PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

SAAD DAHLEB BLIDA 01 UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

MASTER’S INTELLIGENT SYSTEMS ENGINEERING

COMPUTER VISION

REPORT

FASTER RCNN MODEL IMPLEMENTATION FOR FIRE DETECTION TASK

Implemented and written by

Abdelatif Mekri

Halima Nfidsa

Imad Eddine Boukader

Nahla Yasmine Mihoubi

Academic year : 2024-2025

Content Table

Table Of figures

Table Of tables

1. Data Collection and Processing

1.1- Dataset Creation

In this phase we opted to merge several datasets found on the roboflow website , since the size of each of them was small , and to insure that we get the best results possible using our model the first step was to merge them into one big dataset , we have used the following datasets :

Dataset Name	Link	Preprocessing	Augmentations	Image Size	Distribution	Additional Notes
Wildfire N12RK	https://universe.roboflow.com/testes-jnrhu/wildfire-n12rk/dataset/1	Auto-Orient, Resize	None	640x640	train 350 test 75 validation 75	Basic preprocessing, no augmentations.
Wildfire Satellite	https://universe.roboflow.com/bairock/wildfire-satellite/dataset/2	Auto-Orient, Resize	Horizontal Flip, Vertical Flip	416x416	train 8828 valid 200 test -	Each training example produces 3 augmented outputs.
Wildfire YH86L	https://universe.roboflow.com/mini-z0ruz/wildfire-yh86l/dataset/2	Auto-Orient, Resize	None	640x640	train 3206 valid 174 test 609	Basic preprocessing, no augmentations.
Fire Forest New	https://universe.roboflow.com/fireforestgen/fire_forest_new/dataset/8	None	None	N/A	train 413 valid 105 test -	No preprocessing or augmentations.
Fire Forest Gen		Resize	None	800x800	train 421 valid 72	Stretch resizing applied.

https://universe.roboflow.com/fireforestgen/fire_forest_gen/dataset/8				test -	
12345-su5ee https://universe.roboflow.com/12345-su5ee/-c956b/dataset/1	Auto-Orient, Resize	None	640x640	train 80 valid 23 test 11	Stretch resizing applied
640x640 XDhu3 https://universe.roboflow.com/davidturati/640x640-xdhu3/dataset/24#	Auto-Orient, Resize	Multiple: Flipping, Rotation, Cropping, Grayscale, Hue, Saturation, Brightness, Exposure, Noise, Cutout	640x640	train 3117 valid 296 test 150	Extensive augmentations applied.
Fire Detection2 Wayva https://universe.roboflow.com/pfc-dshky/fire-detection2-wayva/dataset/1	Auto-Orient, Resize	Horizontal Flip, Vertical Flip, 90° Rotation	640x640	train 1953 valid 579 test 356	Moderate augmentations for diverse transformations
DLR https://universe.roboflow.com/htw-berlin-xv7eo/dlr/dataset/1	Auto-Orient, Resize	Multiple: Rotation, Cropping, Hue, Saturation, Exposure, Blur, Noise	1000x1000	train 1044 valid 99 test 50	Large size; augmentations enhance variety.
Volcano NZTT3 https://universe.roboflow.com/di-gao/volcano-nztt3/dataset/19	Auto-Orient, Resize	Multiple: Flipping, Rotation, Blur, Noise	416x416	train 237 valid 39 test 27	Augmentations include noise and blurring.
Wildfire Detection Satellite	Auto-Orient, Resize	Horizontal Flip, Vertical Flip,	640x640	train 23064 valid 5948	Includes tiling preprocessing.

https://universe.roboflow.com/umbc-s6g8c/wildfire-detection-satellite/dataset/3		Rotation, Saturation		test 1468	

Table 1 : comparative table of the used datasets and their details

1.2-

A- Reading Datasets

1. Importing Roboflow Library

This imports the Roboflow class from the roboflow library. This library allows interaction programmatically with datasets and models hosted on the Roboflow platform.

2. Authenticating with the API

Here, instantiating the Roboflow class with a unique API key. The API key authenticates access to the Roboflow platform and grants permission to interact with the datasets and models associated with the account.

3. Selecting the Workspace and Project

This line specifies the workspace (USER_X) and the specific project (PROJECT_123) that we want to work with. The workspace method allows selecting a project workspace, and the project specifies which project within that workspace we are accessing.

4. Selecting the Dataset Version

You are selecting version X of the project (PROJECT_123). Roboflow supports versioning of datasets, so we can choose specific versions to ensure consistency in our work.

5. Downloading the Dataset in YOLOv8-OBB Format

Finally, this downloads the dataset in the YOLOv8 OBB (Oriented Bounding Box) format. The download() method provides the dataset in the format you specify (e.g., yolov8-obb, yolov5, etc.), which is tailored for a chosen object detection model. The dataset is returned as an object, typically including images and annotations in the format we requested.

Example :

```
from roboflow import Roboflow
rf = Roboflow(api_key="XXXXXXXXXXXXXXXXXXXX")
```

```
project = rf.workspace("di-gao").project("volcano-nztt3")
version = project.version(19)
dataset = version.download("yolov8-obb")
```

B- Making sure the datasets loaded correctly

```
base_path = '/kaggle/working'
folders = [
    "640x640-24", "DLR-1", "Wildfire-2", "Wildfire-Detection-Satellite-3",
    "Wildfire-Satellite-2", "fire-detection2-1", "fire_forest_gen-8",
    "fire_forest_new-8", "volcano-19", "wildfire-1", "Хакатон-Пожары-1"
]

# Check the folder structure of each dataset
for folder in folders:
    dataset_path = os.path.join(base_path, folder)
    print(f"Contents of {folder}:")
    print(os.listdir(dataset_path))
    print("-" * 50)
```

Example :

```
--------------------------------------------------
Contents of Хакатон-Пожары-1:
['test', 'README.roboflow.txt', 'train', 'README.dataset.txt', 'valid']
--------------------------------------------------
```

C- Merging

```
for folder in folders:
    dataset_path = os.path.join(base_path, folder)

    for split in ['train', 'valid', 'test']:
        split_images = os.path.join(dataset_path, split, 'images')
        split_labels = os.path.join(dataset_path, split, 'labels')

        merge_files(split_images, combined_paths[split]['images'], folder)
        merge_files(split_labels, combined_paths[split]['labels'], folder)
```

The result of this merge was a one dataset that has the following :

train 80% valid : 15% test : 5%

Train Folder	Valid Folder	Test Folder
- Images folder contains 42705 files. - Labels folder contains 42705 files.	- Images folder contains 7610 files. - Labels folder contains 7610 files.	- Images folder contains 2746 files. - Labels folder contains 2746 files.

1.3- Data Preprocessing

Data preprocessing is a crucial step in ensuring high-quality inputs for the model, improving accuracy, and reducing noise. The following steps were applied to refine the dataset before training the Faster R-CNN model.

A.Preprocessing Steps

1. **Data Cleaning:**
    - Removed blurry, distorted, or low-quality images that could negatively impact model performance.
    - Eliminated duplicate images to avoid redundancy in the dataset.
    - Checked for missing or corrupted files and replaced them if necessary.
2. **Image Resizing:**
    - Standardized all images to a fixed resolution to ensure consistency across different data sources.
    - Maintained the aspect ratio where possible to prevent distortion.
3. **Data Normalization:**
    - Scaled pixel values between 0 and 255 to ensure a uniform data distribution.
    - Normalization improves model convergence and stabilizes training by standardizing input features.
4. **Dataset Augmentation (Not Applied):**
    - While data augmentation can enhance model generalization, it was not implemented in this case.
    - The decision was made due to the high computational demands of Faster R-CNN and memory limitations, which could lead to inefficient processing.

