National College of Ireland

Configuration Manual

MSc Research Project
MSc in Artificial Intelligence for Business

Sanjay Rastogi
Student ID: x23160977

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sanjay Rastogi
Student ID:	x23160977
Programme:	MSc in Artificial Intelligence for Business
Year:	2024
Module:	MSc Research Project
Supervisor:	Anderson Simiscuka
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	2288
Page Count:	20

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:	
Date:	9th 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:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sanjay Rastogi
x23160977

1. System Requirements

RAM: 32GB

OS: Windows

Processor: Core i9, 13th Generation

GPU: 12 GB Nvidia, 3060 RTX

Platform: Online Google Colab Pro Plus

1.1 Yolov8

```
from google.colab import files
files.upload()
```

Figure 1: Google Colab staging

This code lets you upload files to Google Colab:

- Import: `files` module from `google.colab`.

- Upload: `files.upload()` opens a file picker for uploading files.

```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Figure 2: Kaggle API set up

This code sets up Kaggle API credentials:

1. `!mkdir -p ~/.kaggle`: Creates a `.kaggle` directory.

2. `!mv kaggle.json ~/.kaggle/`: Moves `kaggle.json` to `.kaggle`.

3. `!chmod 600 ~/.kaggle/kaggle.json`: Restricts file access for security.

```
!kaggle datasets download -d shreydan/kitti-dataset-yolo-format -p /tmp
```

Figure 3: Kitti Datasets download

This code downloads a dataset from Kaggle:

- `!kaggle datasets download -d shreydan/kitti-dataset-yolo-format`: Downloads the "kitti-dataset-yolo-format" dataset.

- `-p /tmp`: Saves the downloaded dataset to the `/tmp` directory.

```
!kaggle datasets download -d klemenko/kitti-dataset -p /tmp

!unzip /tmp/weather-data.zip -d /tmp
```

Figure 4: Staging for some extra data

This code downloads and extracts a dataset from Kaggle:

1. `!kaggle datasets download -d klemenko/kitti-dataset -p /tmp`: Downloads the "kitti-dataset" to the `/tmp` directory.

2. `!unzip /tmp/weather-data.zip -d /tmp`: Extracts `weather-data.zip` contents into the `/tmp` directory. (Note: Ensure the file name matches the downloaded file if you intended to unzip the downloaded dataset.)

```
!kaggle datasets download -d klemenko/kitti-dataset
```

Figure 5: Code for the dataset download

This command downloads a dataset from Kaggle:

- `!kaggle datasets download -d klemenko/kitti-dataset`: Downloads the "kitti-dataset" from Kaggle to your current working directory.

```
import zipfile
import os

# Unzip the file
zip_file_path = '/content/kitti-dataset.zip'
extract_to_path = '/content/dataset/'

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_path)

# List files in the extracted directory to verify
os.listdir(extract_to_path)
```

Figure 6: List of the datasets downloaded

This code extracts `kitti-dataset.zip` into `/content/dataset/` and then lists the files in that directory to verify the extraction.

```python
import zipfile
import os

# Unzip the file
zip_file_path = '/tmp/kitti-dataset-yolo-format.zip'
extract_to_path = '/content/dataset_yolo/'

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_path)

# List files in the extracted directory to verify
os.listdir(extract_to_path)
```

Figure 7: Datasets extraction and listing the contents in the path

This code extracts `kitti-dataset-yolo-format.zip` to `/content/dataset_yolo/` and then lists the files in that directory to confirm the extraction.

```
!pip install ultralytics
```

Figure 8: YOLO models package for version 8

This command installs the `ultralytics` library, which is used for advanced computer vision tasks, such as object detection and segmentation.

```
%env WANDB_DISABLED=True
```

Figure 9: The command disables Weights & Biases (WandB) integration by setting the `WANDB_DISABLED`

This command disables Weights & Biases (WandB) integration by setting the `WANDB_DISABLED` environment variable to `True`. This is useful to prevent automatic logging and tracking if you don't want to use WandB for experiments.

