University for the Creative Arts

BERLIN SCHOOL OF BUSINESS & INNOVATION

**Essay / Assignment Title: Enhancing Environmental Monitoring through Advanced Object Detection in Satellite Imagery**

**Programme title: MSc Data Analytics**

**Name: Birce SARI**

**Year: 2024**

## TABLE OF CONTENTS

## Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the postgraduate program).

Name and Surname (Capital letters):

.....BIRCE SARI.....

Date: ...31...../...10.../..24..

## INTRODUCTION: PROBLEM FORMULATION

Automation plays a great part in our lives after traditional methods of time-consuming manual data collection and analysis are insufficient with the need for more human labor. They were successful until recently, whereas an evolving era needs more real-time surveillance around the world (Envirosuite, 2024). Automated systems provide continuous data streams with reduced human errors, less need for skilled personnel (Zainurin et al., 2022), and increased reliability while expanding their reach for environmental research (Nazarov et al., 2024) to inaccessible areas. Real-time data collection, and cost-effectiveness in long-term monitoring are significant benefits of object detection that help rapid responses to changes, decision-making processes, and early detection of problems (Sani et al., 2023). Object detection can help environmental research institutes with satellite imagery analysis and predictions of urban sprawl areas, water-body shrinkage problems, fire detection possibilities, or deforestation.

Automated interpretation of satellite images helps environmental monitoring to realize the problems in advance than traditional analysis. Architectures like faster R-CNN and YOLO are particularly effective for object detection with identifying them. Identifying small objects requires more precise detection. We will be considering their limitations and efficiencies in environmental research. The airplane detection is the main subject to monitor the existing of them to demonstrate one class object detection and corresponding applications for other fields. Yolo is a one-stage (proposal free) algorithm while faster R-CNN is two stage (proposal) object detection algorithm.

Object detection algorithms in general is based on regional proposal generator, feature extraction, and classification. Region-based convolutional neural networks (R-CNN) has a pipeline of input being taken, extracted the regions accordingly, computing the convolutional neural networks (CNN) for feature extraction, and object classification. Where Faster R-CNN is introducing a new layer called ROI pooling for equal length vectors with one stage to decrease the computational weight of the problem, and also not keeping the information about the unused features for decreasing the disk usage that ends up with a faster and more accurate process than R-CNN models (Gad & Skeltonn, 2024). In addition to ROI layer, a region proposal network

(RPN) is added before ROI layer to scan all the image and defines the area of objects before computations. This lead to a higher accuracy in small object detection especially in occlusion problems. Its ability with different scaling object detection makes it particularly suitable for analysis.

YOLO (you only look once) is a model that divides the images into grids to predict bounding boxes and computes the class probabilities for each grid cell in a single forward pass. This helps us to use this model for real-time applications which requires fast processing. The model has fewer parameters than faster R-CNN and the single-row grid makes the processing favorable for bigger images.

| Feature/Mode | Faster R-CNN | YOLO |
|---|---|---|
| **Architecture** | Combines region proposal networks with a fully convolutional network for instance segmentation and object detection. | A single-stage model that processes the entire image at once, predicting bounding boxes and class probabilities simultaneously. |
| **Output** | Produces bounding boxes, class labels, and segmentation masks for each object. | Outputs bounding boxes and class probabilities, focused on speed. |
| **Speed** | Slower due to its two-stage process, making it less suitable for real-time applications. | Extremely fast, making it ideal for real-time applications and processing large volumes of satellite data. |
| **Accuracy** | High accuracy, especially in detecting small objects and providing detailed segmentation. | High accuracy with a focus on detecting larger objects; recent versions have improved small object detection. |
| **Use Cases** | Best suited for applications requiring detailed object delineation, such as habitat mapping and land cover classification. | Ideal for real-time environmental monitoring, such as tracking wildlife or observing changes in land use. |
| **Complexity** | More complex due to its two-stage nature and additional components for segmentation. | Less complex, with straightforward implementation and rapid deployment. |

**Figure 1: Differences in architectural designs and their strengths**

## CHAPTER ONE: DATA PREPARATION

Our key indicator is airplane or aircraft detection. Our dataset is 'SkyFusion: Aerial Object Detection' by Sudheer prepared in 2023 with satellite images of different areas. The dataset contains three subfolders; train, test, and valid with each of their annotations. We repeated the process for three time to view the sample images and reach the file count for further analysis. Dataset_inspection.py file is created for visualization needs.

```python
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import os
import random

def load_image(path):  1 usage
    img = Image.open(path).convert('RGB')
    img = img.resize((300,300))  # Resize for consistency
    return np.array(img)

# Function to get random images from a folder
def get_random_images(folder_path,num_images=12):  1 usage
    image_files = [f for f in os.listdir(folder_path) if
                   f.lower().endswith(('.jpg'))]

    if len(image_files) < num_images:
        print(f"Warning: Only {len(image_files)} images found in the folder.")
        num_images = len(image_files)

    selected_images = random.sample(image_files,num_images)
    return [os.path.join(folder_path,img) for img in selected_images]

# Specify the folder containing your images
folder_path = 'SkyFusion/train'

# Get 12 random image paths
image_paths = get_random_images(folder_path)
```

**Figure 2: Code snippet for visualizing the dataset elements randomly (part 1)**

```python
# Load the images
images = [load_image(path) for path in image_paths]

# Count files by type
image_counts, other_counts = count_files_by_type(folder_path)

# Create the 3x4 grid and display the images
fig, axes = plt.subplots( nrows: 3,  ncols: 4, figsize=(15, 12))
fig.subplots_adjust(hspace=0.3, wspace=0.3)

# Set the title for the entire figure
plt.suptitle( t: 'Images from Validation Set', fontsize=16)

for i, ax in enumerate(axes.flat):
    if i < len(images):
        ax.imshow(images[i])
        #ax.set_title(os.path.basename(image_paths[i]), fontsize=8)
        ax.axis('off')
    else:
        ax.remove()  # Remove unused subplots

# Add file count information
image_info = "Image file counts:\n" + "\n".join([f"{ext}: {count}" for ext, count in sorted(image_counts.items())])
other_info = "Other file counts:\n" + "\n".join([f"{ext}: {count}" for ext, count in sorted(other_counts.items())])

plt.figtext( x: 0.02,  y: 0.02, image_info, fontsize=10, va="bottom")
plt.figtext( x: 0.5,  y: 0.02, other_info, fontsize=10, va="bottom")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout to make room for the title
plt.show()
```

**Figure 3: Code snippet for visualizing the dataset elements randomly (part 2)**

We obtained that train images (2094), test images (449), and validation images (449) counts.