Note: Despite not applying augmentation, other preprocessing techniques were optimized to ensure high-quality input for training.

B.Visualization of Transformed Data

Visualizing the data before and after preprocessing helps assess the effectiveness of transformations. Below are examples illustrating the impact of preprocessing:

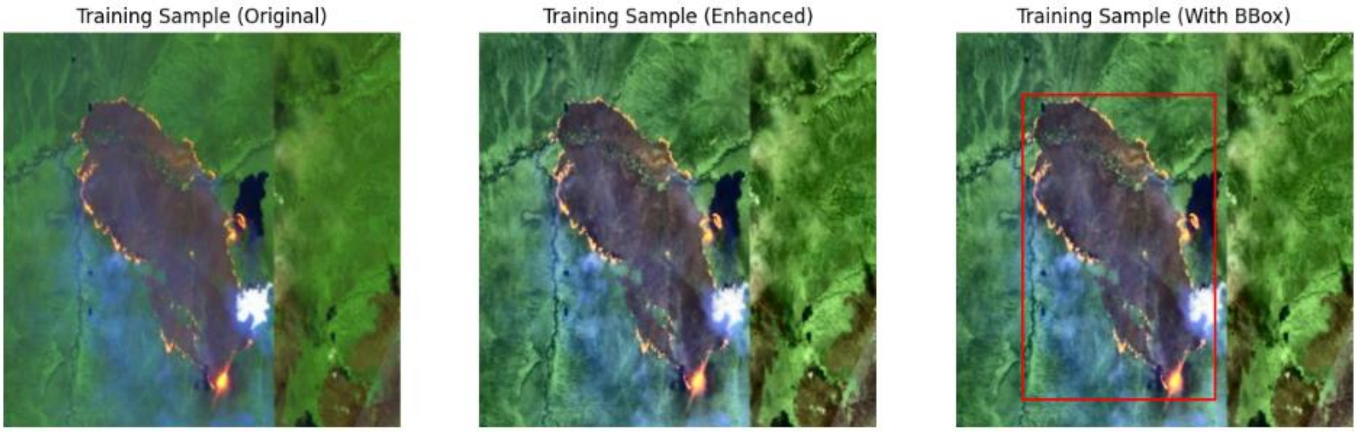


Figure 1 : Comparison between original image and enhanced image using histogram equalization for global contrast improvement.

1.4-Data reduction

We reduced the data in the model due to memory limitations in the Kaggle environment. The available memory was insufficient to handle large amounts of data, leading to performance issues and potential crashes. To ensure the model could function properly, we opted to reduce the dataset size, balancing efficiency and accuracy while working within resource constraints. This resulted in a 54.5% reduction in data usage, allowing for smoother model training and execution.
As a result we got the following :

Train Folder:
- Images folder contains 19407 files.
- Labels folder contains 19407 files.

Valid Folder:
- Images folder contains 1623 files.
- Labels folder contains 1623 files.

Test Folder:
- Images folder contains 1251 files.
- Labels folder contains 1251 files.

1.5-Dataset Splitting

To train and evaluate the model effectively, the dataset was divided into three subsets, ensuring class distribution balance and preventing data leakage:

- **Training Set:** [≈ 87.1%] of the dataset

- Used for learning model parameters and feature extraction.
    - Contains a diverse set of images to improve generalization.
- **Validation Set:** [≈ 7.3%] of the dataset
    - Used to fine-tune hyperparameters and avoid overfitting.
    - Helps assess intermediate model performance before final evaluation.
- **Test Set:**[≈ 5.6%] of the dataset
    - Reserved exclusively for the final evaluation of the model.
    - Ensures that the model's accuracy reflects real-world performance.

Importantly, we did not perform any manual splitting of the original dataset. The dataset had its own predefined splits, and when we merged them, we maintained the original structure. This meant that the final merged dataset had three folders (train, valid, and test), each being a union of the corresponding parent folders, preserving the integrity of the original splits.

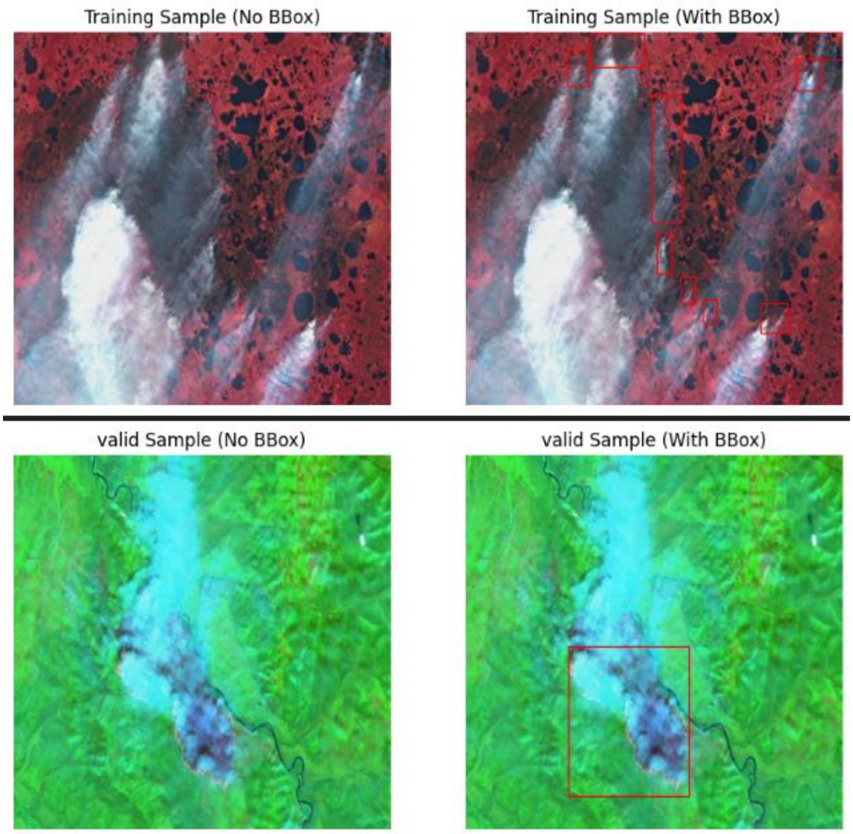


Figure 2: Showcase of few samples of the reduced dataset ,image with and without the bounding box

2. Model Presentation and Training

2.1 Introduction to Faster R-CNN

Faster R-CNN (Region-Based Convolutional Neural Network) is an evolution of R-CNN and Fast R-CNN models, significantly improving the efficiency and speed of object detection. It consists of three main components:

1. **A Backbone Network for Feature Extraction**
    - Typically a pre-trained CNN (such as ResNet, VGG, or MobileNet) that transforms the input image into a **feature map**.
2. **The Region Proposal Network (RPN)**
    - A specialized network that generates **regions of interest (RoIs)** likely to contain objects.
    - Replaces the selective search approach used in earlier R-CNN versions, making the model much faster.
3. **The CNN-Based Detector**
    - A layer that **refines the predictions** made by the RPN.
    - Adjusts the bounding box coordinates and classifies detected objects.