```python
from ultralytics import YOLO
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import json
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm
import shutil
from PIL import Image
```

Figure 10: Code imports various libraries and modules for working with the YOLO object detection model and handling data

- `ultralytics.YOLO`: Imports the YOLO model for object detection.

- `numpy`, `pandas`, `matplotlib.pyplot`: For numerical operations, data manipulation, and plotting.

- `pathlib.Path`: For handling filesystem paths.

- `json`: For parsing JSON data.

- `sklearn.model_selection.train_test_split`: For splitting data into training and test sets.

- `tqdm.auto`: For displaying progress bars.

- `shutil`: For file operations.

- `PIL.Image`: For image processing.

```
base_dir = Path('content/dataset/')
img_path = Path('/content/dataset/data_object_image_2/training/image_2')
label_path = Path('/content/dataset_yolo/labels')
with open('/content/dataset_yolo/classes.json','r') as f:
    classes = json.load(f)

classes
```

Figure 11: This code sets up paths for image and label directories, then loads class labels from a JSON file

- Paths Setup: Defines `base_dir`, `img_path`, and `label_path` for dataset directories.

- Load Classes: Reads and loads class labels from `classes.json` into the `classes` variable, then displays the content.

```
ims = sorted(list(img_path.glob('*')))
labels = sorted(list(label_path.glob('*')))

# Print the first few entries to verify
print(ims[:2])
print(labels[:2])
```

Figure 13: This code lists and sorts image and label files

- List Files: Retrieves and sorts file paths from `img_path` and `label_path`.

- Verify: Prints the first two entries from both lists to check that the files have been correctly listed and sorted.

```
ims = sorted(list(img_path.glob('*')))
labels = sorted(list(label_path.glob('*')))
pairs = list(zip(ims,labels))
pairs[:2]
```

Figure 14: This code pairs image files with their corresponding labels

- List and Sort: Retrieves and sorts image and label file paths.

- Pair Files: Zips the sorted image and label lists together into pairs.

- Check Pairs: Displays the first two image-label pairs to verify correct pairing.

```
train, test = train_test_split(pairs,test_size=0.1,shuffle=True)
len(train), len(test)
```

Figure 15: This code splits the image-label pairs into training and testing sets

- Split Data: Divides the `pairs` list into training and testing sets with 10% of data allocated to testing.

- Check Sizes: Prints the number of items in the training and testing sets.

```
train_path = Path('train').resolve()
train_path.mkdir(exist_ok=True)
valid_path = Path('valid').resolve()
valid_path.mkdir(exist_ok=True)
```

Figure 16: This code creates directories for storing training and validation data

- `train_path`: Creates a `train` directory if it doesn't exist.

- `valid_path`: Creates a `valid` directory if it doesn't exist.

```
for t_img, t_lb in tqdm(train):
    im_path = train_path / t_img.name
    lb_path = train_path / t_lb.name
    shutil.copy(t_img,im_path)
    shutil.copy(t_lb,lb_path)
```

Figure 17: This code copies training images and labels to the `train` directory

- Loop Through Pairs: Iterates over image-label pairs in the training set.

- Copy Files: Copies each image and its corresponding label to the `train` directory.

```
for t_img, t_lb in tqdm(test):
    im_path = valid_path / t_img.name
    lb_path = valid_path / t_lb.name
    shutil.copy(t_img,im_path)
    shutil.copy(t_lb,lb_path)
```

Figure 18: This code copies test images and labels to the `valid` directory

- Loop Through Pairs: Iterates over image-label pairs in the test set.

- Copy Files: Copies each image and its corresponding label to the `valid` directory.