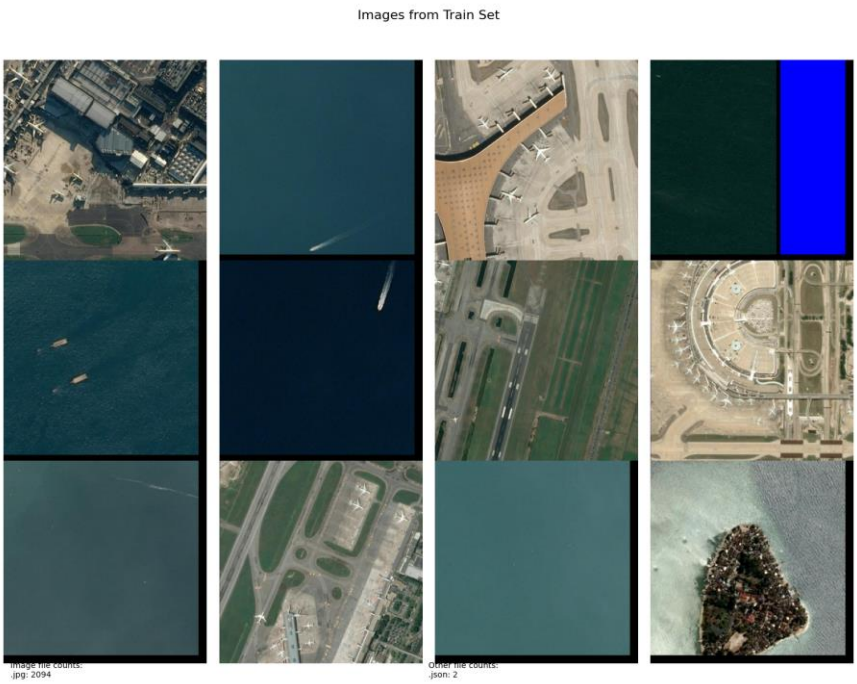


**Figure 4: Output for visualizing the train dataset elements randomly**

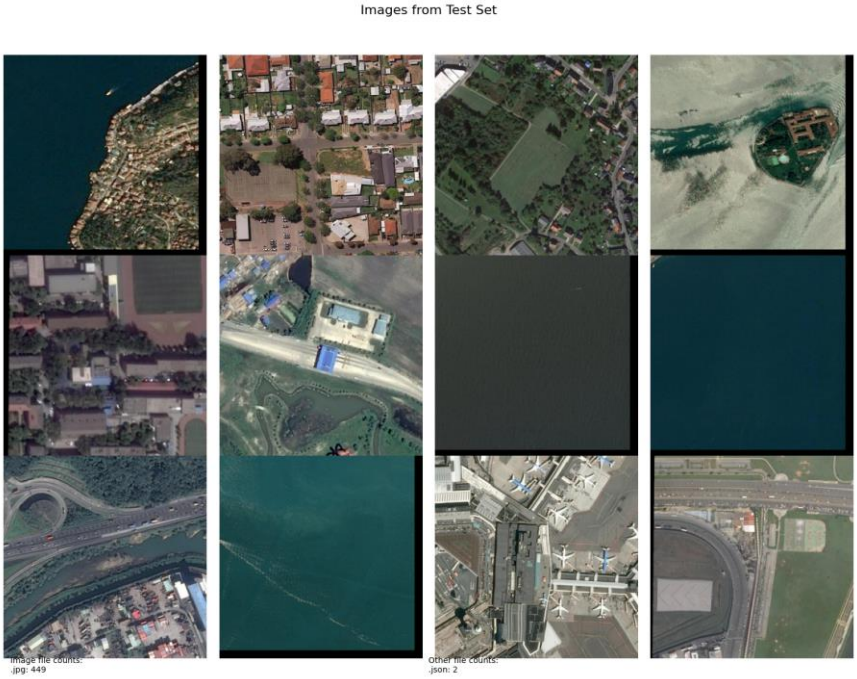


**Figure 5: Output for visualizing the test dataset elements randomly**

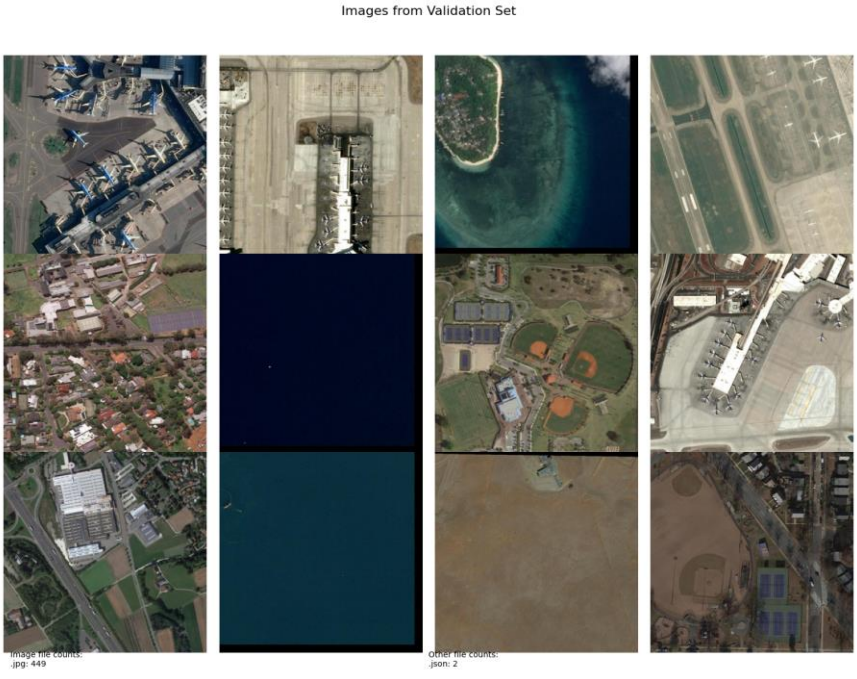


**Figure 6: Output for visualizing the validation dataset elements randomly**

After the visualization of the actual dataset, our project will continue in calculations to understand the nature of the YOLO model. Due to technical problems in my computer, the YOLO model is failed therefore the preparation is done in one environment, and the model constructed in another one, unfortunately. While controlling different model computations, my computer lost its package definition processes and started giving an 'undefined Numpy error', just as I showed you in the classroom. Therefore, the previous codes and outcomes will be used to clean and separate the data even uninstalling and reinstalling programs didn't solve this problem in the model preparation duration. Faster R-CNN will show only one epoch as output and will be described hypothetically to be able to make comparison. This model's separation of files used for data separating process for supervised learning for airplane existence versus non-existence under name of Faster R-CNN_aerial_DataCleaning.py.

Speed, accuracy, and scalability questioned us to choose the best model for our purpose. Packages had problems because of the technical incapability of hardware, special modeling was needed to mimic similar processes. Since the actual code couldn't apply to our case, we needed to compare the results.

