt.imread(image_path)

        plt.imshow(image)
        plt.title(image_file)
        plt.axis('off')
        plt.show()

# Specify the folder containing the training result images
result_folder = 'Animal_Keypoints/yolov8n-pose_50_epochs'

# Show the training result images
show_training_results(result_folder)
```

labels_correlogram.jpg



labels.jpg

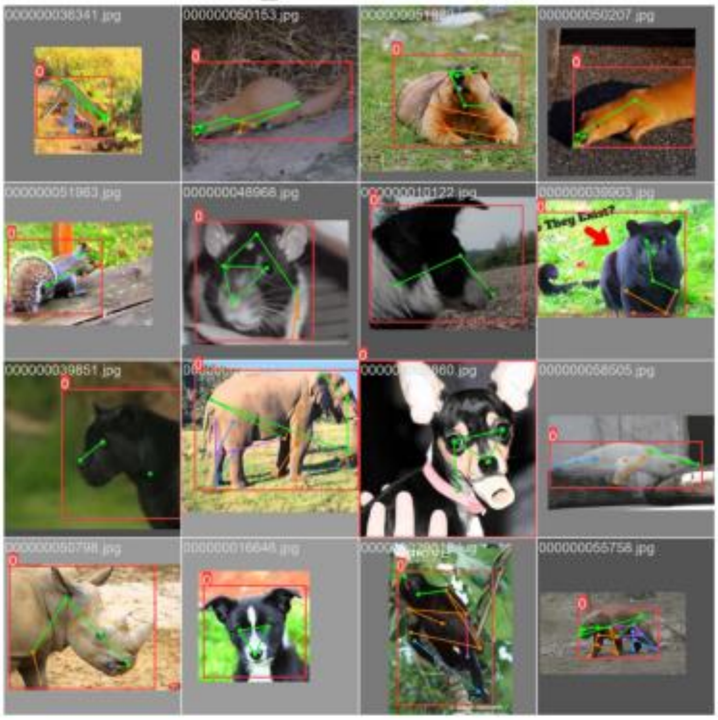


train_batch0.jpg