```
yaml_file = 'names:\n'
yaml_file += '\n'.join(f'- {c}' for c in classes)
yaml_file += f'\nnc: {len(classes)}'
yaml_file += f'\ntrain: {str(train_path)}\nval: {str(valid_path)}'
with open('kitti.yaml','w') as f:
    f.write(yaml_file)
```

Figure 19: This code generates and saves a YAML configuration file

- Create YAML Content: Constructs YAML text with class names, number of classes, and paths for training and validation data.

- Save to File: Writes the YAML content to `kitti.yaml`.

```
!cat kitti.yaml
```

Figure 20: This command displays the contents of the `kitti.yaml` file in the output.

```
model = YOLO('yolov8n.yaml')
model = YOLO('yolov8n.pt')
```

Figure 21: This code initializes a YOLO model

1. Load Configuration: `YOLO('yolov8n.yaml')` initializes the model with a YAML configuration file.

2. Load Pretrained Weights: `YOLO('yolov8n.pt')` loads a YOLO model with pretrained weights from the `yolov8n.pt` file.

```
train_results = model.train(
    data='/content/kitti.yaml',
    epochs=10,
    patience=3,
    mixup=0.1,
    project='yolov8n-kitti'
)
```

Figure 22: This code trains the YOLO model

- Train Model: Uses the configuration specified in `kitti.yaml`.

- Parameters:

  - `epochs=10`: Runs for 10 epochs.

  - `patience=3`: Stops early if no improvement is seen for 3 epochs.

- `mixup=0.1`: Applies mixup data augmentation with a 10% mix rate.

- `project='yolov8n-kitti'`: Saves results in the `yolov8n-kitti` project directory.

```
valid_results = model.val()
```

Figure 23: This code evaluates the trained YOLO model on the validation dataset

- Validate Model: Runs the validation process and returns performance metrics for the validation set.

```
plt.figure(figsize=(10,20))
plt.imshow(Image.open('/content/yolov8n-kitti/train22/confusion_matrix.png'))
plt.axis('off')
plt.show()
```

Figure 24: This code displays a confusion matrix

- Load Image: Opens the confusion matrix image from the specified path.

- Plot Image: Shows the image in a 10x20 inch figure without axis labels.

```
plt.figure(figsize=(10,20))
plt.imshow(Image.open('/content/yolov8n-kitti/train22/val_batch0_pred.jpg'))
plt.axis('off')
plt.show()
```

Figure 25: This code displays a prediction image

- Load Image: Opens the prediction image from the specified path.

- Plot Image: Shows the image in a 10x20 inch figure without axis labels.

```
preds = model.predict([test[idx][0] for idx in np.random.randint(0,len(test),(20,))],save=True)
```

Figure 26: This code makes predictions on a random sample of 20 test images

- Random Sample: Selects 20 random test images.

- Predict: Runs the model's prediction on these images and saves the results.

```
preds = list(Path('/content/yolov8n-kitti/train23').glob('*'))
```

Figure 27: This code retrieves all files from the specified directory:

- List Files: Collects and lists all files in the `/content/yolov8n-kitti/train23` directory.

```python
def plot_images(images):
    num_images = len(images)
    rows = num_images
    cols = 1
    fig, axes = plt.subplots(rows, cols, figsize=(15, 80))
    for ax in axes.flat:
        ax.axis('off')
    for i, img_path in enumerate(images):
        img = Image.open(img_path)
        axes[i].imshow(img)

    plt.tight_layout()
    plt.show()

plot_images(preds)
```

Figure 28: This code defines and uses a function to display a list of images

1. Define Function: `plot_images(images)` creates a vertical grid of subplots for each image in the list.

2. Load and Plot: Opens each image and displays it in the subplot, with each image shown in a column of its own.

3. Show Plot: Adjusts layout and displays the images.

```python
plot_images(preds)
```

Figure 29: This command uses the `plot_images` function to display all images listed in `preds`

- Display Images: Shows the images in a vertical grid format, with each image occupying its own subplot.