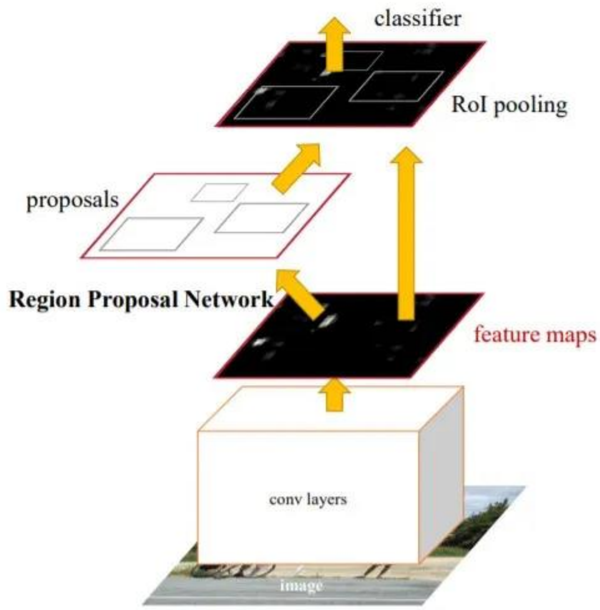


Figure 3: Simple demonstration of Faster R-CNN Architecture[1]

2.2 Faster R-CNN Architecture

The Faster R-CNN architecture model is designed for object detection, improving both the speed and accuracy of previous models like R-CNN and Fast R-CNN. Here's an overview of its architecture:

The architecture follows these key steps:

1.Input

The raw image is fed into the model.

[1] Ren, S., He, K., Girshick, R., & Sun, J. (2016). *Faster R-CNN: Towards real-time object detection with region proposal networks*. arXiv. https://arxiv.org/abs/1506.01497

2. Convolutional Backbone (Feature Extraction)

- The first component of Faster R-CNN is a convolutional neural network (CNN) used for feature extraction. This part typically uses well-known networks like VGG16 or ResNet to extract feature maps from the input image.
- These feature maps are used by subsequent components for identifying objects in the image.

3. Region Proposal Network (RPN)

- One of the key innovations of Faster R-CNN is the Region Proposal Network (RPN). The RPN is responsible for generating region proposals, which are candidate regions in the image that are likely to contain objects.
- The RPN shares convolutional layers with the rest of the network (i.e., the CNN used for feature extraction), enabling efficient generation of proposals. It slides over the feature map produced by the CNN, predicting objectness scores (whether the region contains an object) and the bounding box coordinates for each proposal.
- The RPN produces multiple anchor boxes at each spatial location on the feature map. These anchor boxes have different aspect ratios and scales to cover various object sizes. The RPN then classifies each anchor as either background or foreground and refines the bounding boxes.

4. RoI Pooling (Region of Interest Pooling)

- After the RPN generates region proposals, the next step is to extract features for these proposals using a process called Region of Interest (RoI) pooling.
- RoI pooling takes the feature map produced by the CNN and uses the region proposals (bounding boxes) to extract a fixed-size feature map from each region. The feature map for each proposal is resized to a fixed size (e.g., 7x7) to make it compatible for further processing.

5. Fully Connected Layers (Classification and Bounding Box Regression)

1. The final step in Faster R-CNN involves passing the pooled features into fully connected (FC) layers.
2. The FC layers perform two tasks:
    - **Classification**: Each region proposal is classified into different object categories (e.g., car, dog, person) or background (no object).
    - **Bounding Box Regression**: For each proposal, the bounding box is refined to better fit the object. This involves predicting the offset (or correction) for each bounding box to improve its accuracy.

6. Output

The final list of detected objects with their **category** and **bounding box**.

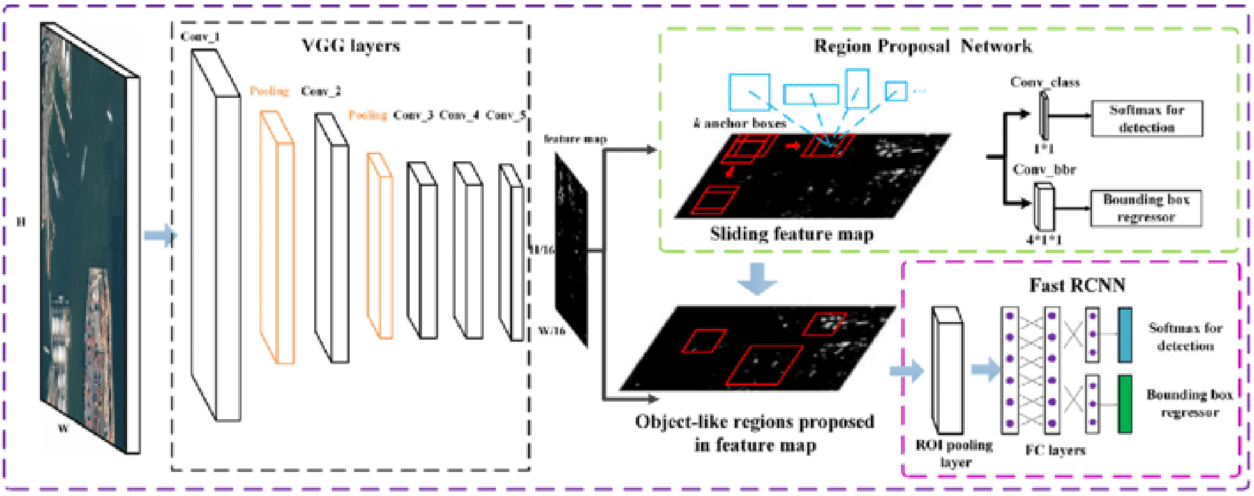


Figure 4: Detailed demonstration of Faster R-CNN Architecture[2]

4. Hyperparameter Configuration

4.1 Hyperparameters Used for Training

- **Learning Rate**: 0.001
- **Batch Size**: 16
- **Epochs**: 10
- **Loss Function**: Cross-Entropy Loss
- **Optimizer**: Adam

4.2 Hyperparameter Selection

The hyperparameters were determined using the following methods:

- **Grid Search**: Used to fine-tune the learning rate and batch size.
- **HyperTune**: ten used in **automated hyperparameter optimization** to improve model performance efficiently.

4.3 Differences Between Models

- **EfficientNet (Backbone)**: A **learning rate of 0.0005** and a **batch size of 32** were chosen to balance performance and efficiency.

[2] Deng, Z., Sun, H., Zhou, S., Zhao, J., Lei, L., & Zou, H. (2018). Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing, 145*, 1-12. https://doi.org/10.1016/j.isprsjprs.2018.04.003

- **ResNet-50 (Backbone)**: A **learning rate of 0.001** and a **batch size of 16** were selected for better stability during training.
- **ResNet-101 (Backbone)**: A **learning rate of 0.001** and a **batch size of 16** were used, similar to ResNet-50, but ResNet-101 has more layers, allowing it to capture deeper features at the cost of higher computational requirements.