We need to separate relevant images first to control them thoroughly with the existence of either of the classes. To be able to do that we divided the dataset into two subfolders. Only two categories of objects are considered as mentioned before and declared in both codes accordingly.

```
import os
import json
import shutil
from PIL import Image, ImageDraw

# Define desired categories for filtering
categories = ['airplane', 'aircraft']

# List of paths to the annotation files and corresponding image directories
dataset_splits = [
    {'name': 'train', 'annotation_file': 'SkyFusion/train/_annotations.coco.json', 'input_image_dir': 'SkyFusion/train/'},
    {'name': 'valid', 'annotation_file': 'SkyFusion/valid/_annotations.coco.json', 'input_image_dir': 'SkyFusion/valid/'},
    {'name': 'test', 'annotation_file': 'SkyFusion/test/_annotations.coco.json', 'input_image_dir': 'SkyFusion/test/'}
]

# Define base paths for output
base_output_dir = 'SkyFusion/yolo'
output_with_detections_dir = os.path.join(base_output_dir, 'detections')
output_without_detections_dir = os.path.join(base_output_dir, 'no_detections')
```

**Figure 7: importing variables and directory introduction to separate the detected and undetected images**

We split the folders for learning and testing usage. That gives us the labels and images in different directories and groups the ones with an object and without any object detected.

```
# Loop through dataset splits (train, valid, test)
for split in dataset_splits:
    split_name = split['name']
    annotation_file = split['annotation_file']
    input_image_dir = split['input_image_dir']

    # Define paths for split-specific images and labels in both folders
    output_image_dir_with_detections = os.path.join(output_with_detections_dir, split_name, 'images')
    output_label_dir_with_detections = os.path.join(output_with_detections_dir, split_name, 'labels')
    output_image_dir_without_detections = os.path.join(output_without_detections_dir, split_name, 'images')
    output_label_dir_without_detections = os.path.join(output_without_detections_dir, split_name, 'labels')

    # Ensure output directories exist
    os.makedirs(output_image_dir_with_detections, exist_ok=True)
    os.makedirs(output_label_dir_with_detections, exist_ok=True)
    os.makedirs(output_image_dir_without_detections, exist_ok=True)
    os.makedirs(output_label_dir_without_detections, exist_ok=True)
```

**Figure 8: Dataset split for aircraft existence and separation of annotation.json file**

This code shows us the reading of annotation files, categorizing them according to provided categories, and throws whether the class is found or not. Then a dictionary is reserved for our ‘for loop’ which will be run over each of our images.

```
# Load the COCO annotations file
with open(annotation_file) as f:
    coco_data = json.load(f)

# Find category_id for "airplane" or "aircraft"
airplane_category_id = None
for category in coco_data['categories']:
    if category['name'].lower() in categories:
        airplane_category_id = category['id']
        break

if airplane_category_id is None:
    print(f"Category not found in {annotation_file}. Skipping...")
    continue
```

**Figure 9: Annotation readings for categories**

```
# Dictionary to store annotations for each image containing airplanes
annotations_dict = {}

# Filter annotations for the desired category
for annotation in coco_data['annotations']:
    if annotation['category_id'] == airplane_category_id:
        image_id = annotation['image_id']
        if image_id not in annotations_dict:
            annotations_dict[image_id] = []
        annotations_dict[image_id].append(annotation)
```

**Figure 10: Annotation set creation by defining the image files accordingly**

The numbers will be incrementally increased due to the image controls, therefore we defined data blocks for each variable to be adjusted accordingly in the evaluation step. This part can be used training model and then controlling predictions. However since our prototype is conducted by single run, the numbers will not give accurate evaluation.

```
# Initialize evaluation metrics
total_true_positives = 0
total_false_positives = 0
total_false_negatives = 0
```

**Figure 11: Evaluation introductions to the code**

```
# Process images with relevant annotations
for image in coco_data['images']:
    image_id = image['id']
    if image_id in annotations_dict:
        ground_truth_boxes = annotations_dict[image_id]
        if not ground_truth_boxes:
            print(f"No ground truth boxes found for image {image_id}")
            continue

        src_image_path = os.path.join(input_image_dir, image['file_name'])
        dst_image_path = os.path.join(output_image_dir_with_detections, image['file_name'])

        try:
            img = Image.open(src_image_path).convert("RGB")
            # Apply augmentations only to the training set
            if split_name == 'train':
                img = apply_augmentations(img)
            img.save(dst_image_path)
        except Exception as e:
            print(f"Error processing {src_image_path}: {e}")
            continue
```

**Figure 12: Image annotating, drawing area and distributing the corresponding folders code snippet (part 1)**

```
        # Open image to draw bounding boxes
        with Image.open(dst_image_path) as img:
            draw = ImageDraw.Draw(img)
            img_width, img_height = img.size

            yolo_labels = []
            detected_ground_truths = set()

            for annotation in ground_truth_boxes:
                if 'bbox' not in annotation or len(annotation['bbox']) != 4:
                    print(f"Invalid bbox format in annotation for image {image_id}")
                    continue

                bbox = annotation['bbox']
                x_min, y_min, width, height = bbox
                x_max = x_min + width
                y_max = y_min + height

                # Draw the rectangle on the image
                draw.rectangle( xy: [x_min, y_min, x_max, y_max], outline="red", width=2)

                # Convert to YOLO format
                x_center = x_min + width / 2
                y_center = y_min + height / 2
                x_center /= img_width
                y_center /= img_height
                width /= img_width
                height /= img_height

                yolo_labels.append(f"{airplane_category_id - 1} {x_center} {y_center} {width} {height}\n")

                # Evaluation (assuming perfect detection for this example)
                total_true_positives += 1
                detected_ground_truths.add(len(detected_ground_truths))

            total_false_negatives += len(ground_truth_boxes) - len(detected_ground_truths)

            # Save the image with bounding boxes
            img.save(dst_image_path)
```

**Figure 13: Image annotating, drawing area and distributing the corresponding folders code snippet (part 2)**

```
            # Write YOLO format labels
            yolo_label_path = os.path.join(output_label_dir_with_detections, image['file_name'].replace('.jpg', '.txt'))
            with open(yolo_label_path, 'w') as label_file:
                label_file.writelines(yolo_labels)

    else:
        # Copy images without detections to the 'no_detections' folder
        src_image_path = os.path.join(input_image_dir, image['file_name'])
        dst_image_path = os.path.join(output_image_dir_without_detections, image['file_name'])
        shutil.copy(src_image_path, dst_image_path)

        # Create an empty label file
        yolo_label_path = os.path.join(output_label_dir_without_detections, image['file_name'].replace('.jpg', '.txt'))
        open(yolo_label_path, 'w').close()
```

**Figure 14: Image annotating, drawing area and distributing the corresponding folders code snippet (part 3)**

## CHAPTER TWO: MODEL IMPLEMENTATION

We needed to calculate the intercept of union (IoU) values are the overlapping information for two boxes that may share a portion of the area, this calculation will be used in further selection parts. In the YOLO model, we eliminated the smaller size for more accurate findings.

```python
# Function to calculate IoU
def calculate_iou(box1, box2):  1 usage
    x1, y1, w1, h1 = box1
    x2, y2, w2, h2 = box2

    # Calculate overlap coordinates
    x_left = max(x1, x2)
    y_top = max(y1, y2)
    x_right = min(x1 + w1, x2 + w2)
    y_bottom = min(y1 + h1, y2 + h2)

    if x_right < x_left or y_bottom < y_top:
        return 0.0  # No overlap

    intersection_area = (x_right - x_left) * (y_bottom - y_top)
    box1_area = w1 * h1
    box2_area = w2 * h2
    union_area = box1_area + box2_area - intersection_area

    return intersection_area / union_area
```

**Figure 15: Model1 intercept of union (IoU) calculation**

To understand our process, architectural differences need to be considered. Literature shows us that both models have significant usage but they have different approaches. Model1 proposes regions (RPN), feature maps are extracted from the RoI Pooling layer, and the classification network refines the detection boxes. Model proposes grid cells with key features and class probabilities and then puts them into a single network pass to compute them faster.