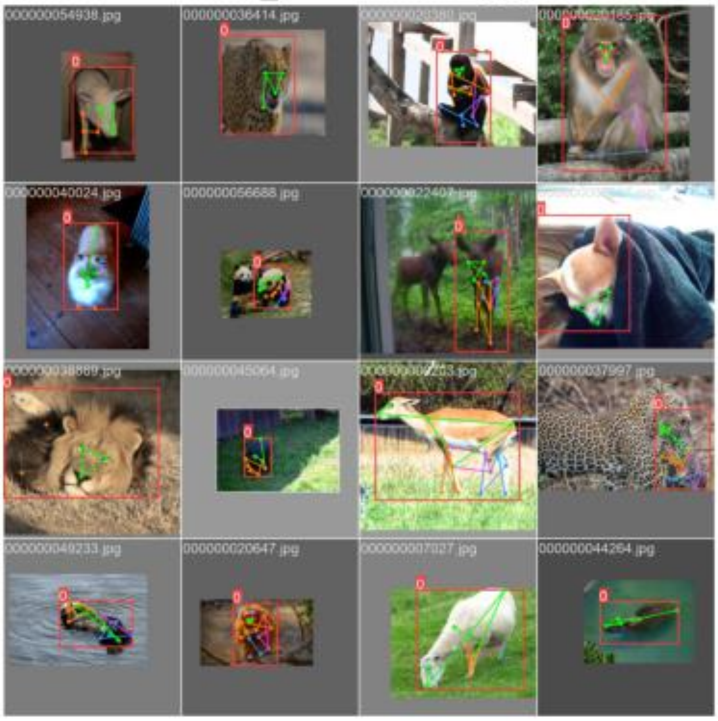
train_batch1.jpg

train_batch2.jpg

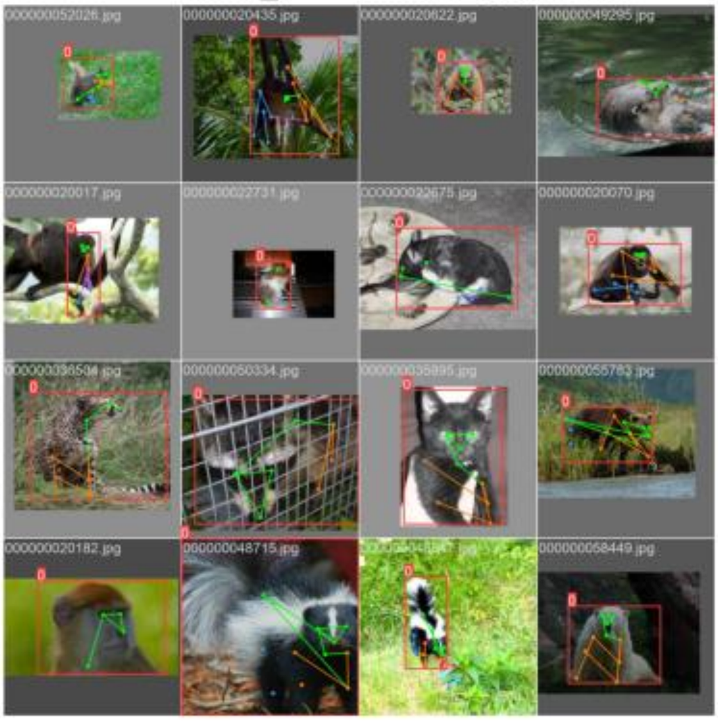
train_batch17560.jpg



train_batch17561.jpg



train_batch17562.jpg

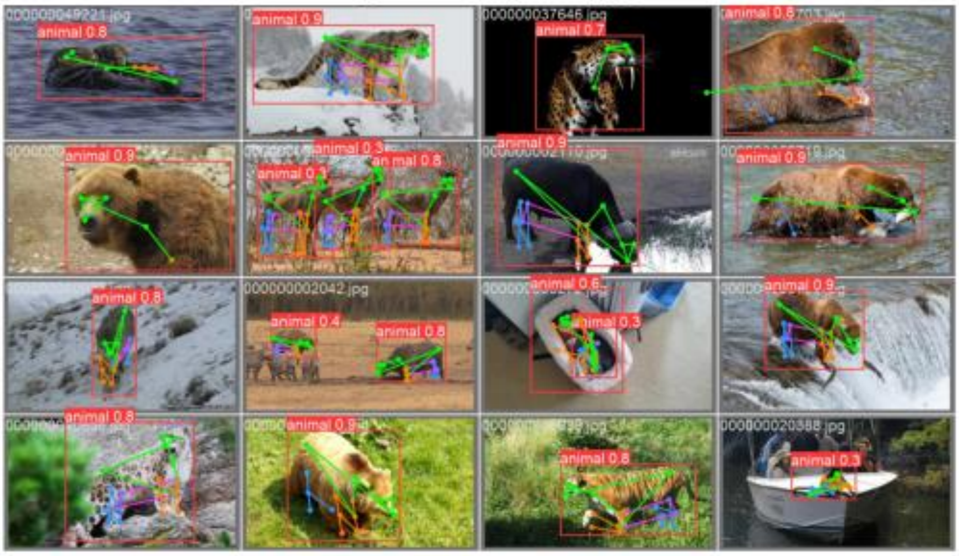