```python
import shutil
from google.colab import files

# Specify the folder you want to zip
folder_to_zip = '/content/yolov8n-kitti/train23'  # Change this to your folder's name
output_filename = 'prediction_folder.zip'

# Zip the folder
shutil.make_archive(output_filename.replace('.zip', ''), 'zip', folder_to_zip)

# Download the zipped folder
files.download(output_filename)
```

Figure 30: This code zips a folder and downloads it

1. Zip Folder: Compresses `/content/yolov8n-kitti/train23` into `prediction_folder.zip`.

2. Download: Uses Google Colab's `files.download` to download the zipped folder to your local machine.

```
import shutil
from google.colab import files

# Specify the folder you want to zip
folder_to_zip = '/content/yolov8n-kitti/train22'  # Change this to your folder's name
output_filename = 'validation_folder.zip'

# Zip the folder
shutil.make_archive(output_filename.replace('.zip', ''), 'zip', folder_to_zip)

# Download the zipped folder
files.download(output_filename)
```

Figure 31: This code compresses and downloads a folder

1. Zip Folder: Compresses `/content/yolov8n-kitti/train22` into `validation_folder.zip`.

2. Download: Uses Google Colab's `files.download` to download the zipped folder to your local machine.

```
import shutil
from google.colab import files

# Specify the folder you want to zip
folder_to_zip = '/content/yolov8n-kitti/train2'  # Change this to your folder's name
output_filename = 'results.zip'

# Zip the folder
shutil.make_archive(output_filename.replace('.zip', ''), 'zip', folder_to_zip)

# Download the zipped folder
files.download(output_filename)
```

Figure 32: This code compresses a folder and prepares it for download

1. Zip Folder: Compresses `/content/yolov8n-kitti/train2` into `results.zip`.

2. Download: Uses `files.download` to download the zipped folder to your local machine.

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

# Specify the log directory
log_dir = '/content/yolov8n-kitti/train2'  # Update with your actual path
log_file = os.path.join(log_dir, 'results.csv')

# Load the CSV file
log_data = pd.read_csv(log_file)
log_data.columns = log_data.columns.str.strip()  # Remove leading/trailing spaces
print(log_data.head())

# Extract metrics
epochs = log_data.index
box_loss = log_data['train/box_loss'] if 'train/box_loss' in log_data.columns else None
obj_loss = log_data['train/obj_loss'] if 'train/obj_loss' in log_data.columns else None
cls_loss = log_data['train/cls_loss'] if 'train/cls_loss' in log_data.columns else None
val_box_loss = log_data['val/box_loss'] if 'val/box_loss' in log_data.columns else None
val_obj_loss = log_data['val/obj_loss'] if 'val/obj_loss' in log_data.columns else None
val_cls_loss = log_data['val/cls_loss'] if 'val/cls_loss' in log_data.columns else None

# Plot loss curves
plt.figure(figsize=(10, 5))

# Plot training loss curves
if box_loss is not None:
    plt.subplot(1, 2, 1)
    plt.plot(epochs, box_loss, label='Box Loss')
if obj_loss is not None:
    plt.plot(epochs, obj_loss, label='Objectness Loss')
if cls_loss is not None:
    plt.plot(epochs, cls_loss, label='Classification Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Curves')
plt.legend()
```

```python
# Plot validation loss curves
if val_box_loss is not None:
    plt.subplot(1, 2, 2)
    plt.plot(epochs, val_box_loss, label='Validation Box Loss')
if val_obj_loss is not None:
    plt.plot(epochs, val_obj_loss, label='Validation Objectness Loss')
if val_cls_loss is not None:
    plt.plot(epochs, val_cls_loss, label='Validation Classification Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Validation Loss Curves')
plt.legend()

plt.tight_layout()
plt.show()
```

Figure 33: This code analyzes and visualizes training and validation loss metrics

1. Load Data: Reads the `results.csv` file from the specified log directory into a DataFrame.

2. Extract Metrics: Retrieves loss metrics for training and validation, if available.

3. Plot Loss Curves:

- Training Loss: Plots box loss, objectness loss, and classification loss over epochs.

