National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Himani Sharma
Student ID: X22224815

School of Computing
National College of Ireland

Supervisor: Mr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Himani Sharma
Student ID:	X22224815
Programme:	MSC-Data Analytics
Year:	2023-24
Module:	MSc Research Project
Supervisor:	Mr. Vladimir Milosavljevic
Submission Due Date:	12/8/2024
Project Title:	Configuration Manual
Word Count:	2400
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Himani Sharma
Date:	11 August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	Himani Sharma
Date:	August 11, 2024
Penalty Applied (if applicable):	

Configuration Manual

Himani Sharma
SID: X22224815

1 Overview

This manual explains the general setup and use of the various Python scripts for data preprocessing, training of Faster R-CNN, RT-DETR, and YOLO V8. Programs are written in the form of scripts so that they can be tuned by adjusting many factors that are common to all the scripts being used.

2 Prerequisites

2.1 Python Libraries

- `import json`
- `import os`
- `import random`
- `import xml.etree.ElementTree as ET`
- `from PIL import Image`
- `import yaml`
- `import shutil`
- `import cv2`
- `import numpy as np`
- `import albumentations as A`

2.2 Hardware

The following hardware is present by default on the current laptop, which is known as the Analytics Tier. These are therefore not prerequisites:

- Laptop/Desktop Computer: HP Pavilion Power Laptop 15-cb0xx series

- CPU/Processor: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50 GHz, Intel(R) Core(TM) i7-8750H CPU with a clock speed of 50 GHz, 2496 MHz, 4 cores, 4 threads.
- RAM: 16GB
- Graphics Card: NVIDIA GEFORCE GTX

2.3 Software

The following software is installed on the present laptop. These are not requirements but are helpful in making replication easier:

- Operating system: Windows 10 Home (Microsoft)
- Interactive Development Environment (IDE): PyCharm Community Edition 2019.2.3
- Anaconda Python 3 Distribution: Version 4.10.1

3 Dataset

https://www.kaggle.com/datasets/truthisneverlinear/dentex-challenge-2023

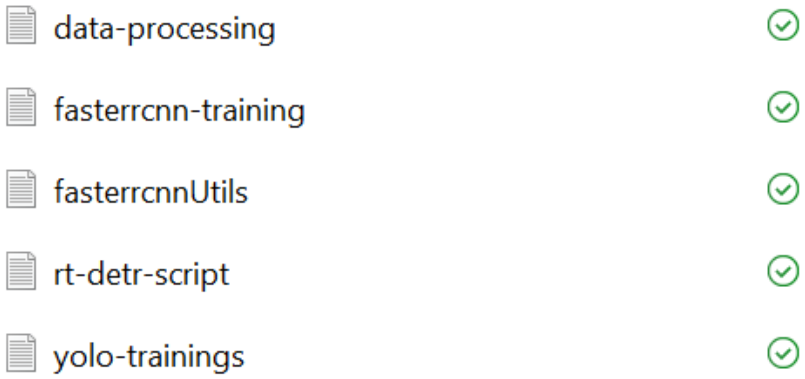

Figure 1: Scrpits

4 Data Processing Script

4.1 Data Input and Check

The **Data Input and Check** function loads the COCO dataset JSON file and opens it in write mode. It retrieves image IDs associated with `category_id_3` by looping through all annotations and adding the image IDs of the annotations related to this category. Next, it compares these image IDs to select those for which no annotations for `category_id_3` have been made. Finally, it displays the names of these images.

```python
ef check():
    # Load the COCO JSON file
    with open('train/quadrant-enumeration-disease/train_quadrant_enumeration_disease.json') as f:
        coco_data = json.load(f)

    # Create a set of image IDs that have annotations with categories_3 labels
    image_ids_with_categories_3 = set()
    for annotation in coco_data['annotations']:
        if 'category_id_3' in annotation:
            image_ids_with_categories_3.add(annotation['image_id'])

    # Create a set of all image IDs
    all_image_ids = {image['id'] for image in coco_data['images']}

    # Find image IDs that do not have any categories_3 annotations
    image_ids_without_categories_3 = all_image_ids - image_ids_with_categories_3

    # Get the file names of these images
    images_without_categories_3 = [image['file_name'] for image in coco_data['images'] if image['id'] in

    # Print the file names
```

Figure 2: Data Processing - Annotations

4.2 2. Remove Images Without Annotations

The **Remove Images Without Annotations(data_dir)** function eliminates images from directories where there is no corresponding label file. It sets paths to the images and labels for the training and validation data splits. This function compares file names to find images that are not labeled and removes those that lack descriptions.