Annotations are labels for images or videos to help machine learning processes for supervised learning algorithms. They are used in object detection and image classification models to provide interfaces for drawing bounding boxes, segmenting different regions, or pointing special features. In our project, annotations are supplied from the original dataset however, for special processes, it is possible to make them manually (Alvi, 2024).

The COCO dataset is helping us with the training process to enhance our model with pre-trained examples. Each image has given an ID for easier identification throughout the system for debugging reasons. Imaged are processed with augmentation (flipping, rotating, and color

adjustments) for a diversity of training images for a stronger training process. Unsqueeze(0), with torch.no_grad() functions added for compatibility and to avoid calculating gradients. Saving outputs, bounding boxes, labels and scores will be helpful for the evaluation step. Then the classifications are filtered to keep only the categories that we introduced in the beginning. The images are distributed due to object detection. As the end line shows, the image is saved to the assigned directory with the stored information.

The virtual dataset enlargement, and augmentation, were applied to both models to enhance the detection process. Randomly flipped images in 50% of the train set, adjusted the brightness to 0.2, and increased contrast and hue of colors by 0.2 and 0.1 respectively. We also rotated images randomly between 0 to 15 degrees. This part is also not used under data preparation but is added to demonstrate the actual implementations.

```
# Define augmentation transformations
augmentations = T.Compose([
    T.RandomHorizontalFlip(p=0.5),
    T.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    T.RandomRotation(degrees=10)
])

# Function to apply augmentations
def apply_augmentations(img):  1 usage
    return augmentations(img)
```

**Figure 16: Augmentation code snippet**

The bounding boxes are introduced and designed in this step with empty data blocks for given det function, and details of the points. Each image is saved with the marks of blue or red if the classification is detected.

```python
# Open image to draw bounding boxes
draw = ImageDraw.Draw(img)
img_width, img_height = img.size

yolo_labels = []

# Draw and process Faster R-CNN detections
for det in airplane_detections:
    x1, y1, x2, y2 = det
    x_center = (x1 + x2) / 2 / img_width
    y_center = (y1 + y2) / 2 / img_height
    width = (x2 - x1) / img_width
    height = (y2 - y1) / img_height

    draw.rectangle( xy: [x1, y1, x2, y2], outline="blue", width=2)
    yolo_labels.append(f"0 {x_center} {y_center} {width} {height}\n")

# Draw ground truth boxes
if image_id in annotations_dict:
    for annotation in annotations_dict[image_id]:
        bbox = annotation['bbox']
        x_min, y_min, width, height = bbox
        x_max = x_min + width
        y_max = y_min + height

        draw.rectangle( xy: [x_min, y_min, x_max, y_max], outline="red", width=2)

# Save the image with bounding boxes
img.save(dst_image_path)
```

**Figure 17: Model1 image processing, annotating, drawing detected area, and distributing the corresponding folders code snippet (part 2)**

## CHAPTER THREE: MODEL TRAINING AND EVALUATION

Due to the technical difficulties, the training and evaluation will be done on the new environment with renewed codes. Hypothetical (the code not compiling anymore) code will be defined. After separating directories for supervised learning of known labels. The model faster R-CNN Resnet50 pre-trained model is called for training process for defined two categories.

```
import os
import json
import shutil
from PIL import Image
import torch
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.transforms import functional as F
import torchvision.transforms as T
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim

# Load Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.train()  # Set the model to training mode

# Define desired categories for filtering
categories = ['airplane', 'aircraft']
```

**Figure 18: Model dependencies are described and model (faster R-CNN) is called**

The COCO transfer learning, that is introduced for the enhanced training process. We aimed to fine-tune the pre-trained models for this dataset for our usage for environmental monitoring purposes. This defined class is to load and process our dataset with filtering the categories, widening our dataset with augmentations, and handling bounding boxes.

```python
# Custom dataset class with validation for bounding boxes
class SkyFusionDataset(Dataset):  3 usages
    def __init__(self, annotation_file, image_dir, transform=None, augment=False):
        self.image_dir = image_dir
        self.transform = transform
        self.augment = augment

        # Load COCO annotations
        with open(annotation_file) as f:
            coco_data = json.load(f)

        self.images = {img['id']: img for img in coco_data['images']}
        self.annotations = coco_data['annotations']
        self.categories = {airplane['id']: airplane['name'] for airplane in coco_data['categories']}

        # Filter for desired categories
        self.airplane_category_id = [airplane['id'] for airplane in coco_data['categories'] if airplane['name'].lower() in categories]
        self.filtered_annotations = [ann for ann in self.annotations if ann['category_id'] in self.airplane_category_id]

        # Remove annotations with missing images
        self.filtered_annotations = [
            ann for ann in self.filtered_annotations if
            os.path.exists(os.path.join(self.image_dir, self.images[ann['image_id']]['file_name']))
        ]
```

**Figure 19: Dataset applications are defined in this code snippet**

In order to get the lengths of filtered annotations, __len__ class was overwritten, since this is an inherited class, we tried to eliminate as many problem as we could.

```python
    def __len__(self):
        return len(self.filtered_annotations)

    def __getitem__(self,idx):
        annotation = self.filtered_annotations[idx]
        img_id = annotation['image_id']
        img_info = self.images[img_id]
        img_path = os.path.join(self.image_dir,img_info['file_name'])

        # Load the image if it exists
        try:
            image = Image.open(img_path).convert("RGB")
        except FileNotFoundError:
            print(f"Warning: Image file {img_info['file_name']} not found, skipping.")
            return None,None  # or raise an exception if desired

        # Apply transformations
        if self.transform:
            image = self.transform(image)

        # Format bounding box data
        box = annotation['bbox']
        target = {
            "boxes": torch.tensor( data: [box],dtype=torch.float32),
            "labels": torch.tensor( data: [1],dtype=torch.int64),  # 1 for airplane
        }

        return F.to_tensor(image),target
```