- Validation Loss: Plots validation losses similarly.

It shows two side-by-side plots for training and validation loss curves.

```
import os

log_dir = '/content/yolov8n-kitti/train2'  # Update with your actual path
print(os.listdir(log_dir))
```

Figure 34: This code lists the contents of the specified directory

- List Files: Displays all files and folders within `/content/yolov8n-kitti/train2` to help you identify available log files and directories.

1.2 Yolov9

```
from ultralytics import YOLO
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import json
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm
import shutil
from PIL import Image
```

Figure 35: This code imports various libraries and modules for working with the YOLO object detection model and handling data

- `ultralytics.YOLO`: Imports the YOLO model for object detection.

- `numpy`, `pandas`, `matplotlib.pyplot`: For numerical operations, data manipulation, and plotting.

- `pathlib.Path`: For handling filesystem paths.

- `json`: For parsing JSON data.

- `sklearn.model_selection.train_test_split`: For splitting data into training and test sets.

- `tqdm.auto`: For displaying progress bars.

- `shutil`: For file operations.

- `PIL.Image`: For image processing.

```
base_dir = Path('content/dataset/')
img_path = Path('/content/dataset/data_object_image_2/training/image_2')
label_path = Path('/content/dataset_yolo/labels')
with open('/content/dataset_yolo/classes.json','r') as f:
    classes = json.load(f)

classes
```

Figure 36: This code sets up paths for image and label directories, then loads class labels from a JSON file

- Paths Setup: Defines `base_dir`, `img_path`, and `label_path` for dataset directories.

- Load Classes: Reads and loads class labels from `classes.json` into the `classes` variable, then displays the content.

```
ims = sorted(list(img_path.glob('*')))
labels = sorted(list(label_path.glob('*')))

# Print the first few entries to verify
print(ims[:2])
print(labels[:2])
```

Figure 37: This code lists and sorts image and label files

- List Files: Retrieves and sorts file paths from `img_path` and `label_path`.

- Verify: Prints the first two entries from both lists to check that the files have been correctly listed and sorted.

```
ims = sorted(list(img_path.glob('*')))
labels = sorted(list(label_path.glob('*')))
pairs = list(zip(ims,labels))
pairs[:2]
```

Figure 38: This code pairs image files with their corresponding labels:

- List and Sort: Retrieves and sorts image and label file paths.

- Pair Files: Zips the sorted image and label lists together into pairs.

- Check Pairs: Displays the first two image-label pairs to verify correct pairing.

```
train, test = train_test_split(pairs,test_size=0.1,shuffle=True)
len(train), len(test)
```

Figure 39: This code splits the image-label pairs into training and testing sets

- Split Data: Divides the `pairs` list into training and testing sets with 10% of data allocated to testing.

- Check Sizes: Prints the number of items in the training and testing sets.

```
train_path = Path('train').resolve()
train_path.mkdir(exist_ok=True)
valid_path = Path('valid').resolve()
valid_path.mkdir(exist_ok=True)
```

Figure 40: This code creates directories for storing training and validation data

- `train_path`: Creates a `train` directory if it doesn't exist.

- `valid_path`: Creates a `valid` directory if it doesn't exist.

```
for t_img, t_lb in tqdm(train):
    im_path = train_path / t_img.name
    lb_path = train_path / t_lb.name
    shutil.copy(t_img,im_path)
    shutil.copy(t_lb,lb_path)
```

Figure 41: This code copies training images and labels to the `train` directory

- Loop Through Pairs: Iterates over image-label pairs in the training set.

- Copy Files: Copies each image and its corresponding label to the `train` directory.

```
for t_img, t_lb in tqdm(test):
    im_path = valid_path / t_img.name
    lb_path = valid_path / t_lb.name
    shutil.copy(t_img,im_path)
    shutil.copy(t_lb,lb_path)
```

Figure 42: This code copies test images and labels to the `valid` directory

- Loop Through Pairs: Iterates over image-label pairs in the test set.