4.3 Load COCO JSON

The **Load COCO JSON** function reads the COCO JSON file and transforms bounding boxes from COCO format to YOLO format with the help of a helper function. It then converts class IDs to the corresponding YOLO formatted annotation files and saves these YOLO annotation files to the labels directory. Additionally, it includes a `create_xml(filename, width, height, objects)` function that converts Pascal VOC XML files from the annotation data. This function creates XML elements regarding size and bounding boxes and returns an XML tree object.

4.4 Convert YOLO to Pascal VOC

The **Convert YOLO to Pascal VOC** tool changes the format of annotations from YOLO to Pascal VOC. It sets paths for YOLO and Pascal VOC directories, allowing the transformation of YOLO formatted labels into Pascal VOC XML format. This function creates Pascal VOC XML files and moves or replaces images into their corresponding directories.

4.5 Create New Custom YAML

The **Create New Custom YAML** function produces a YAML configuration file containing parameters related to the Pascal VOC dataset. It defines the paths for datasets and classes,

```
ef check():
    # Load the COCO JSON file
    with open('train/quadrant-enumeration-disease/train_quadrant_enumeration_disease.json') as f:
        coco_data = json.load(f)

    # Create a set of image IDs that have annotations with categories_3 labels
    image_ids_with_categories_3 = set()
    for annotation in coco_data['annotations']:
        if 'category_id_3' in annotation:
            image_ids_with_categories_3.add(annotation['image_id'])

    # Create a set of all image IDs
    all_image_ids = {image['id'] for image in coco_data['images']}

    # Find image IDs that do not have any categories_3 annotations
    image_ids_without_categories_3 = all_image_ids - image_ids_with_categories_3

    # Get the file names of these images
    images_without_categories_3 = [image['file_name'] for image in coco_data['images'] if image['id'] in

    # Print the file names
```

Figure 3: Data Processing-Input

and saves the YAML configuration to the specified output directory.

4.6 Augment Images

The **Augment Images** function applies various transformations to enhance images based on input and predefined augmentation techniques. It uses methods such as flipping, brightness adjustment, and noise addition. The function applies these augmentations to each image and saves the augmented images in a new folder.

4.7 Main Function

The **Main()** function initiates the entire process, acting as the workflow management function. It enumerates untagged images belonging to categories such as animate, plant, artifact, animal, leaf, and tree. It calls necessary functions for removing images without annotations, sampling and copying images, converting annotations, and creating the YAML configuration. Finally, it enhances images and stores them in the output folder, marking the end of the process.

5 Faster RCNN.py File

5.1 Imports

The imports include the `FastRCNNPredictor` class from `torchvision`, which provides helper functions for the prediction head of Faster R-CNN models. Various utilities are included, which encompass functions related to training, evaluation, datasets, logs, and model inputs/outputs. Additionally, `distributed`, `RandomSampler`, and `SequentialSampler` are two major classes used for distributed and non-distributed data sampling.