**Figure 20: Definitions of custom functions for data information and augmenting**

DataLoader is for reaching the dataframe to turn them into tuples. Optimizer is the hyper parameter tuning with stochastic gradient descent (SGD) with defined parameters.

```
    # Data loader modified to skip None entries
    def collate_fn(batch):
        batch = [item for item in batch if item[0] is not None]  # Skip None entries
        return tuple(zip(*batch))
```

**Figure 21: Data skipping process code**

```
# Define augmentations (only for training data)
augmentations = T.Compose([
    T.RandomHorizontalFlip(p=0.5),
    T.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    T.RandomRotation(degrees=10)
])

# Initialize datasets and data loaders
train_dataset = SkyFusionDataset(
    annotation_file: 'SkyFusion/train/_annotations.coco.json',
    image_dir: 'SkyFusion/train/',
    transform=augmentations,
    augment=True  # Enable augmentations for training data
)
# Initialize datasets and data loaders

train_loader = DataLoader(train_dataset,batch_size=4,shuffle=True,collate_fn=collate_fn)
```

**Figure 22: Augmentation and training of the dataset virtual enlargement with the augmentation**

The model training for 10 epochs is defined considering the loss function. We assigned the total_loss for computing the model's loss in each batch, then back-propagating and updating the model weights. Previously defined functions, integrated for image-wise application.

```
# Define optimizer and hyperparameters
optimizer = torch.optim.SGD(model.parameters(),lr=0.005,momentum=0.9,weight_decay=0.0005)
num_epochs = 10

# Initialize datasets and data loaders

# Training loop
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0
    for images,targets in tqdm(train_loader,desc=f"Epoch {epoch + 1}/{num_epochs}"):
        images = list(image for image in images)
        targets = [{k: v for k,v in t.items()} for t in targets]

        optimizer.zero_grad()
        loss_dict = model(images,targets)
        losses = sum(loss for loss in loss_dict.values())
        losses.backward()

        optimizer.step()
        epoch_loss += losses.item()

    print(f"Epoch {epoch + 1}, Loss: {epoch_loss / len(train_loader):.4f}")
```

**Figure 23: Training the model**

User defined precision, recall, and f1-score to check the overall performance of the model. These actual and predicted variables were defined in the beginning of the code block.

```python
# Evaluation function
def evaluate_model(data_loader):  1 usage
    model.eval()
    total_true_positives = 0
    total_false_positives = 0
    total_false_negatives = 0

    with torch.no_grad():
        for images,targets in tqdm(data_loader,desc="Evaluating"):
            images = list(image for image in images)
            targets = [{k: v for k,v in t.items()} for t in targets]
            predictions = model(images)

            # Evaluate predictions (assumes one object per image)
            for i in range(len(images)):
                gt_boxes = targets[i]["boxes"]
                pred_boxes = predictions[i]["boxes"]

                if pred_boxes.shape[0] > 0:
                    # Assume True Positive if IoU >= 0.5
                    iou = calculate_iou(pred_boxes[0].tolist(),gt_boxes[0].tolist())
                    if iou >= 0.5:
                        total_true_positives += 1
                    else:
                        total_false_positives += 1
                else:
                    total_false_negatives += 1

    precision = total_true_positives / (total_true_positives + total_false_positives) \
        if ( total_true_positives + total_false_positives) > 0 else 0
    recall = total_true_positives / (total_true_positives + total_false_negatives) \
        if ( total_true_positives + total_false_negatives) > 0 else 0
    f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    print(f"Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1_score:.2f}")
    return precision,recall,f1_score
```

**Figure 24: User defined function for evaluating pictures in a loop for overall accumulation**

YOLOv8n is a nano version of the model. Due to incapability, this is chosen for evaluation as it is the best one for our problem.

Exposure (COCO) | Detection (Open Images V7) | Segmentation (COCO) | Classification (ImageNet) | Pose (COCO) | OBB (DOTAv1)

In the Detection Docs you will find application examples with these models, which were trained on COCO and contain 80 pre-trained classes.

| Model | Size (Pixels) | mAPval 50-95 | Speed CPU ONNX (ms) | Speed TensorRT A100 (ms) | params (M) | FLOPs (B) |
|---|---|---|---|---|---|---|
| YOLOv8n | 640 | 37.3 | 80.4 | 0.99 | 3.2 | 8.7 |
| YOLOv8s | 640 | 44.9 | 128.4 | 1.20 | 11.2 | 28.6 |
| YOLOv8m | 640 | 50.2 | 234.7 | 1.83 | 25.9 | 78.9 |
| YOLOv8l | 640 | 52.9 | 375.2 | 2.39 | 43.7 | 165.2 |
| YOLOv8x | 640 | 53.9 | 479.1 | 3.53 | 68.2 | 257.8 |

**Figure 25: YOLO model comparison (Ultralytics, 2024)**

The yaml document is defined for simple version. The needed information where the images stored and the categories that are going to be detected.

```yaml
path: ./SkyFusion
train: train/images
val: valid/images
test: test/images

nc: 2  # number of classes
names: ['airplane', 'aircraft']
```

**Figure 26: Definitions of the directories and related classes**

The packages is defined and constructed with necessary imports.

```python
from ultralytics import YOLO

# Load the model
model = YOLO("yolov8n.pt")

# Train the model
model.train(data="data.yaml", epochs=10, imgsz=416, conf=0.5, max_det=300, device='cpu')
```

**Figure 27: Model of YOLOv8n training information details**

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/10        0G      1.777       2.09      1.149         11        416: 100%|██████████| 131/131 [04:28<00:00,  2.05s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:25<00:00,  1.69s/it]
             all        449       1957      0.992      0.298      0.645      0.468

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/10        0G      1.51       1.227      1.037         52        416: 100%|██████████| 131/131 [04:24<00:00,  2.02s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:25<00:00,  1.71s/it]
             all        449       1957       0.88      0.819      0.887      0.553

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/10        0G      1.488      1.169      1.039         81        416: 100%|██████████| 131/131 [04:25<00:00,  2.03s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:25<00:00,  1.67s/it]
             all        449       1957      0.918       0.86      0.914       0.58

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 4/10        0G      1.454      1.143       1.02         35        416: 100%|██████████| 131/131 [04:23<00:00,  2.01s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:24<00:00,  1.65s/it]
             all        449       1957      0.954      0.816       0.89      0.575