- Copy Files: Copies each image and its corresponding label to the `valid` directory.

```
yaml_file = 'names:\n'
yaml_file += '\n'.join(f'- {c}' for c in classes)
yaml_file += f'\nnc: {len(classes)}'
yaml_file += f'\ntrain: {str(train_path)}\nval: {str(valid_path)}'
with open('kitti.yaml','w') as f:
    f.write(yaml_file)
```

Figure 43: This code generates and saves a YAML configuration file

```
import os
import pandas as pd
import matplotlib.pyplot as plt

# Specify the log directory
log_dir = '/content/yolov8n-kitti/train2'  # Update with your actual path
log_file = os.path.join(log_dir, 'results.csv')

# Load the CSV file
log_data = pd.read_csv(log_file)
log_data.columns = log_data.columns.str.strip()  # Remove leading/trailing spaces
print(log_data.head())

# Extract metrics
epochs = log_data.index
box_loss = log_data['train/box_loss'] if 'train/box_loss' in log_data.columns else None
obj_loss = log_data['train/obj_loss'] if 'train/obj_loss' in log_data.columns else None
cls_loss = log_data['train/cls_loss'] if 'train/cls_loss' in log_data.columns else None
val_box_loss = log_data['val/box_loss'] if 'val/box_loss' in log_data.columns else None
val_obj_loss = log_data['val/obj_loss'] if 'val/obj_loss' in log_data.columns else None
val_cls_loss = log_data['val/cls_loss'] if 'val/cls_loss' in log_data.columns else None

# Plot loss curves
plt.figure(figsize=(10, 5))

# Plot training loss curves
if box_loss is not None:
    plt.subplot(1, 2, 1)
    plt.plot(epochs, box_loss, label='Box Loss')
if obj_loss is not None:
    plt.plot(epochs, obj_loss, label='Objectness Loss')
if cls_loss is not None:
    plt.plot(epochs, cls_loss, label='Classification Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Curves')
plt.legend()
```

```
# Plot validation loss curves
if val_box_loss is not None:
    plt.subplot(1, 2, 2)
    plt.plot(epochs, val_box_loss, label='Validation Box Loss')
if val_obj_loss is not None:
    plt.plot(epochs, val_obj_loss, label='Validation Objectness Loss')
if val_cls_loss is not None:
    plt.plot(epochs, val_cls_loss, label='Validation Classification Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Validation Loss Curves')
plt.legend()

plt.tight_layout()
plt.show()
```

Figure 44: This code analyzes and visualizes training and validation loss metrics

1. Load Data: Reads the `results.csv` file from the specified log directory into a DataFrame.

2. Extract Metrics: Retrieves loss metrics for training and validation, if available.

3. Plot Loss Curves:

- Training Loss: Plots box loss, objectness loss, and classification loss over epochs.

- Validation Loss: Plots validation losses similarly.

It shows two side-by-side plots for training and validation loss curves.

```
import os

log_dir = '/content/yolov8n-kitti/train2'  # Update with your actual path
print(os.listdir(log_dir))
```

Figure 45: This code lists the contents of the specified directory:

- List Files: Displays all files and folders within `/content/yolov8n-kitti/train2` to help you identify available log files and directories.

1.3 Yolov10

```
base_dir = Path('content/dataset/')
img_path = Path('/content/dataset/data_object_image_2/training/image_2')
label_path = Path('/content/dataset_yolo/labels')
with open('/content/dataset_yolo/classes.json','r') as f:
    classes = json.load(f)

classes
```

Figure 46: This code sets up paths for image and label directories and loads class labels

1. Define Paths: Sets `base_dir`, `img_path`, and `label_path` for dataset locations.

2. Load Classes: Reads and loads class names from `classes.json` into the `classes` variable, then displays the loaded class labels.

```
ims = sorted(list(img_path.glob('*')))
labels = sorted(list(label_path.glob('*')))

# Print the first few entries to verify
print(ims[:2])
print(labels[:2])
```

Figure 47: This code retrieves and sorts image and label file paths

- List and Sort: Gathers and sorts file paths from the `img_path` and `label_path` directories.