Feature	**ResNet-50**	**EfficientNet-B3**	**ResNet-101**
Number of Parameters	~97.8million	~47.2 million	~171 million
Model Depth	50 layers	233 layers	101 layers
Accuracy	High (competes well with many models)	Higher accuracy for fewer parameters	very High
Inference Speed	Moderate	Faster than ResNet-50 (in many cases)	Less efficient
Computational Efficiency	Moderate	Very efficient (fewer parameters)	computationally heavier

Table 2: Comparative table of backbones (standard parameters) used for the implementation of the faster RCNN model

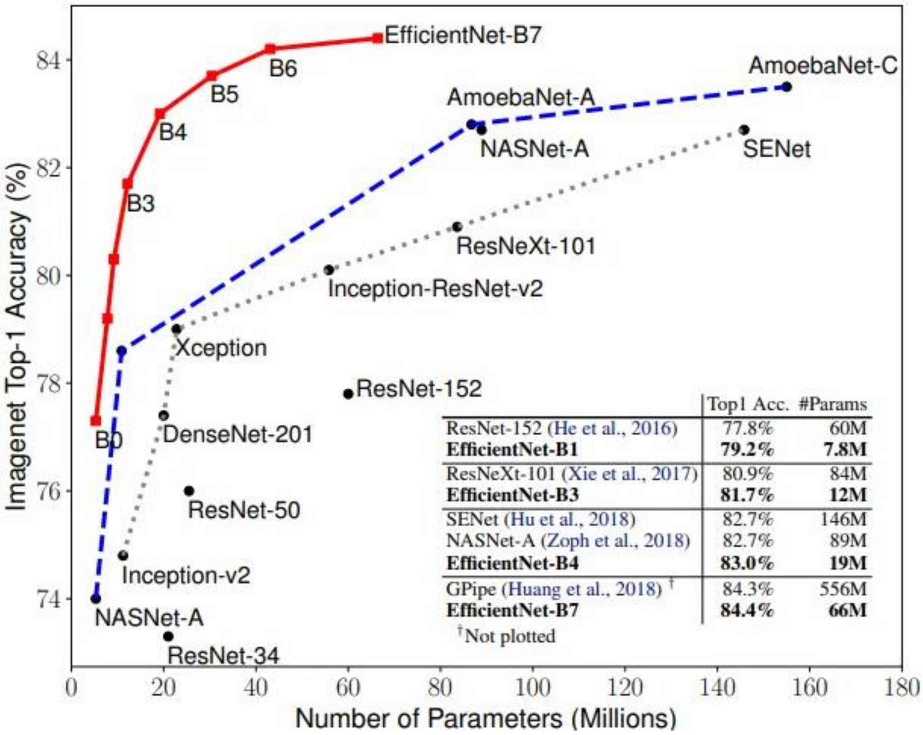


Figure 5: Performance comparison between the backbone models

3

3. Model Evaluation

3.1 Performance Metrics

The evaluation of Faster R-CNN models was conducted on the test set using the following metrics:

3.1.1 Running without augmentation - mixed datasets

EfficientNet-B3			
results	small dataset	medium dataset	large dataset
	train 1000; valid 500 ;epochs : 20	train 3000;valid 1000 ;epochs:20	train 6000;valid 2000 ;epochs:20
first epoch	Epoch 1 Summary: Loss: 13.9911 IoU: 0.0057 Precision: 0.0001 Recall: 0.0052 F1-Score: 0.0002 mAP: 0.0001	Epoch 1 Summary: Loss: 12.5332 IoU: 0.0008 Precision: 0.0000 Recall: 0.0007 F1-Score: 0.0001 mAP: 0.0000	Epoch 1 Summary: Loss: 11.1423 IoU: 0.0011 Precision: 0.0003 Recall: 0.0009 F1-Score: 0.0003 mAP: 0.000
last epoch	Epoch 20 Summary: Loss: 9.5718 IoU: 0.0023 Precision: 0.0010 Recall: 0.0019 F1-Score: 0.0013 mAP: 0.0014	Loss: 9.6040 IoU: 0.0014 Precision: 0.0014 Recall: 0.0012 F1-Score: 0.0009 mAP: 0.0017	/

[3] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv*. https://arxiv.org/pdf/1905.11946v3

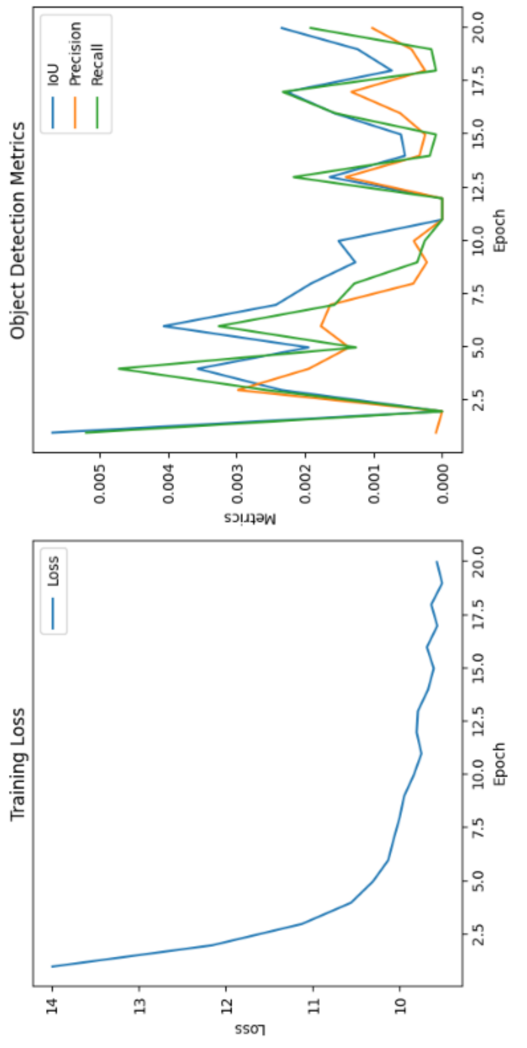
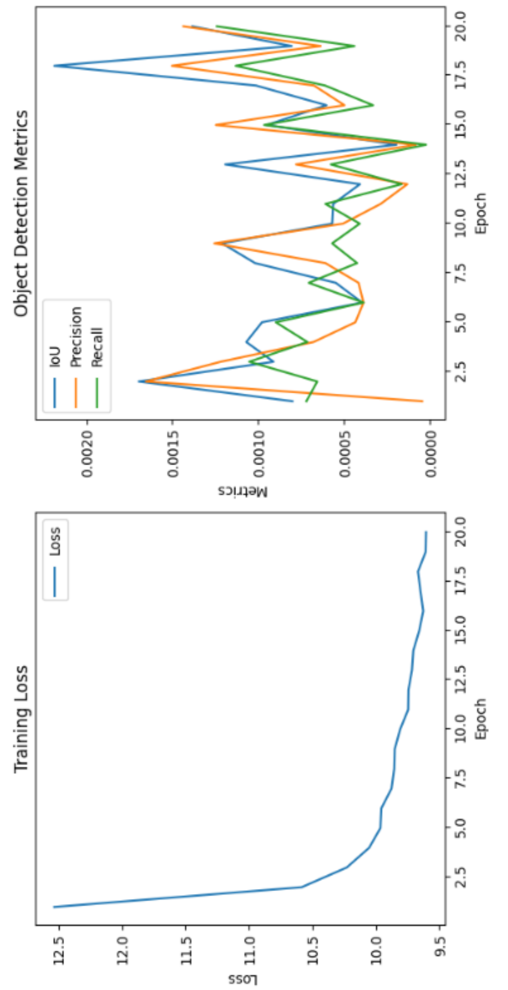
overall			/