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 5/10        0G      1.391       0.83       1.01         22        416: 100%|██████████| 131/131 [04:27<00:00,  2.04s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:25<00:00,  1.68s/it]
             all        449       1957      0.967      0.826      0.901      0.585
```

**Figure 28:The result of epochs by YOLOv8n (part 1)**

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 6/10        0G      1.372     0.7879      1.001         66        416: 100%|██████████| 131/131 [04:32<00:00,  2.08s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:26<00:00,  1.78s/it]
             all        449       1957      0.966      0.866      0.921       0.61

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 7/10        0G      1.329     0.7054      1.001        119        416: 100%|██████████| 131/131 [04:29<00:00,  2.05s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:24<00:00,  1.66s/it]
             all        449       1957      0.961      0.881       0.93      0.631

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 8/10        0G      1.294     0.6635     0.9789         18        416: 100%|██████████| 131/131 [04:20<00:00,  1.99s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:24<00:00,  1.66s/it]
             all        449       1957       0.97      0.881      0.932      0.645

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 9/10        0G      1.275     0.6542     0.9827          5        416: 100%|██████████| 131/131 [04:22<00:00,  2.01s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:24<00:00,  1.66s/it]
             all        449       1957      0.959      0.906      0.945      0.649

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
10/10        0G      1.253     0.6346     0.9595         47        416: 100%|██████████| 131/131 [04:19<00:00,  1.98s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:24<00:00,  1.66s/it]
             all        449       1957      0.958      0.911      0.947      0.663
```

**Figure 29: The result of epochs by YOLOv8n (part 2)**

```
10 epochs completed in 0.810 hours.
Optimizer stripped from runs\detect\train6\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train6\weights\best.pt, 6.2MB

Validating runs\detect\train6\weights\best.pt...
Ultralytics 8.3.26  Python-3.11.1 torch-2.5.1+cpu CPU (Intel Core(TM) i7-8750H 2.20GHz)
Model summary (fused): 168 layers, 3,006,038 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 15/15 [00:20<00:00,  1.36s/it]
                   all        449       1957      0.958      0.911      0.947      0.663
              airplane        149       1957      0.958      0.911      0.947      0.663
Speed: 0.8ms preprocess, 33.9ms inference, 0.0ms loss, 0.2ms postprocess per image
Results saved to runs\detect\train6
```

**Figure 30: Evaluation metrics that are defined by the model.**

We can see the high precision (0.958) and high recall (0.911) we can conclude that even though the version is a nano version, it can be suitable for less computation-powered devices, and the prediction rates are more than the expected values.