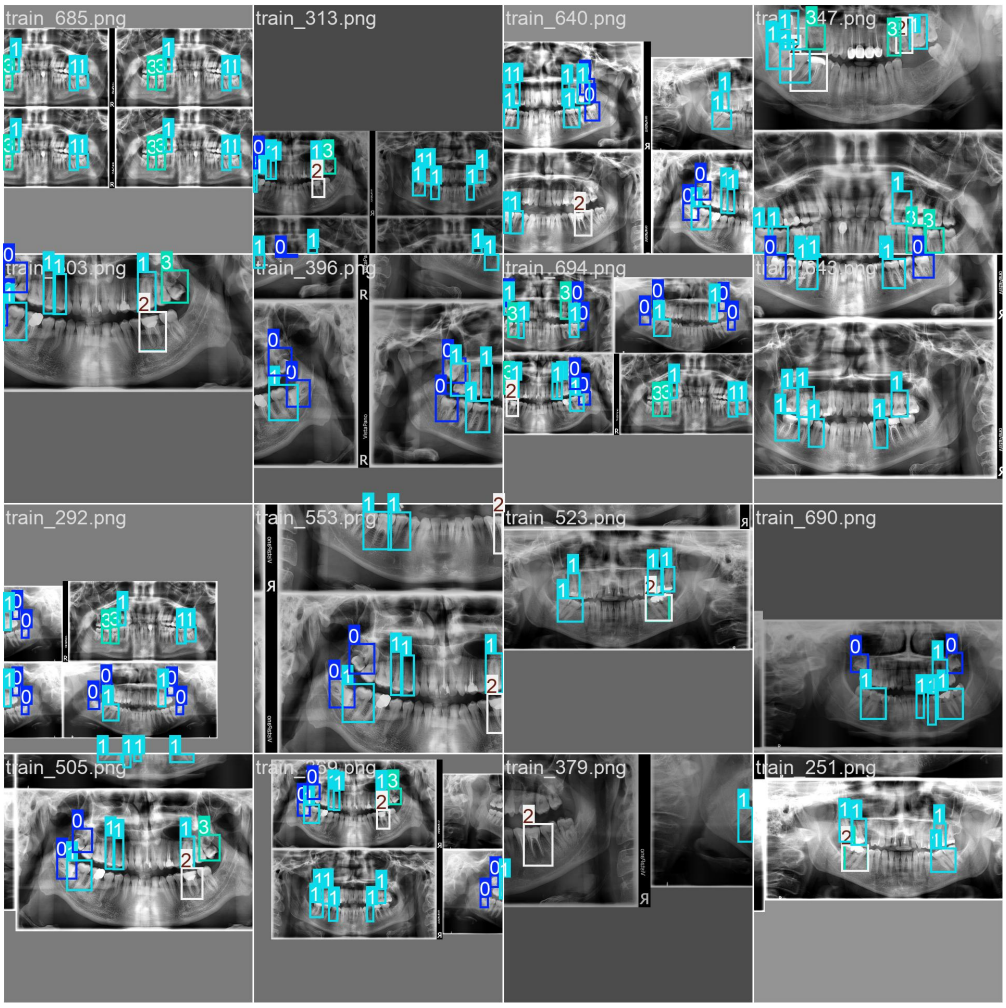

Figure 4: Train Data

```
rt yaml
rt numpy as np
rt torchinfo
rt os
rt torchvision

torchvision.models.detection.faster_rcnn import FastRCNNPredictor

fasterrcnnUtils import (
train_one_epoch, evaluate, utils, create_train_dataset, create_valid_dataset,
create_train_loader, create_valid_loader, create_model, set_training_dir, Averager,
save_model, save_loss_plot,
show_tranformed_image,
save_mAP, save_model_state, SaveBestModel,
yaml_save, init_seeds, set_log, coco_log,
set_summary_writer,
tensorboard_loss_log,
tensorboard_map_log,
csv_log,
wandb_log,
wandb_save_model,
```

Figure 5: Faster RCNN

```
import json
import os
import random
import shutil
import xml.etree.ElementTree as ET
from PIL import Image
import yaml
import shutil
import cv2
import numpy as np
import albumentations as A
from albumentations.augmentations.transforms import (
    HorizontalFlip,
    RandomBrightnessContrast,
    ShiftScaleRotate,
    RandomCrop,
    GaussNoise
)
```

Figure 6: Faster RCNN

5.2 Model Creation

The `create_model` function has the following arguments: `num_classes`, and `coco_model=False`. It constructs a Faster R-CNN model based on a pretrained MobileNetV3 backbone. The function modifies the final part of the structure, specifically the `roi_heads.box_predictor`, to match the required number of classes. Additionally, it includes an option `coco_model`, which, if set to `True`, returns a model pretrained on the COCO dataset.

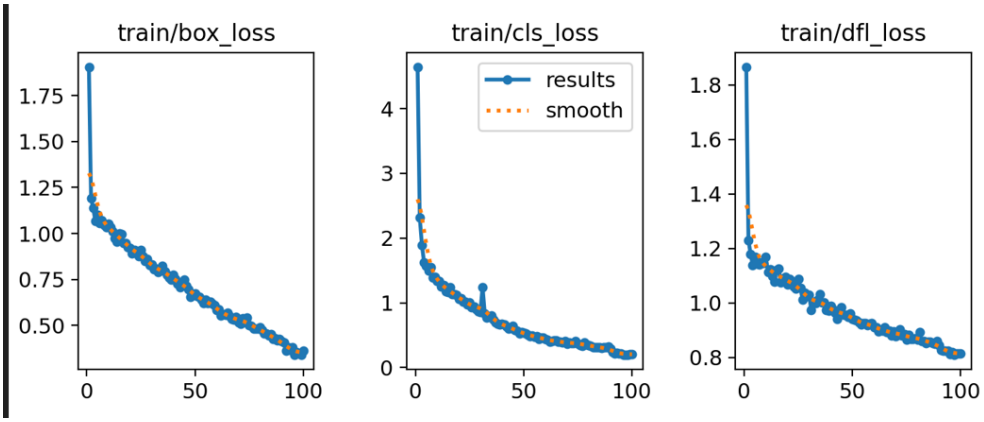

Figure 7: Trainning graph

Main Function (`main`)