Table 3 Comparative table of EfficientNet-B3 Backbone with a subset of the mixed Dataset

Res-Net 101			
results	medium dataset	large dataset	X-Large dataset
	Train: 1000 Valid: 500 epochs: 20	Train: 3000 Valid: 1000 epochs: 35	Train: 6000 Valid: 3000 epochs: 40
first epoch	Epoch 1 Summary: Loss: 11.2846 IoU: 0.0006 Precision: 0.0000 Recall: 0.0010 F1-Score: 0.0000 mAP: 0.0000	Epoch 1 Summary: Loss: 11.0472 IoU: 0.0000 Precision: 0.0000 Recall: 0.0000 F1-Score: 0.0000 mAP: 0.0000	Epoch 1 Summary: Loss: 10.0129 IoU: 0.0002 Precision: 0.0001 Recall: 0.0001 F1-Score: 0.0001 mAP: 0.0001

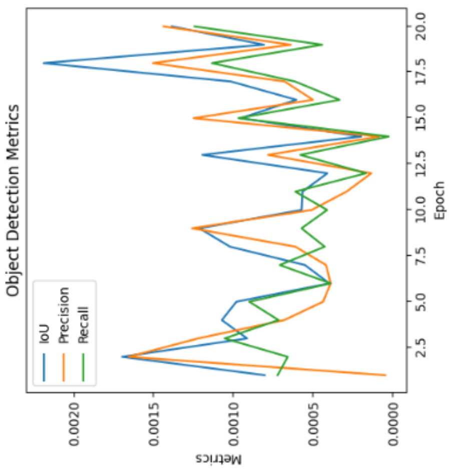
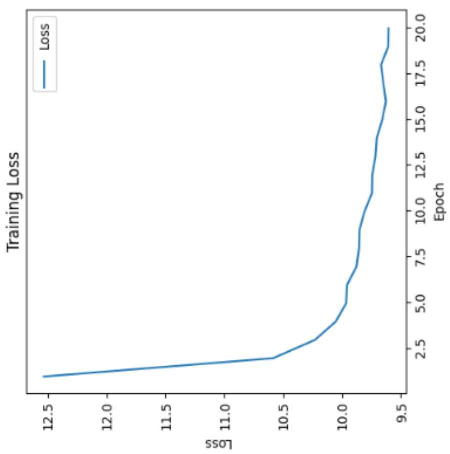
last epoch	Epoch 20 Summary: Loss: 9.5133 IoU: 0.0022 Precision: 0.0012 Recall: 0.0026 F1-Score: 0.0010 mAP: 0.0012	Epoch 35 Summary: Loss: 9.5207 IoU: 0.0002 Precision: 0.0001 Recall: 0.0000 F1-Score: 0.0000 mAP: 0.0000	Epoch 40 Summary: Loss: 8.7704 IoU: 0.0049 Precision: 0.0030 Recall: 0.0031 F1-Score: 0.0028 mAP: 0.0040
overall	 	/	/

Table 4 Comparative table of ResNet-101 Backbone with a subset of the mixed Dataset

Res-Net 50			
results	medium dataset	large dataset	X-Large dataset
	Train: 1000 Valid: 500 epochs: 20	Train: 3000 Valid: 1000 epochs: 30	Train: 6000 Valid: 3000 epochs: 40
first epoch	Epoch 1 Summary: Loss: 11.4229 IoU: 0.0000 Precision: 0.0000 Recall: 0.0000 F1-Score: 0.0000 mAP: 0.0000	Epoch 1 Summary: Loss: 10.8595 IoU: 0.0000 Precision: 0.0000 Recall: 0.0000 F1-Score: 0.0000 mAP: 0.0000	/

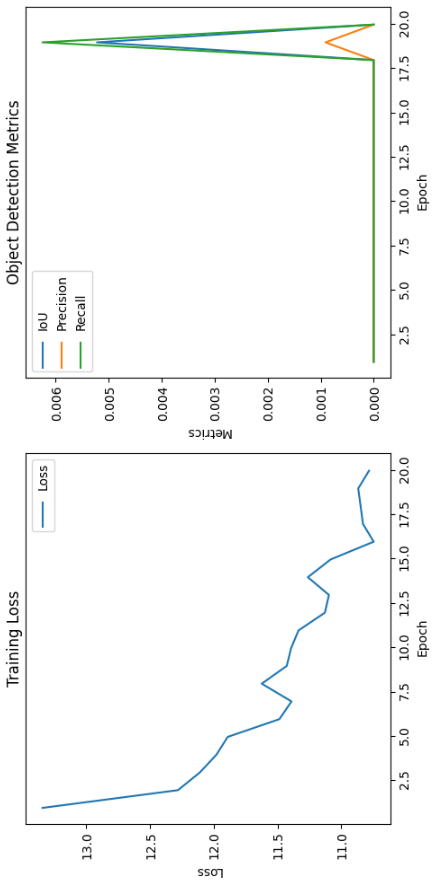
last epoch	Epoch 20 Summary: Loss: 9.6990 IoU: 0.0019 Precision: 0.0004 Recall: 0.0013 F1-Score: 0.0004 mAP: 0.0001	Epoch 30 Summary: Loss: 9.6977 IoU: 0.0003 Precision: 0.0002 Recall: 0.0002 F1-Score: 0.0002 mAP: 0.0004	/
overall		/	/

Table 5 Comparative table of ResNet-50 Backbone with a subset of the mixed Dataset

3.1.2 Running without augmentation - **One dataset (*wildfire-satellite2*)**

EfficientNet-B3		
results	train : all ; valid all;epochs : 20	train : all ; valid all;epochs : 20 + **Histogram equalisation**
first epoch	Epoch 1 Summary: Loss: 0.3014 IoU: 0.3764 Precision: 0.0419 Recall: 0.9402 F1-Score: 0.0773 mAP: 0.0704	Epoch 1 Summary: Loss: 0.3057 IoU: 0.3786 Precision: 0.0413 Recall: 0.9444 F1-Score: 0.0765 mAP: 0.0718