val_batch0_labels.jpg

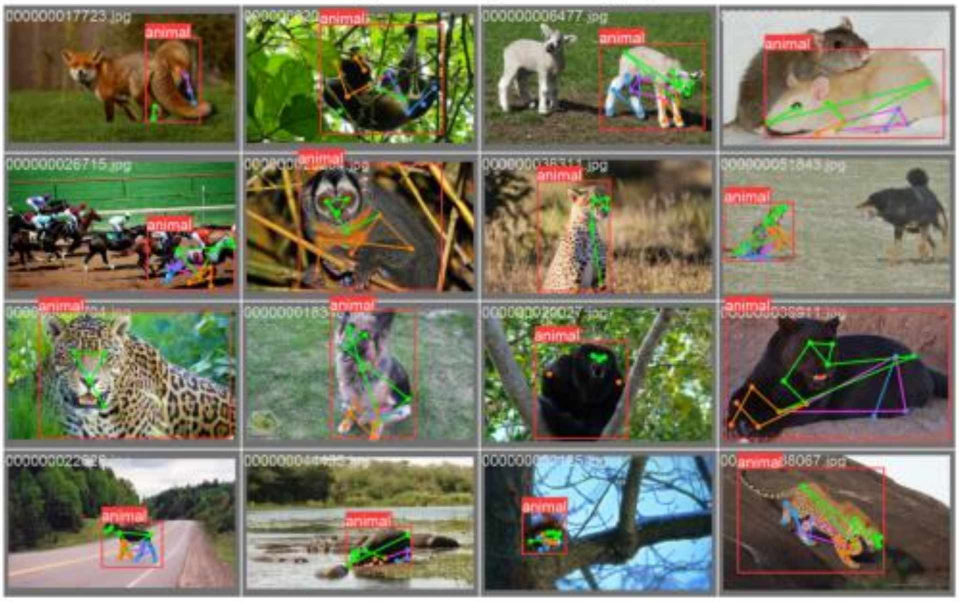
val_batch0_pred.jpg

val_batch1_labels.jpg

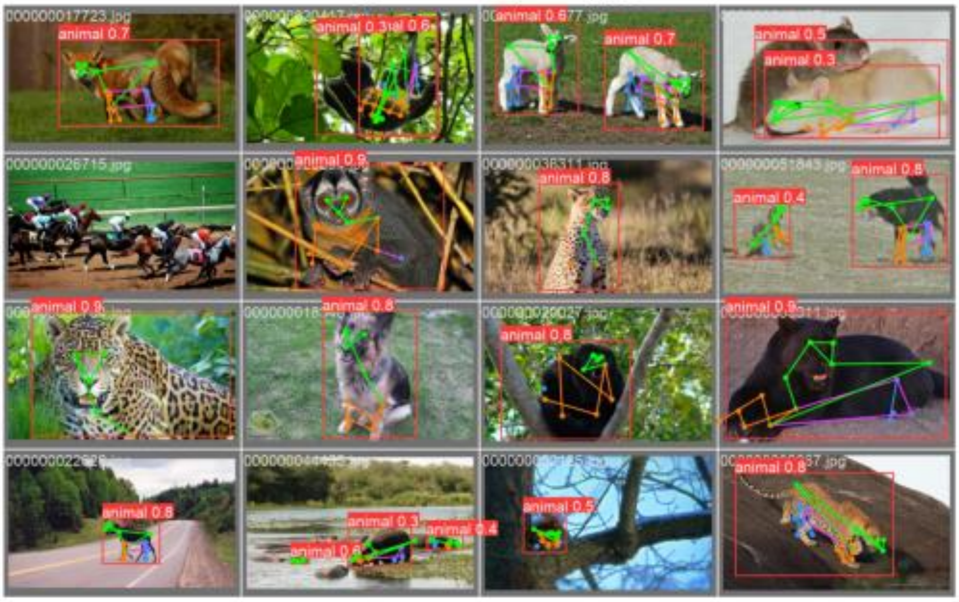
val_batch1_pred.jpg



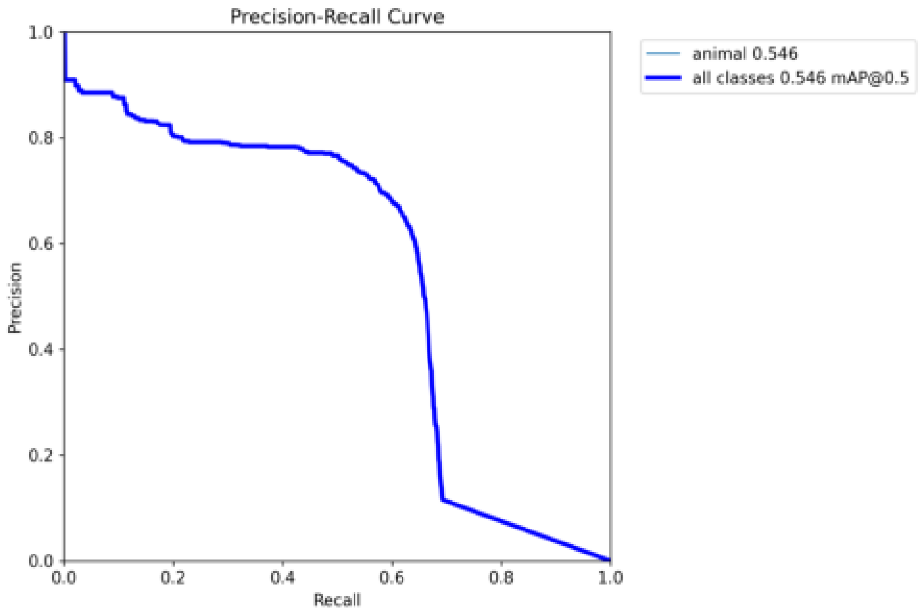