The `main` function initializes distributed training mode if applicable and sets up Weights & Biases (W&B) logging if not disabled. It loads data configurations from a YAML file. To

ensure reproducibility, the function sets random seeds at the start of the run. It also ensures that both the training and validation datasets are fully loaded. Depending on whether distributed training is employed or not, data loaders are created with appropriate sampling strategies.

5.3 Visualization

This section optionally displays the converted images.The model can be constructed from scratch or, if weights are provided, continued from where it left off. The classification head is set to match the number of classes specified in the input. The function also defines data distribution, data parallelism, and synchronization.

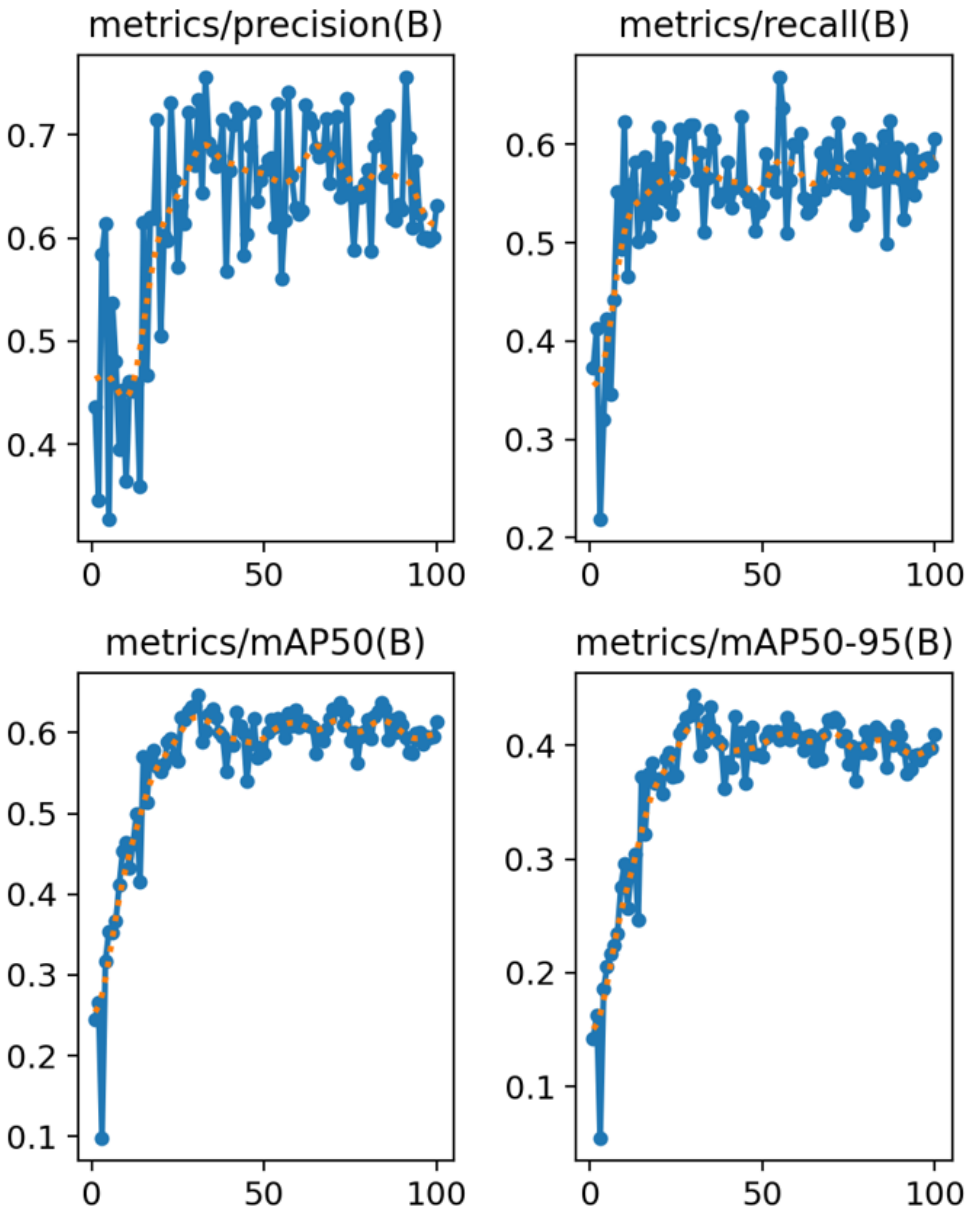

Figure 8: Visualization-output

5.4 Training Loop

The training loop works through a specified number of iterations, known as epochs. It saves the training and validation losses and mAP (mean Average Precision). Performance is tracked using TensorBoard, W&B, or CSV. All model checkpoints are saved, and the best model is selected based on validation performance.

5.5 Key Utility Functions

The function trains the model for one epoch, computes the required losses, and handles gradient updates.Runs the model on the validation set to obtain performance metrics. Backs up the state of the model and its parameters.Provides methods for creating plots of training losses and mAP, and includes functions for saving these plots. Includes additional logging functions for Weights & Biases and TensorBoard.

5.6 Logging

Handles plotting of training loss and mAP. Training logs can be saved.Used for logging and graphic analysis.Utilizes SGD with an optional cosine annealing learning rate scheduler. Optionally performs automatic mixed precision (AMP) to accelerate training. Supports saving both single-machine and multi-machine distributively trained models.

```
def check():
    # Load the COCO JSON file
    with open('train/quadrant-enumeration-disease/train_quadrant_enumeration_disease.json') as f:
        coco_data = json.load(f)

    # Create a set of image IDs that have annotations with categories_3 labels
    image_ids_with_categories_3 = set()
    for annotation in coco_data['annotations']:
        if 'category_id_3' in annotation:
            image_ids_with_categories_3.add(annotation['image_id'])

    # Create a set of all image IDs
    all_image_ids = {image['id'] for image in coco_data['images']}

    # Find image IDs that do not have any categories_3 annotations