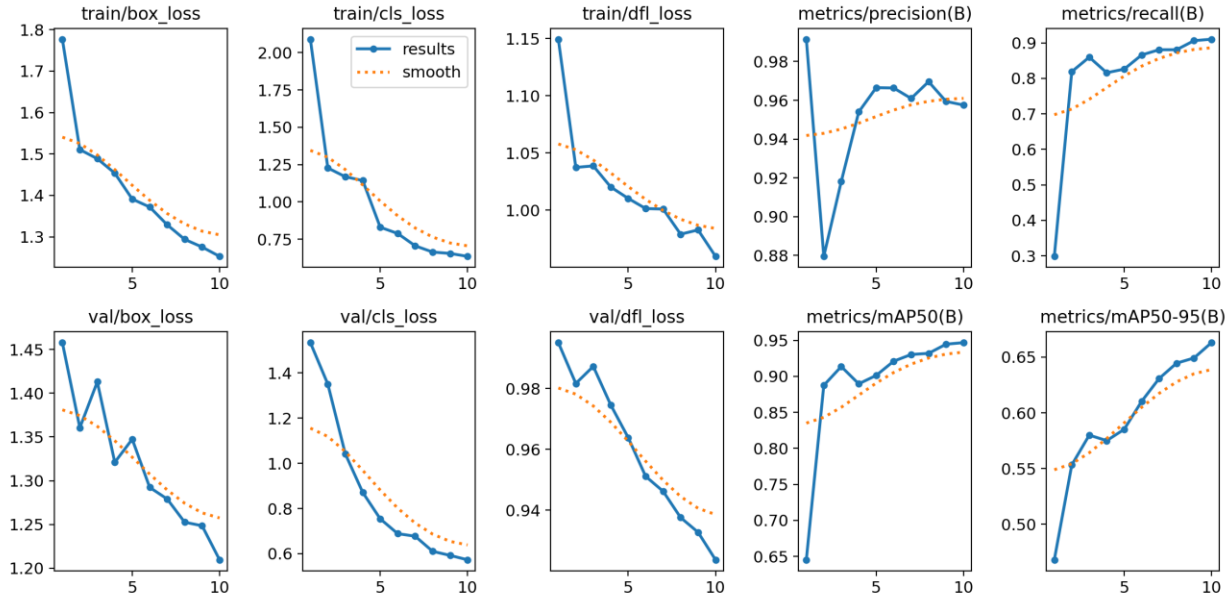


**Figure 31: Performance metrics comparison for each epoch**

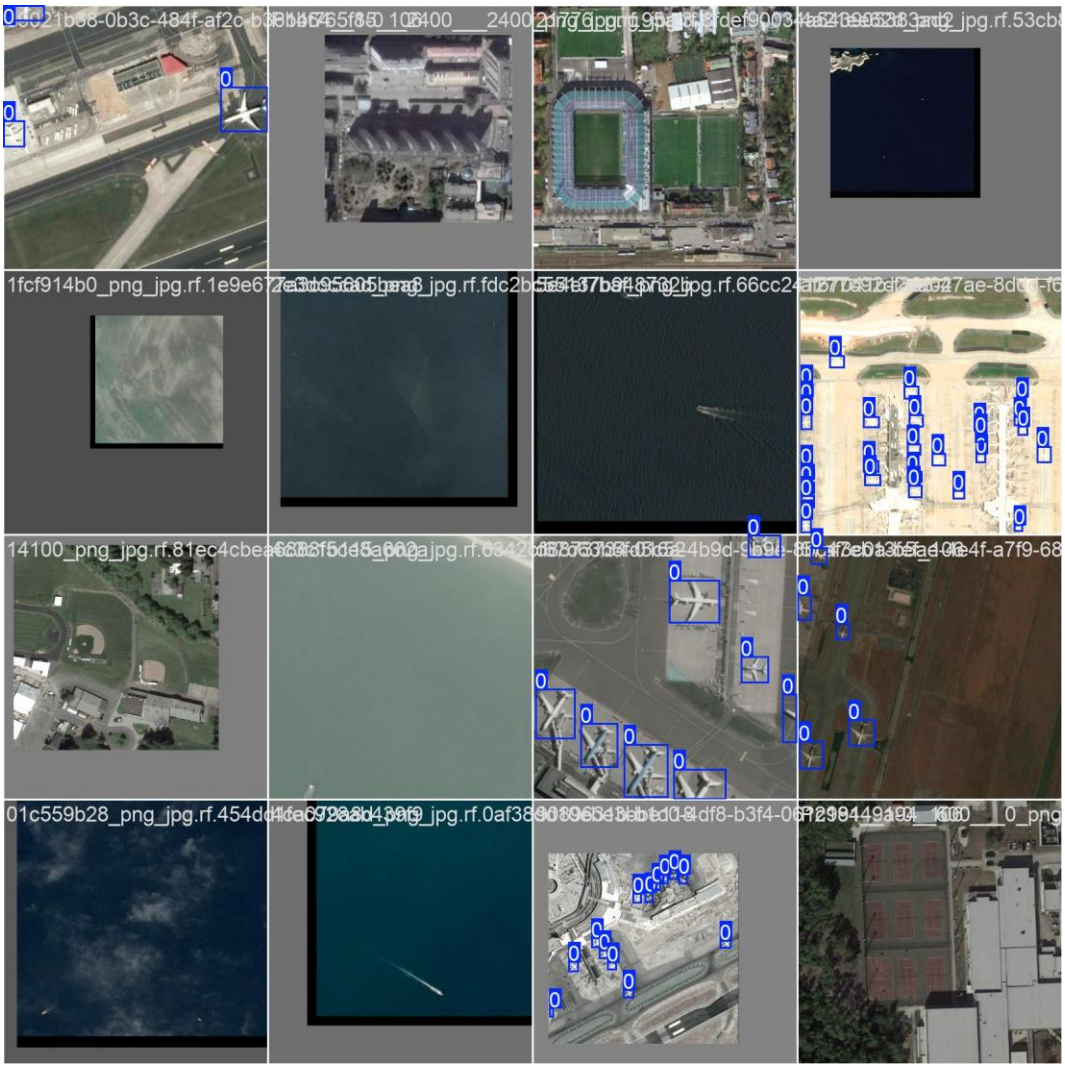

**Figure 32: Training batch samples for YOLOv8n model**

**Figure 33: Labeled outputs of training set for YOLOv8n model**

After eye comparison for some images, we weren't expecting to find an aircraft, there it is visible in the confusion matrix that no aircraft, stands in the middle line) detected, trained, or learned.

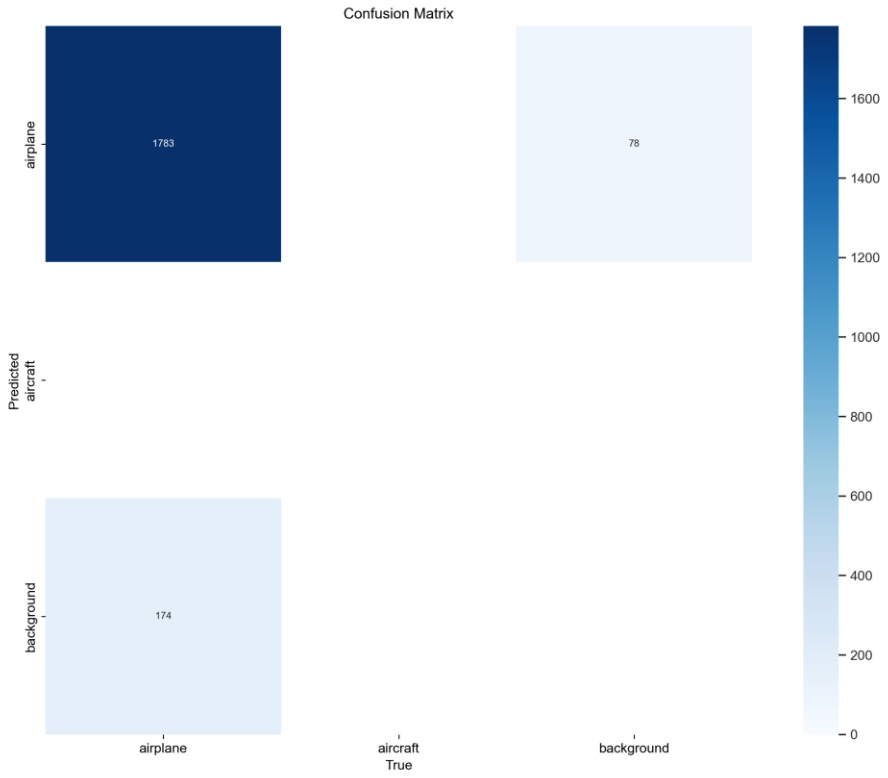


**Figure 34: Confusion matrix for the predictions in YOLOv8n model**

Since the information is not healthy to make a comparison, we will be just showing the decreasing rates form training set to validation set.

```
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-258fb6c6.pth" to C:\Users\x/
 .cache\torch\hub\checkpoints\fasterrcnn_resnet50_fpn_coco-258fb6c6.pth
100%|██████████| 160M/160M [00:42<00:00, 3.97MB/s]
Processed annotations for train
Precision: 0.40, Recall: 0.08, F1 Score: 0.13
Total true positives: 695
Total false negatives: 8001
Total false positives: 1058
Processed annotations for valid
Precision: 0.37, Recall: 0.06, F1 Score: 0.10
Total true positives: 118
Total false negatives: 1839
Total false positives: 200
Processed annotations for test
Precision: 0.41, Recall: 0.07, F1 Score: 0.12
Total true positives: 138
Total false negatives: 1830
Total false positives: 201
faster_R-CNN dataset compressed and saved as faster_R-CNN_dataset.zip
```