val_batch2_labels.jpg



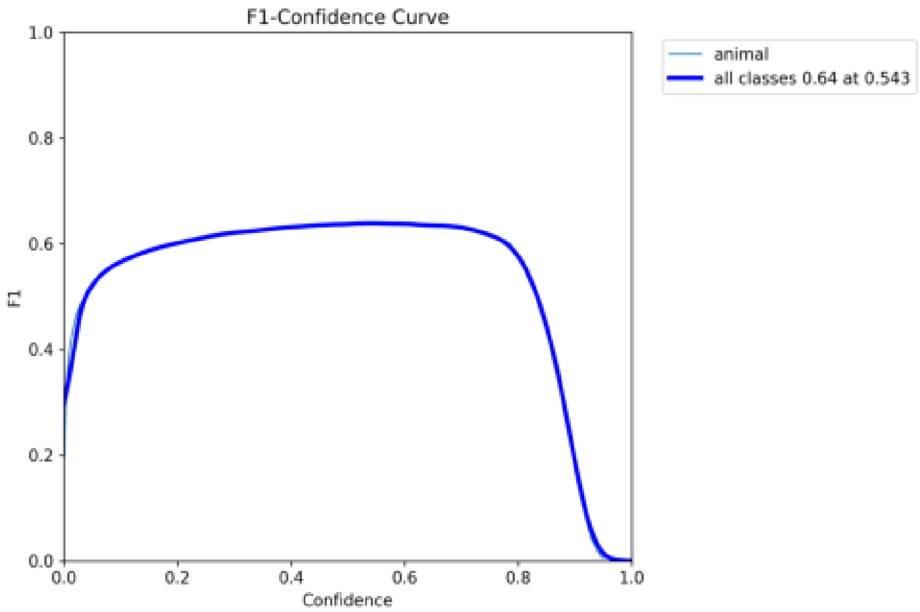
val_batch2_pred.jpg



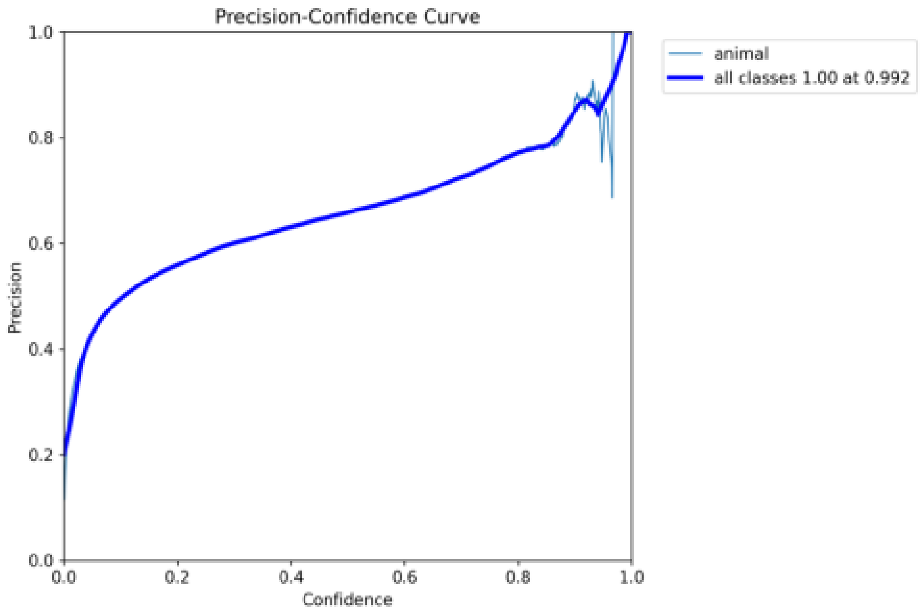
PosePR_curve.png



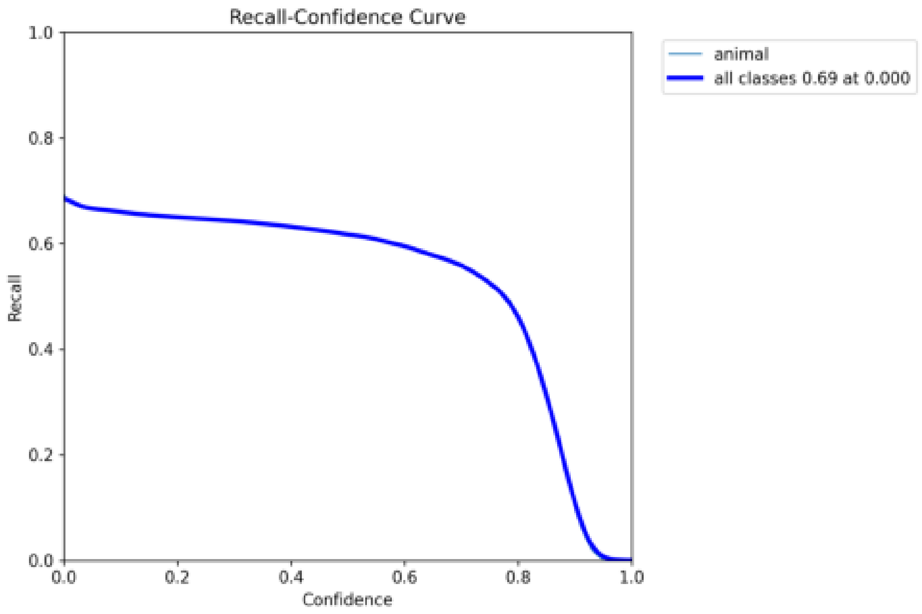
PoseF1_curve.png