    image_ids_without_categories_3 = all_image_ids - image_ids_with_categories_3

    # Get the file names of these images
    images_without_categories_3 = [image['file_name'] for image in coco_data['images'] if image['id'] in image_ids_without_categ

    # Print the file names
```

Figure 9: Logging

6 YOLOV8

6.1 Define `train_v8s` Function

The `train_v8s` function is responsible for training and validating the YOLOv8 small model. It calls the constructor of the YOLOv8 small model with pre-trained weights from a file. This function trains the model using a JSON file containing parameters such as the training data size, number of epochs, image size, batch size, and patience for early stopping. After training, it assesses the model on a validation dataset and displays the mean Average Precision (mAP) scores at various Intersection over Union (IoU) values to show model performance.

```
in_v8s():
oad the model
el = YOLO('yolov8s.pt')

rain the model
lts = model.train(
 data='custom.yaml',
 epochs=100,
 imgsz=1280,
 patience=50,
 batch=16,
 save=True


alidate the model
_results = model.val()

nt(f"mAP50-95: {val_results.box.map}")
nt(f"mAP50: {val_results.box.map50}")
nt(f"mAP75: {val_results.box.map75}")

in_v8m():
oad the model
el = YOLO('yolov8m.pt')
```

Figure 10: YOLOV8

6.2 Define `train_v8m` Function

Similar to `train_v8s`, the `train_v8m` function trains and validates the YOLOv8 medium model. It sets the model to YOLOv8 medium and loads the weights from a file. This function uses the same training parameters as those in the `train_v8s` function. It measures the effectiveness of the developed model and provides the evaluation metrics.

6.3 Define `train_v8l` Function

The `train_v8l` function handles the training and validation for the YOLOv8 large model. It initializes the YOLOv8 large model with pre-trained weights. This function uses similar training parameters as the other training functions mentioned. It evaluates the model's performance on a validation dataset and prints out the evaluation metrics for the large model.

6.4 Define `main` Function

The `main` function orchestrates the training process for all three YOLOv8 models. It prints messages to notify the start of training for each model and specifies the path where the results will be stored. The function sequentially uses `train_v8s` to train the small model, `train_v8m` for the medium-sized model, and `train_v8l` for the large model. After each model's training is complete, the function prints a message informing users of the location where the results have been written.