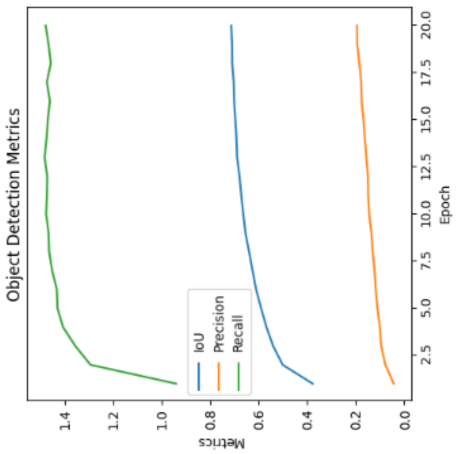
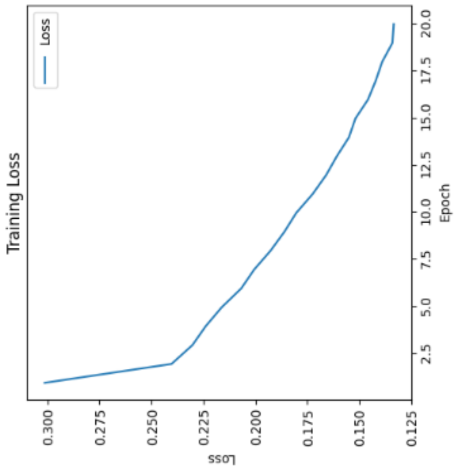
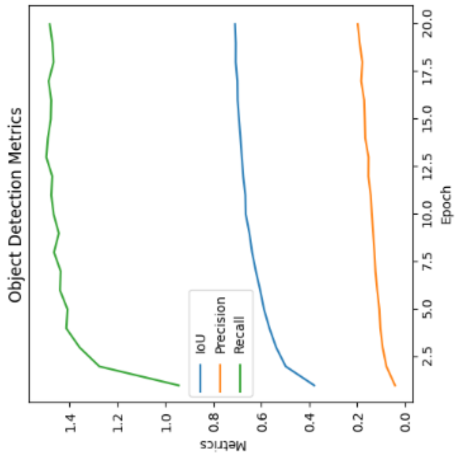
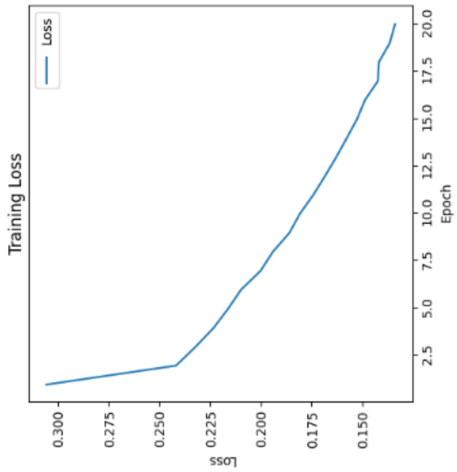
last epoch	Epoch 20 Summary: Loss: 0.1334 IoU: 0.7127 Precision: 0.1937 Recall: 1.4782 F1-Score: 0.3296 mAP: 0.3946	Epoch 20 Summary: Loss: 0.1337 IoU: 0.7094 Precision: 0.1971 Recall: 1.4831 F1-Score: 0.3329 mAP: 0.3960
overall	  	 

Table 6 : Comparative table of EfficientNet-B3 Backbone with the wildfire-satellite2 Dataset

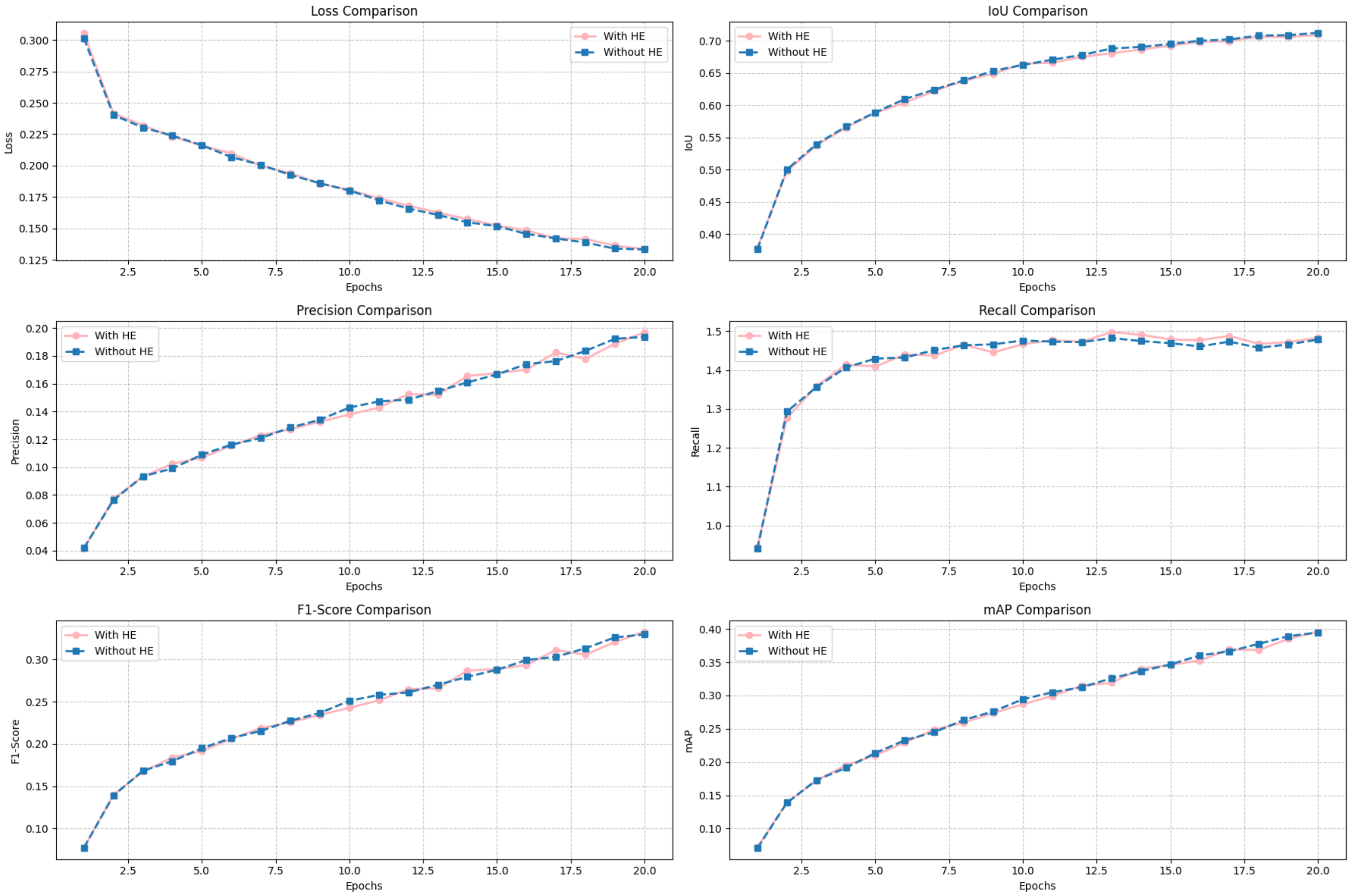


Figure 6 : Comparing results for the Efficient-NET-B3 Backbone for a model WITHOUT histogram equalization and for the model WITH histogram equalization (HE)

ResNet101		
results	train : all ; valid all ;epochs : 20	train : all ; valid all;epochs : 20 + **Histogram equalisation**
first epoch	Epoch 1 Summary: Loss: 0.3040 IoU: 0.3624 Precision: 0.0755 Recall: 0.7956 F1-Score: 0.1225 mAP: 0.1105	Epoch 1 Summary: Loss: 0.3061 IoU: 0.3608 Precision: 0.0723 Recall: 0.7921 F1-Score: 0.1187 mAP: 0.1070
last epoch	42504.8s 282 **Epoch 18** Summary: 42504.8s 283 Loss: 0.2025 42504.8s 284 IoU: 0.6710 **42504**.8s 285 Precision: 0.1551 42504.8s 286 Recall: 1.3622	**40884**.2s 257 **Epoch 15** Summary: 40884.2s 258 Loss: 0.2152 40884.2s 259 IoU: 0.6572 40884.2s 260 Precision: 0.1492 40884.2s 261 Recall: 1.3544