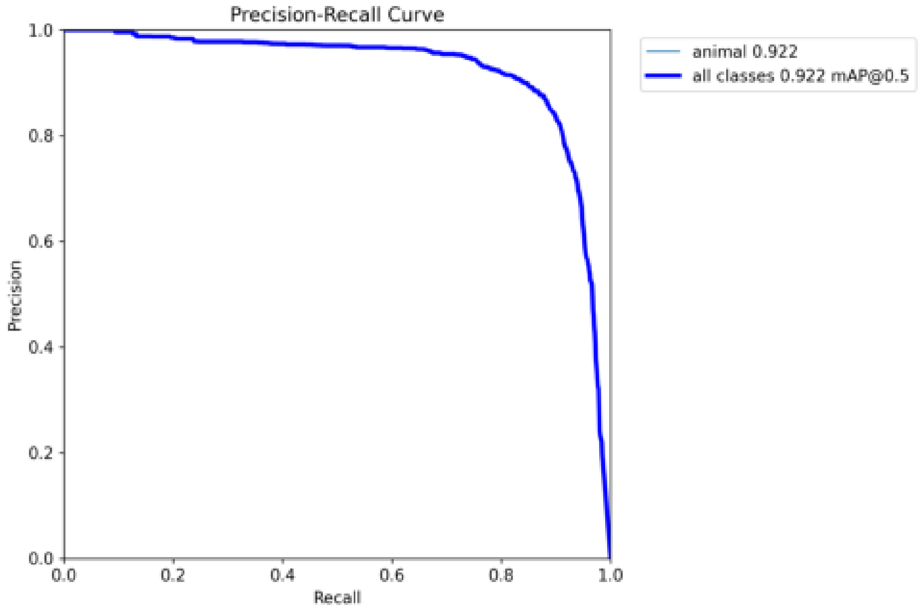
PoseP_curve.png



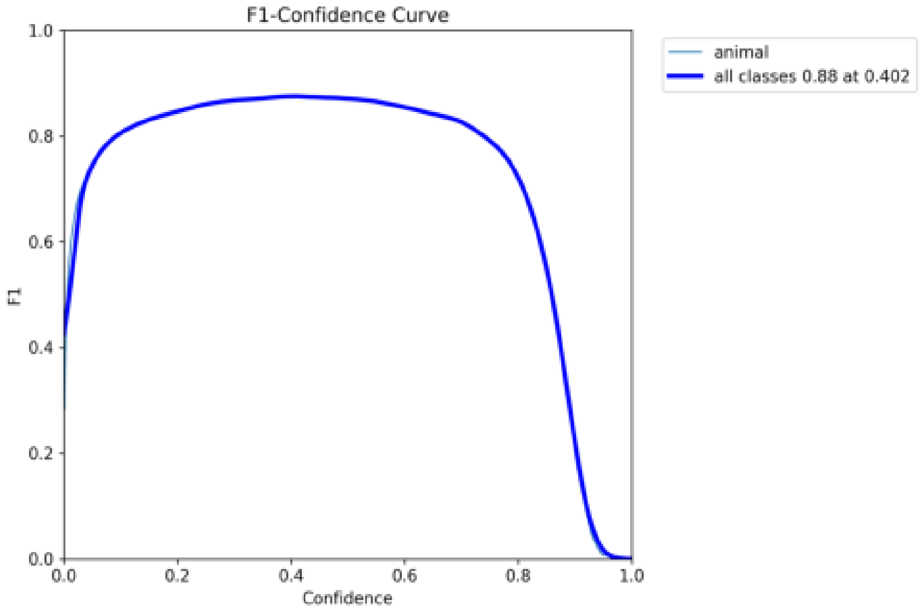
PoseR_curve.png



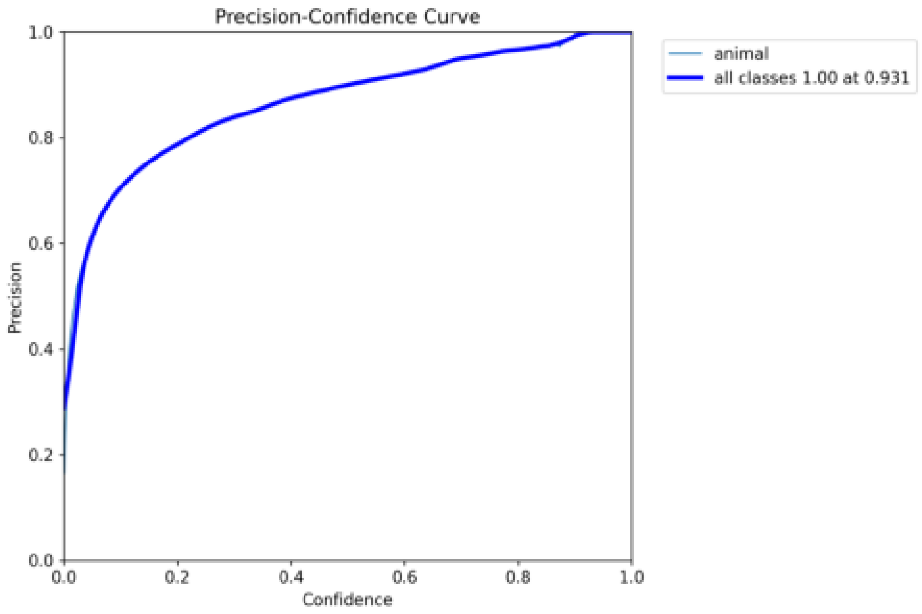
BoxPR_curve.png



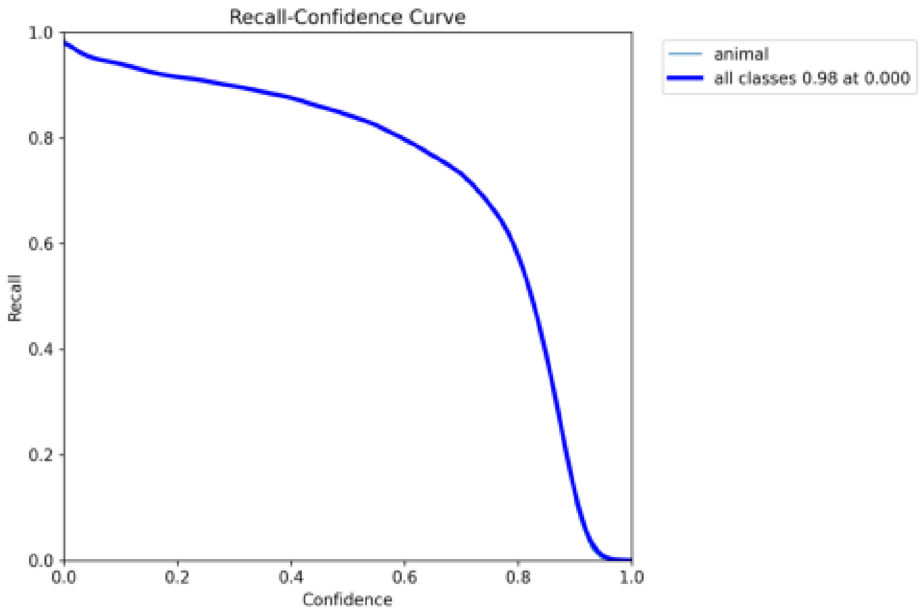