7 RT-DETR

```python
# Load the model
model = RTDETR('rtdetr-l.pt')

# Train the model
results = model.train(
    data='custom.yaml',
    epochs=100,
    imgsz=1280,
    patience=50,
    batch=16,
    save=True
)

# Validate the model
val_results = model.val()

print(f"mAP50-95: {val_results.box.map}")
print(f"mAP50: {val_results.box.map50}")
print(f"mAP75: {val_results.box.map75}")

print("Results saved in re-detr_training")
```

Figure 11: RT-DETR

7.1 Load the Model

An example of the RT-DETR model is established by loading a pre-trained model file named `rtdetr-l.pt`. This file contains the weights for the RT-DETR large model, which is a variant designed for object detection tasks. The `train` method is then used on the model object to perform training.

7.2 Training Configuration

The dataset configuration is specified via a YAML file, referred to as `custom.yaml`. This file includes paths to the training and validation images, as well as class labels. The number of training epochs is set to 100, where an epoch represents a complete pass through the entire training dataset.

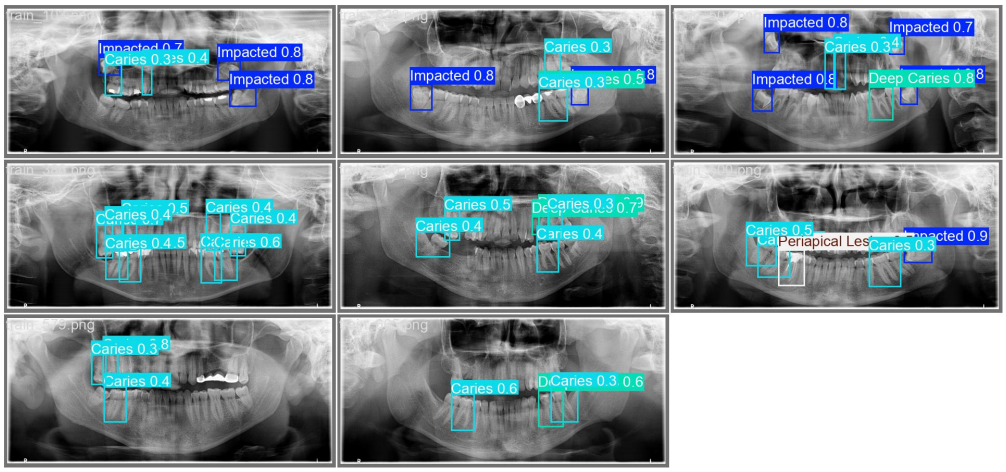

Figure 12: Training Data

The input image size for training is specified as 1280 by 1280 pixels. The parameter `time_to_stop=50` with `patience=50` indicates that the training will terminate if there is no improvement in validation performance for 50 consecutive epochs, in order to prevent overfitting. The batch size is set to 16, defining the number of samples processed before updating the model parameters. Additionally, it is suggested that model checkpoints are saved during training and that the final model is saved after training is complete.

7.3 Validate the Model

To assess the performance of the model, the `val` method is used. This method evaluates the model on the validation dataset, which consists of data that the model has not seen during training. The evaluation provides metrics that reflect the model's efficiency and effectiveness in making predictions.

8 References

Gad, A.F. (2024) 'PyGAD: an intuitive genetic algorithm Python library', *Multimedia Tools and Applications*, 83, pp. 58029–58042. Available at: https://doi.org/10.1007/s11042-023-17167-y (Accessed: 11 August 2024).

Sirin, U. and Idreos, S. (2024) 'The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format', *Proceedings of the ACM on Management of Data*, 2(1), Article 52, pp. 1–31. Available at: https://doi.org/10.1145/3639307 (Accessed: 11 August 2024).

Creswell, J., Vo, L.N.Q., Qin, Z.Z., Muyoyeta, M., Tovar, M., Wong, E.B., Ahmed, S., Vijayan, S., John, S., Maniar, R., Rahman, T., MacPherson, P., Banu, S. and Codlin, A.J. (2023) 'Early user perspectives on using computer-aided detection software for interpreting chest X-ray images to enhance access and quality of care for persons with tuberculosis', *BMC Global and Public Health*, 1(30). Available at: https://doi.org/10.1186/

s44263-023-00033-2 (Accessed: 11 August 2024).

Si, T., He, F., Li, P. and Gao, X. (2022) 'Tri-modality consistency optimization with heterogeneous augmented images for visible-infrared person re-identification', *Neurocomputing*, 522, pp. 12–24. Available at: https://doi.org/10.1016/j.neucom.2022.12.042 (Accessed: 11 August 2024).