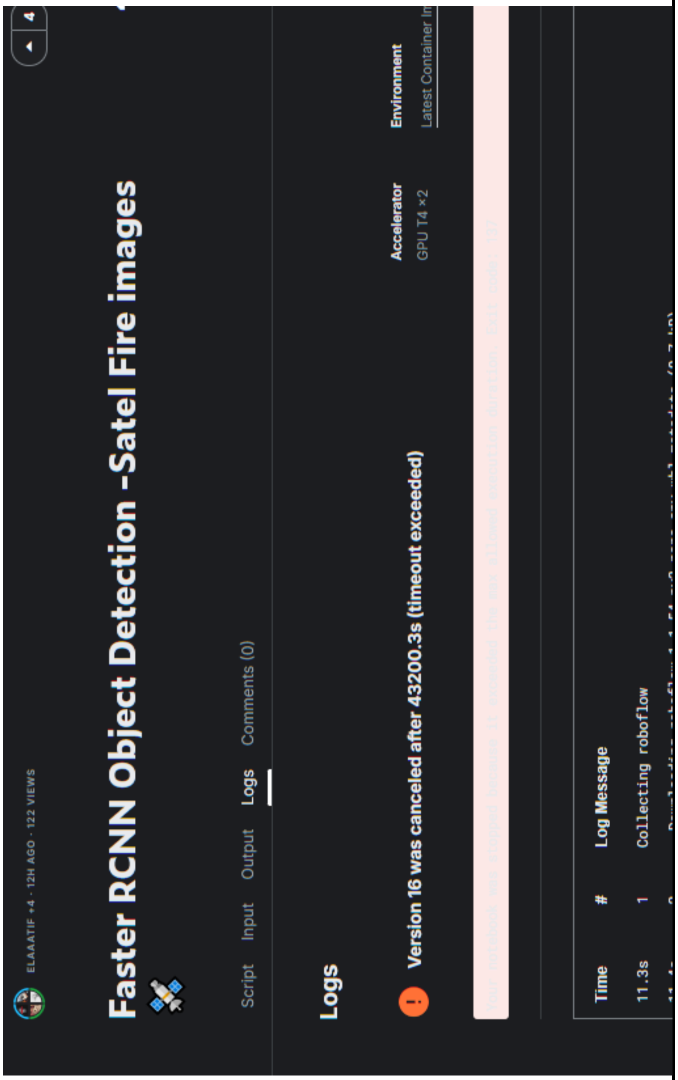
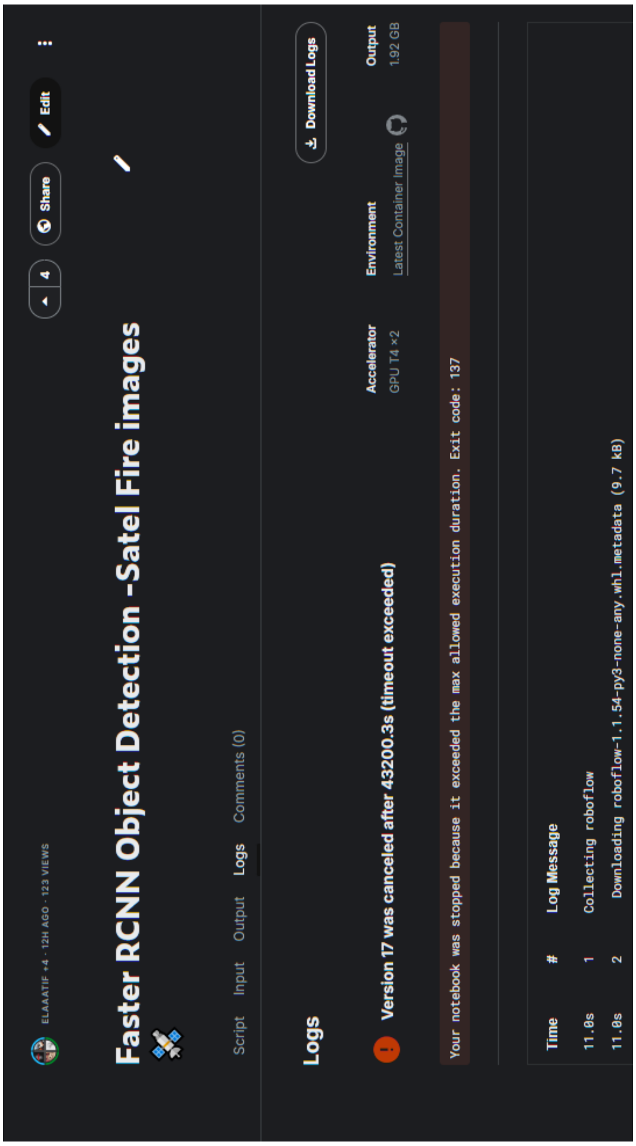
	42504.8s 287 F1-Score: 0.2649 42504.8s 288 mAP: 0.3022	40884.2s 262 F1-Score: 0.2553 40884.2s 263 mAP: 0.2862
overall		

Table 7 : Comparative table of ResNet-101 Backbone with the wildfire-satellite2 Dataset

The error message indicates that your notebook on Kaggle was **terminated due to exceeding the maximum runtime limit of 12 hours (43,200 seconds)**. Here's a breakdown of the issue:

Explanation of the Error

1. **Timeout Exceeded (43200.3s)**
   - Kaggle imposes a time limit on notebook executions, which is **12 hours for GPU sessions**.
   - Your notebook ran for **43,200.3 seconds (~12 hours)** and was **automatically stopped**.
2. **Hardware Used: GPU T4 x2**
   - running the notebook with **two NVIDIA Tesla T4 GPUs**.

  - Despite using GPUs, the process was still time-consuming, likely due to computationally intensive task .
3. **Environment: Latest Container Image**
  - This indicates using the most up-to-date Kaggle container for execution environment.
4. **Output File Size: 1.92 GB**
  - The output generated by your notebook was **almost 2GB**, suggesting it was handling a large dataset and a complex model.

Possible Causes

- **Long training and computation time**
  - Training a deep learning model requires more epochs and processed a large dataset.
- **Memory constraints**
  - Large dataset processing or deep learning models might have caused excessive memory usage, leading to inefficiencies.
- **Inefficient use of GPUs**
  - GPU acceleration was limited, execution have been slower than expected.

ResNet50		
results	train : all ; valid all ;epochs : 20	train : all ; valid all;epochs : 20 + **Histogram equalisation**
first epoch	1- Epoch 1 Summary: Loss: 0.3271 IoU: 0.4535 Precision: 0.0656 Recall: 1.1450 F1-Score: 0.1195 mAP: 0.1121	/
last epoch	Epoch 7 Summary: Loss: 0.2411 IoU: 0.6449 Precision: 0.1393 Recall: 1.3537 F1-Score: 0.2423 mAP: 0.2714	/