BoxF1_curve.png



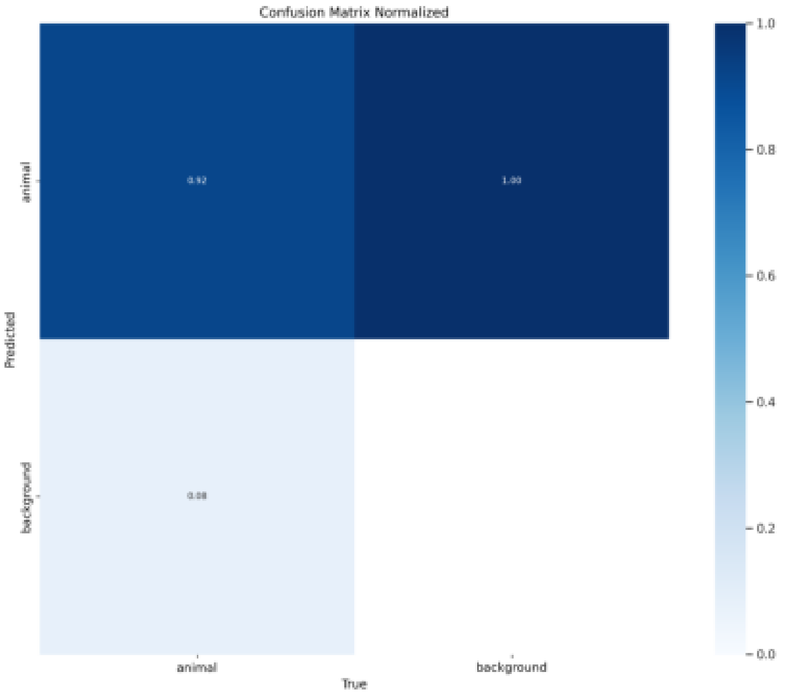
BoxP_curve.png



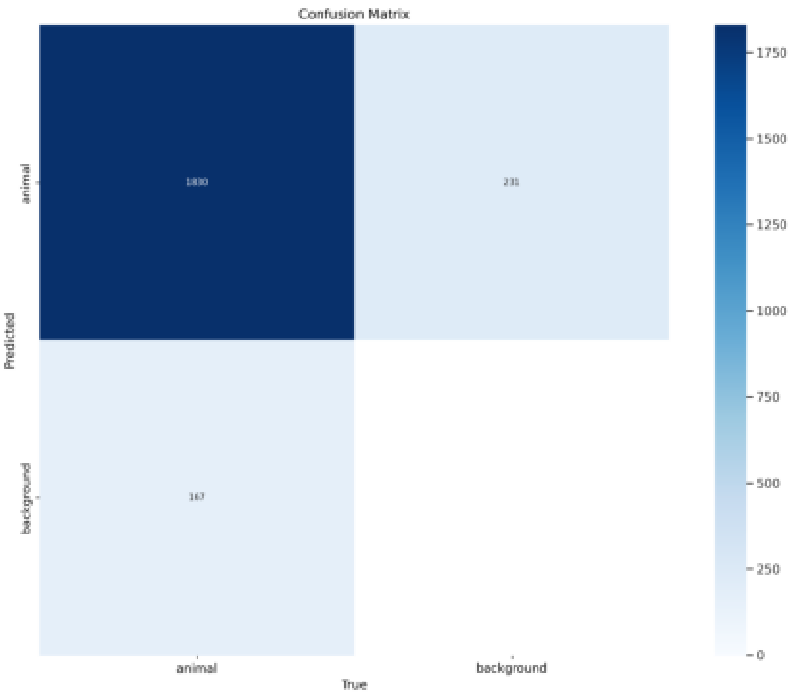
BoxR_curve.png



confusion_matrix_normalized.png



confusion_matrix.png



YOLOv8n-pose on AP-10K