**Animal Type Detection Project Documentation**

**Overview**

This document provides a step-by-step guide on how to run the code for the Animal Type Detection project. The code leverages YOLOv8 for detecting and classifying animals such as cows, goats, and pigs from images. It includes instructions for setting up the environment, specifying data paths, and executing the code.

To be specific the YOLOv8s.pt and the ”s” stands for "small." This model is designed to balance accuracy with inference speed, making it suitable for real-time applications or when working with limited computational resources.

We chose PyTorch for the training environment due to its optimal compatibility with the GPU(RTX 4080 SUPER) we utilized in the CUDA environment.

**Prerequisites**

Before running the code, ensure that you have the following installed:

* Python Version: The code requires Python 3.6 or later.
* Required Libraries: Install the required Python libraries by running:
* You can get all requirements in the requirements.txt file

**Data Setup**

The project uses a dataset stored in a GitHub repository. The data includes labeled images for training and testing.

**1. Downloading the Dataset**

The dataset is downloaded from a specific GitHub branch. The code creates a directory called animal\_type\_dataset where the data is stored.

**os.makedirs('animal\_type\_dataset', exist\_ok=True)**

**!git clone -b animal\_type\_detection\_dataset https://github.com/MVet-Platform/M-Vet\_Hackathon24.git ./animal\_type\_dataset**

Data Directory Structure: After downloading, te data is epeted to be in the following strcuture

animal\_type\_dataset/

├── images/

├── label\_train.csv

├── sample\_submission.csv

└── test\_images/

**2. Loading and Preparing Data**

The labels are loaded from the label\_train.csv file, and the paths to the image files are extracted and stored in the dataframe.

**df = pd.read\_csv('./animal\_type\_dataset/label\_train.csv')**

**df['filepath'] = df.apply(lambda row: glob(f'animal\_type\_dataset/\*\*/{row.filename}')[0], axis=1)**

**3. Train-Validation Split**

The data is split into training and validation sets using a 70-30 split.

**df\_train, df\_valid = train\_test\_split(df, test\_size=0.3, random\_state=32)**

**4. Creating YOLO Dataset Structure**

The YOLO dataset structure is created by organizing the images and labels into appropriate directories:

* **Training Data**:

**os.makedirs('yolo\_dataset/train/labels', exist\_ok=True)**

**os.makedirs('yolo\_dataset/train/images', exist\_ok=True)**

* **Validation Data**:

**os.makedirs('yolo\_dataset/valid/labels', exist\_ok=True)**

**os.makedirs('yolo\_dataset/valid/images', exist\_ok=True)**

The labels are converted from **VOC** format **(xmin, ymin, xmax, ymax)** to **YOLO** format **(center\_x, center\_y, width, height)**, and the images are copied to their respective directories.

**5. Configuring YOLO**

A YAML file is created to specify the paths to the training, validation, and test data, as well as the class names and the number of classes:

**data = {**

**'names': ['cow', 'goat', 'pig'],**

**'nc': 3,**

**'test': './yolo\_dataset/test/images',**

**'train': './yolo\_dataset/train/images',**

**'val': './yolo\_dataset/valid/images'**

**}**

**with open('yolo\_dataset/data.yaml', 'w') as file:**

**yaml.dump(data, file, default\_flow\_style=False)**

**Running the Code**

**1. Training the Model**

The YOLOv8 model is trained using the data specified in the YAML file. The training process includes setting the number of epochs and image size:

**model = YOLO("yolov8s.pt")**

**model.train(data="yolo\_dataset/data.yaml", epochs=210, imgsz=640, plots=True)**

**2. Result from the model**

**Ultralytics YOLOv8.0.145** is running on **Python 3.7.16** with **PyTorch 1.13.1** and **CUDA 11.7**, using an **NVIDIA GeForce RTX 4080 SUPER GPU**. The model has 168 layers and approximately 11.1 million parameters, with a computational complexity of **28.4 GFLOPs**. Performance metrics across all classes are high, with an overall **mAP50 of 0.945 and mAP50-95 of 0.74**. Specifically:

* Cow: mAP50 of 0.933, mAP50-95 of 0.712
* Goat: mAP50 of 0.966, mAP50-95 of 0.786
* Pig: mAP50 of 0.935, mAP50-95 of 0.721

The model processes images with an average speed of 0.1ms for preprocessing, 1.7ms for inference, 0.0ms for loss calculation, and 0.2ms for postprocessing. Results are saved in the runs/detect/train60 directory.