- Verify: Prints the first two image and label paths to ensure correct listing and sorting.

```
ims = sorted(list(img_path.glob('*')))
labels = sorted(list(label_path.glob('*')))
pairs = list(zip(ims,labels))
pairs[:2]
```

Figure 48: This code pairs image files with their corresponding labels

- List and Sort: Retrieves and sorts image and label file paths.

- Create Pairs: Zips the sorted image and label lists together.

- Check Pairs: Displays the first two image-label pairs to verify correct pairing.

```
train, test = train_test_split(pairs,test_size=0.1,shuffle=True)
len(train), len(test)
```

Figure 49: This code splits the paired image-label data into training and testing sets

- Split Data: Divides the `pairs` list into `train` (90%) and `test` (10%) sets.

- Check Sizes: Prints the number of items in the training and testing sets.

```
train_path = Path('train').resolve()
train_path.mkdir(exist_ok=True)
valid_path = Path('valid').resolve()
valid_path.mkdir(exist_ok=True)
```

Figure 50: This code creates directories for training and validation data

- Create Directories: Creates `train` and `valid` directories if they don't already exist, ensuring they're ready for storing data.

```
for t_img, t_lb in tqdm(train):
    im_path = train_path / t_img.name
    lb_path = train_path / t_lb.name
    shutil.copy(t_img,im_path)
    shutil.copy(t_lb,lb_path)
```

Figure 51: This code copies training images and labels to the `train` directory

- Iterate and Copy: For each image-label pair in the training set, copies the image and its corresponding label to the `train` directory. The progress is shown using a progress bar.

```
for t_img, t_lb in tqdm(test):
    im_path = valid_path / t_img.name
    lb_path = valid_path / t_lb.name
    shutil.copy(t_img,im_path)
    shutil.copy(t_lb,lb_path)
```

Figure 52: This code copies test images and labels to the `valid` directory:

- Iterate and Copy: For each image-label pair in the test set, it copies the image and label to the `valid` directory, with progress displayed by a progress bar.

```python
from ultralytics import YOLO

# Load YOLOv10n model from scratch
model = YOLO("yolov10n.pt")

# Train the model
model.train(data="kitti.yaml", epochs=100, imgsz=640)
```

Figure 53: Yolov10 weights load and training

Here's a concise breakdown of the code:

- `from ultralytics import YOLO`: Imports the YOLO class from the `ultralytics` library.

- `model = YOLO("yolov10n.pt")`: Loads the YOLOv10n model with pre-trained weights from the `yolov10n.pt` file.

- `model.train(data="kitti.yaml", epochs=100, imgsz=640)`: Trains the model using the dataset specified in `kitti.yaml` for 100 epochs with an image size of 640x640 pixels.

```
!yolo task=detect mode=train epochs=10 batch=16 plots=true model=yolov10n.pt data=kitti.yaml
```

Figure 54: This command trains a YOLOv10 model for object detection using the `yolo` CLI tool

- `task=detect`: Specifies the task as object detection.

- `mode=train`: Sets the mode to training.

- `epochs=10`: Trains for 10 epochs.

- `batch=16`: Uses a batch size of 16.

- `plots=true`: Enables plotting of training metrics.

- `model=yolov10n.pt`: Uses `yolov10n.pt` as the pre-trained model weights.

- `data=kitti.yaml`: Uses `kitti.yaml` for dataset configuration.

2. Steps to Reproduce

Step 1: Stage the code in the google colab

Step 2: Authenticate the google drive in which this google colab is linked

Step 3: Execute each step

Step 4: Verify the results

Reference:

[1] https://encord.com/blog/yolo-object-detection-guide/

[2] https://docs.ultralytics.com/

[3] https://www.datacamp.com/blog/yolo-object-detection-explained