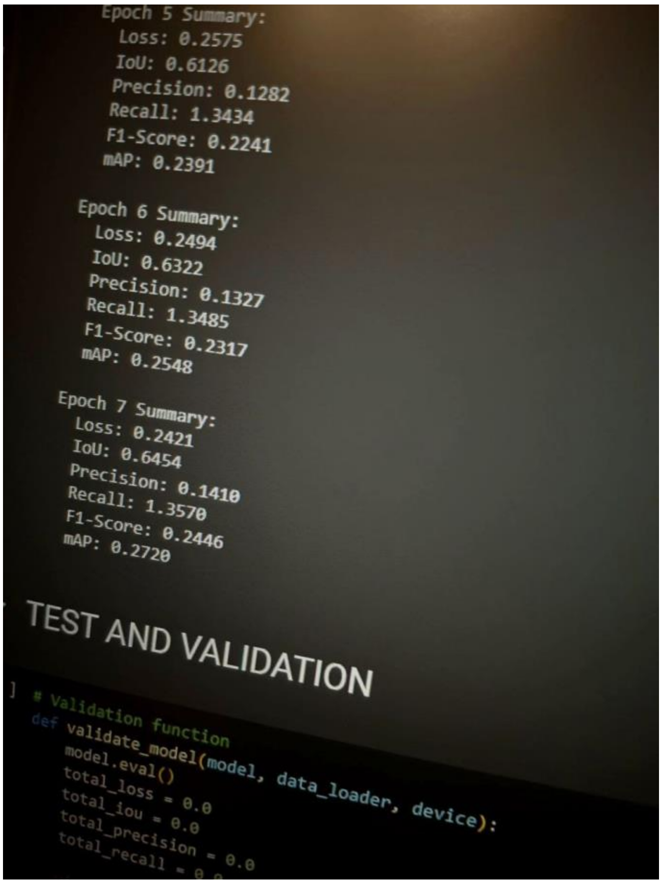
<table>
<tr><td>overall</td><td>
</td><td>/</td></tr>
</table>

Table 8 : Comparative table of ResNet-50 Backbone with the wildfire-satellite2 Dataset

Notebook Execution Issue: Unstable Connection

Issue:
During execution, the notebook was **interrupted due to an unstable connection**, causing the session to stop unexpectedly. This can happen due to network instability on Kaggle's servers or issues with the user's internet connection.

Possible Causes:

- **Kaggle server-side disconnection:** Sometimes, Kaggle automatically **disconnects idle sessions** or experiences temporary service disruptions.
- **User-side internet issues:** A weak or interrupted internet connection can lead to disconnections.
- **Long execution times:** If the notebook runs for an extended period, there is a higher chance of connection loss.

Impact:

- The execution was **not completed**, and any unsaved progress was lost.
- If the notebook was training a model, the results were **not saved** unless checkpointing was implemented.

3.2 Importance of Metrics in Wildfire Detection

- **IoU and mAP** ensure precise localization of fires, which is crucial for rapid intervention.
- **Precision and Recall** guarantee that fires are detected accurately without excessive false alarms.
- **F1-score** evaluates the balance between precision and recall.
- **Inference Time** is essential for real-time detection.
- **Memory Usage** impacts the feasibility of deploying the model on embedded systems or cloud platforms.

3.3 Technical Environment

The following technical setup was used for the experiments:

- **Platform**

  Kaggle is an online platform that provides a collaborative environment for data science and machine learning practitioners. It offers cloud-based computational resources, datasets, and machine learning competitions. Users can build, train, and deploy machine learning models using various frameworks, including TensorFlow and PyTorch. it provides gpu access (two type of GPUs; GPU P100 and two GPU t4 and NVIDIA Tesla P100)

- **Framework**:
  - **Primary Framework**: PyTorch
  - **Additional Tools**: roboflow, torch, torchvision, matplotlib, numpy ,pycocotools,
  -
- **Hardware**:
  - **GPU**: NVIDIA Tesla P100 comes with 16GB of high-bandwidth memory (HBM2). It is designed for high-performance computing (HPC) and deep learning workloads, supporting CUDA and cuDNN for optimized GPU acceleration.
  - **CPU**: kaggle provides a maximum of 4 Intel(R) Xeon(R) CPU @ 2.20GHz
  - **Memory**: kaggle provides a maximum of 29 GiB of RAM for notebooks, which is crucial for handling large datasets and deep learning models requiring high memory bandwidth

Although Kaggle provided a well-equipped environment for training and testing, it was not sufficient to run computationally intensive model such as Faster R-CNN efficiently. The resource limitations, particularly in terms of GPU availability and memory constraints, posed challenges in achieving optimal performance and faster convergence.

4. Discussion

Although the model is known for being fast and resilient for real-time object detection, we encountered significant challenges in implementing it for fire detection due to hardware and software limitations. Running an intensive task such as Faster R-CNN, along with its backbone models, is not only time-consuming but also highly resource-intensive. The computational demands exceeded the available hardware capabilities, leading to slower training times and suboptimal performance. These constraints impacted the model's ability to efficiently process high-resolution images and detect fire instances in real-time.

When experimenting with different backbones for Faster R-CNN, we explored alternatives such as Swin-Tiny and MobileNet, in addition to our initial choices of EfficientNet and ResNet-50. While EfficientNet and ResNet-50 provided relatively stable and well-structured feature maps that seamlessly integrated into the Faster R-CNN architecture, Swin-Tiny and MobileNet introduced challenges due to their distinct architectural properties. Swin-Tiny, being a transformer-based backbone, generates hierarchical feature representations with a different spatial resolution and attention-based feature extraction mechanism. This resulted in an output structure that did not directly align with the feature pyramid network (FPN) or region proposal network (RPN) expectations. Similarly, MobileNet, designed for lightweight applications, utilizes depthwise separable convolutions that reduce computational complexity but also alter the spatial and channel-wise characteristics of the extracted features. These differences posed integration difficulties, requiring additional modifications or adjustments in the feature alignment process to ensure compatibility with the downstream components of Faster R-CNN. Consequently, while Swin-Tiny and MobileNet hold promise in terms of efficiency and performance trade-offs, their integration into Faster R-CNN necessitates careful consideration of feature compatibility and adaptation strategies.

5. Conclusion

5.1 Summary of Contributions

This study has significantly advanced wildfire detection by leveraging **Faster R-CNN** with three distinct backbone architectures: **ResNet-50, ResNet-101, and EfficientNet**. The key contributions of this work include:

- **Enhanced Detection Performance**: Assessed the effectiveness of various backbone networks in object detection, leading to improved accuracy and robustness.
- **Comprehensive Comparative Analysis**: Investigated the trade-offs between accuracy, inference time, and computational efficiency across the three architectures.
- **Optimized Training Strategy**: Employed **grid search** and **cross-validation** to fine-tune hyperparameters, ensuring stability and optimal performance.

5.2 Future Perspectives

Several avenues for future enhancements can be explored, including:

- **Real-Time Deployment**: Implementing the model for real-time wildfire detection and automated alert systems.
- **Integration with Surveillance Systems**: Deploying the model in drone-based and satellite-based fire monitoring systems to enhance early detection.
- **Performance Optimization**: Reducing inference time and memory consumption to improve efficiency, particularly for real-time applications on edge devices.
- **Data Augmentation and Expansion**: Increasing dataset diversity through advanced augmentation techniques to enhance generalization across different environmental conditions.

By building upon these improvements, this research paves the way for more accurate and efficient wildfire detection solutions.

References

	Articles
[1]	Ren, S., He, K., Girshick, R., & Sun, J. (2016). *Faster R-CNN: Towards real-time object detection with region proposal networks*. arXiv. https://arxiv.org/abs/1506.01497

[2]	Deng, Z., Sun, H., Zhou, S., Zhao, J., Lei, L., & Zou, H. (2018). Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing, 145*, 1-12. https://doi.org/10.1016/j.isprsjprs.2018.04.003
[3]	Agarwal, V. (2020, May 24). Complete architectural details of all EfficientNet models. *Towards Data Science*. https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142
[4]	Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv*. https://arxiv.org/pdf/1905.11946v3

Code Source

For further details and access to the project code, please visit theGitHub repository bellow

https://github.com/elaaatif/Computer-Vision