**As shown below:**

Ultralytics YOLOv8.0.145 🚀 Python-3.7.16 torch-1.13.1+cu117 CUDA:0 (NVIDIA GeForce RTX 4080 SUPER, 16064MiB)

**Model summary** (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████| 161/161 [00:15<00:00, 10.50it/s]

all 5136 5157 0.918 0.914 0.945 0.74

cow 5136 3260 0.92 0.906 0.933 0.712

goat 5136 1145 0.922 0.922 0.966 0.786

pig 5136 752 0.912 0.912 0.935 0.721

Speed: 0.1ms preprocess, 1.7ms inference, 0.0ms loss, 0.2ms postprocess per image

Results saved to **runs/detect/train60**

**3. Making Predictions**

After training, the model is used to make predictions on the test images which where over 9811 images . The predictions are saved in a specified directory:

**results = model.predict(source='yolo\_dataset/test/images', conf=0.25, save=True, project=f"runs/detect", name="predict", exist\_ok=True)**

**Sample predictions run:**

****

Above this image lies the confidence of the model’s prediction after thoroughly training it

On average the confidence is 0.87 which is 87% and the predictions are all saved in the runs/detect folder

**3. Creating Submission File**

The predictions are processed and saved in a submission file that includes the image name, class name, confidence, and bounding box coordinates:

**data = []**

**for result in results:**

**image\_name = os.path.basename(result.path)**

**class\_names =result.names**

**for box, cls, conf in zip(result.boxes.xyxy, result.boxes.cls, result.boxes.conf):**

**class\_name = class\_names[int(cls)]**

**bbox = box.cpu().tolist()**

**record = [image\_name, class\_name, conf.item()]+bbox**

**data.append(record)**

**with open(f'submission{int(datetime.now().timestamp())}.csv', mode='w', newline='') as file:**

**writer = csv.writer(file)**

**header = ['filename', 'class', 'confidence', 'xmin', 'ymin', 'xmax', 'ymax']**

**writer.writerow(header)**

**writer.writerows(data)**

**Directory Structure**:

The project directory should be organized as follows:

project\_root/

├── animal\_type\_dataset/

│ ├── images/

│ ├── label\_train.csv

│ ├── sample\_submission.csv

│ └── test\_images/

├── yolo\_dataset/

│ ├── train/

│ │ ├── images/

│ │ └── labels/

│ ├── valid/

│ │ ├── images/

│ │ └── labels/

│ ├── test/

│ │ └── images/

│ └── data.yaml

├── runs/

│ ├── detect/

│ │ ├── train/

│ │ ├── predict/

│ │ └── ...

├── your\_script.py

└── requirements.txt

**Explanation of Directories and Files:**

* **animal\_type\_dataset/**: Contains the original dataset files.
  + **images/**: Directory with images used for training.
  + **label\_train.csv**: CSV file containing labels for training images.
  + **sample\_submission.csv**: CSV file for sample submission format.
  + **test\_images/**: Directory with images used for testing.
* **yolo\_dataset/**: Contains the data prepared for YOLO training and validation.
  + **train/**: Contains images and labels for training.
    - **images/**: Training images.
    - **labels/**: YOLO formatted labels for training images.
  + **valid/**: Contains images and labels for validation.
    - **images/**: Validation images.
    - **labels/**: YOLO formatted labels for validation images.
  + **test/**: Contains test images.
    - **images/**: Test images.
  + **data.yaml**: Configuration file specifying paths and class names.
* **runs/**: Directory where YOLO saves the results.
  + **detect/**: Contains subdirectories for training runs and predictions.
    - **train/**: Training results and plots.
    - **predict/**: Prediction results saved by YOLO.
    - **...**: Other directories that may be created during experimentation.
* **your\_script.py**: Python script containing the code for the project.
* **requirements.txt**: File listing the Python libraries required for the project.