**Figure 35: Faster R-CNN 1 epoch results**

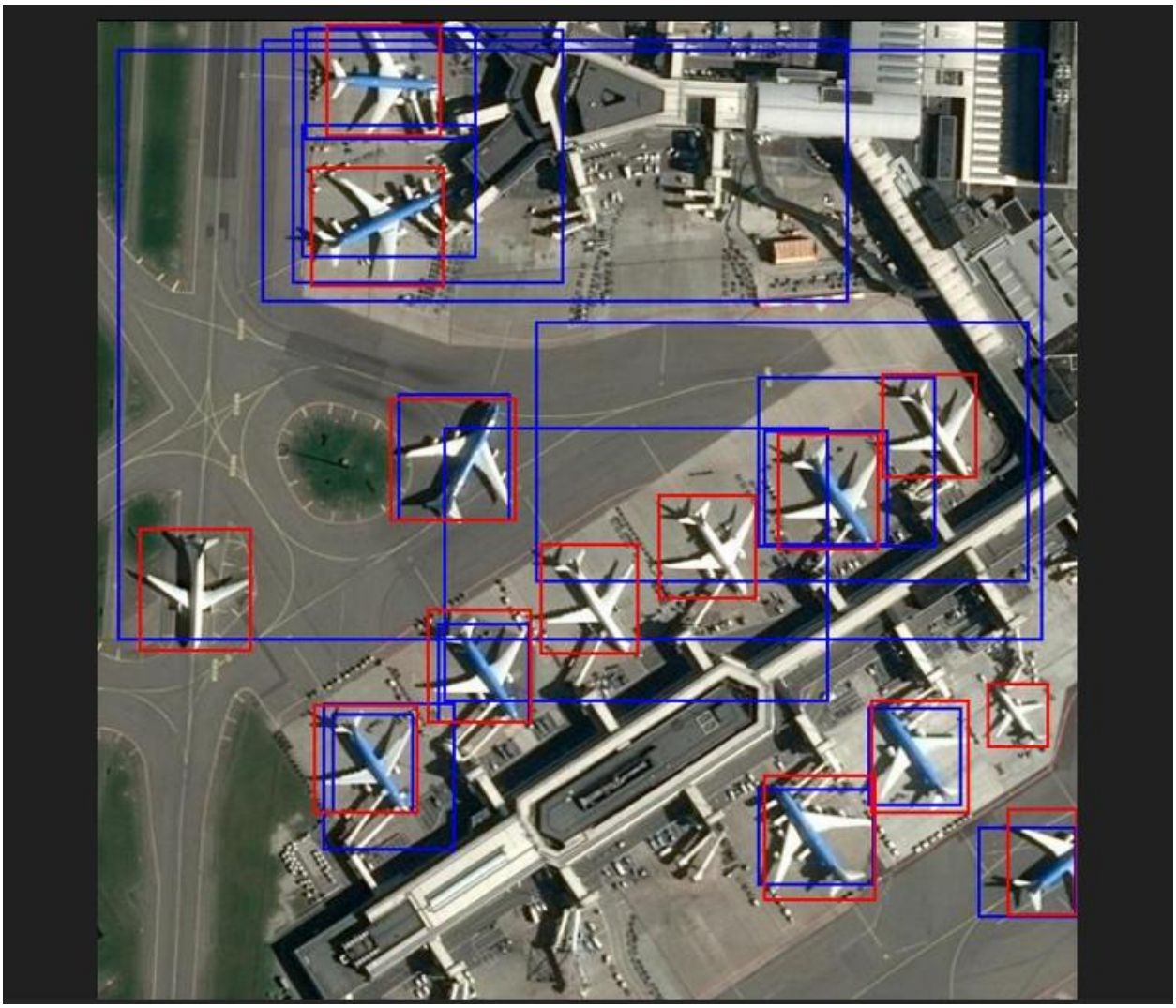

**Figure 36: Model1 annotations with predictions outcome**

## CHAPTER FOUR: PRACTICAL APPLICATION

Using YOLO and faster R-CNN with aircraft detection for environmental monitoring can help us with aerial traffic patterns and natural interactions for authorized or unauthorized activity detection. In the real world, these models can help us to surveillance their existence in remote areas. For instance, if a plane landed in a natural habitat, we can detect it and announce it to authorities to improve security.

Another problem can be wildlife encounters with aerial traffic that can be harmful both for the animals and commuting resources with human lives. Also rescuing operations can be assisted with real-time detections in case of an emergency or disaster situations. Integration of drones with IoT devices can enhance data collection or image retrieval processes for widening the detection possibilities.

Both models have trade-offs like accuracy and time dependence. The choice can be made to the environmental conditions for balancing the response time and optimizing the efficiency, or they can both be used for diminishing these trade-off factors. Their combination can improve the environmental monitoring for security measures.


**Figure 37: Predicted aircraft gallery**

## CONCLUDING REMARKS AND FUTURE SCOPE

The increased usage of advanced object detection techniques like Faster R-CNN and YOLO in environmental monitoring increases our abilities to detect anomalies and unwanted situations to the traditional techniques. With the help of Faster R-CNN, we can increase the accuracy, while YOLO will provide us with fast computations. As the need for these applications increase the more effective models will be evolved. Observing, conserving, managing resources, and understanding ecological factors will scale the models' effectiveness to a higher level.

Despite the technical difficulties, YOLOv8n achieved impressive results for a small package without hyper-parameter tuning or augmenting the dataset beforehand with precision at 0.958 and recall at 0.911. This is also an indicator of how simple deep learning and artificial intelligence applications are evolving into. We first compiled the code to obtain a general idea of the collected data. Then we diverged the directories into labeled systems. Prepared the data with augmentation and optimized our datasets for accurate model training and evaluation. Even though the compile errors of the previously working codes, our prototype of 1 epoch also showed us the working principles of the Faster R-CNN model.

Further improvements can be done with extending the dataset with differing weather conditions, and adding occlusions (closed up or blocked off with clouds maybe). Another usage can be detecting the lost airplanes during the monitoring period just before the network cuts off. This can help the resolution of the unexplained plane crushes even before the black boxes are found.

**BIBLIOGRAPHY**


Alvi, F. (2024) *Data annotation – A beginner's guide*, *OpenCV*. Available at: https://opencv.org/blog/data-annotation/ (Accessed: 04 November 2024).

An, K. *et al.* (2024) 'Enhancing small object detection in aerial images: A novel approach with PCSG model', *Aerospace*, 11(5), p. 392. doi:10.3390/aerospace11050392.

Envirosuite (2024) *Pitfalls of Traditional Environmental Monitoring Software*, *Envirosuite*. Available at: https://envirosuite.com/insights/news/pitfalls-of-traditional-environmental-monitoring-software (Accessed: 28 October 2024).

Gad, A.F. and Skeltonn, J. (2024) *Faster R-CNN explained for Object Detection Tasks*, *DigitalOcean*. Available at: https://www.digitalocean.com/community/tutorials/faster-r-cnn-explained-object-detection (Accessed: 04 November 2024).

Naim, H. (2024) *Aerial detection with Yolo-training*, *Kaggle*. Available at: https://www.kaggle.com/code/hazmannaim/aerial-detection-with-yolo-training/notebook (Accessed: 04 November 2024).

Nazarov, D., Sulimin, V. and Shvedov, V. (2024) 'Advancing Environmental Stewardship: The role of automation in enhanced environmental monitoring', *E3S Web of Conferences*, 542, p. 05005. doi:10.1051/e3sconf/202454205005.

Sani, S.A. *et al.* (2023) 'Drawbacks of traditional environmental monitoring systems', *Computer and Information Science*, 16(3), p. 30. doi:10.5539/cis.v16n3p30.

Sudheer, K.P. (2023) *Skyfusion: Aerial Object Detection*, *Kaggle*. Available at: https://www.kaggle.com/datasets/kailaspsudheer/tiny-object-detection (Accessed: 04 November 2024).

Ultralytics (2024) *Yolov8, YOLOv8 - Ultralytics YOLO Docs*. Available at: https://docs.ultralytics.com/de/models/yolov8/#supported-tasks-and-modes (Accessed: 04 November 2024).

Zainurin, S.N. *et al.* (2022) 'Advancements in monitoring water quality based on various sensing methods: A systematic review', *International Journal of Environmental Research and Public Health*, 19(21), p. 14080. doi:10.3390/ijerph192114080.

## *Table of Figures*

# APPENDIX

My codes are uploaded to this repository in Github:

https://github.com/3irsari/Computer-Vision_Image-Segmentation

These are the code and dataset links that are used during the preparation process:

https://www.kaggle.com/code/arpandhatt/satellite-image-classification/notebook

https://www.kaggle.com/discussions/general/333141

https://cocodataset.org/#format-data

https://www.kaggle.com/code/moodywriter/aerial-detection-with-yolo

https://www.kaggle.com/datasets/airbusgeo/airbus-aircrafts-sample-dataset

https://www.kaggle.com/models/umeradnaan/yolo-v8

https://www.kaggle.com/code/zidane10aa/airplanes-extraction/input