Internship Report

On

REAL TIME OBJECT DETECTION USING YOLOv8 AND MOBILENET V2

Spartificial Innovations Private Limited

(INTERNSHIP ID: meml-gri)

Ms. Vanitha P

Mr. Rajasekhar Kommaraju

Mr. Aswant Suresh

Mr. Shrirang Kanade

Guided by

Mr. Sajan Sudhir

TABLE OF CONTENTS

ABSTRACT

This report presents a comprehensive study on object detection techniques applied to the moon's surface for the identification of rocks. The primary focus is on evaluating two popular models, MobileNet v2 and YOLO v8, for their performance and accuracy in detecting rocks on the lunar terrain.

The study utilizes a diverse dataset consisting of high-resolution images captured by lunar rovers and satellites. The dataset includes various lighting conditions, terrain types, and rock sizes to provide a realistic representation of lunar surface characteristics. A thorough pre-processing phase is conducted, including image enhancement, noise reduction, and data augmentation, to improve the models' robustness.

The MobileNet v2 and YOLO v8 models are both state-of-the-art object detection frameworks known for their speed and accuracy. They are trained on the lunar rock dataset using transfer learning techniques, leveraging pre-trained weights from ImageNet. Extensive experiments are conducted to fine-tune the models and optimize their hyperparameters for the lunar rock detection task.

Overall, this report provides valuable insights into the application of object detection models, specifically MobileNet v2 and YOLO v8, for lunar rock detection. The findings can aid in the development of automated systems for geological analysis, mapping, and future lunar exploration missions, facilitating a deeper understanding of the moon's geology and its potential for scientific exploration.

CHAPTER 1

INTRODUCTION TO OBJECT DETECTION

In recent years, computer vision has witnessed significant advancements, leading to breakthroughs in various applications and industries. One fundamental task within computer vision is object detection, which involves identifying and localizing objects of interest within images or video frames. Object detection plays a vital role in a wide range of domains, including autonomous driving, surveillance, robotics, augmented reality, and healthcare.

Object detection provides machines with a deeper understanding of visual data by accurately detecting and localizing objects. This enables algorithms to extract meaningful information, make informed decisions, and perform complex tasks. Object detection goes beyond recognition, providing spatial information through bounding box coordinates for each detected object.

Object detection techniques have evolved significantly, driven by the advancements in deep learning and convolutional neural networks (CNNs). Traditional methods, such as the sliding window approach and feature-based classification, have paved the way for modern deep learning approaches like region-based CNNs (R-CNN), Single Shot Detectors (SSD), You Only Look Once (YOLO), and EfficientDet. These deep learning models have revolutionized object detection by achieving remarkable accuracy and real-time performance.

The applications of object detection are vast and diverse. In autonomous driving, object detection is critical for identifying pedestrians, vehicles, and traffic signs, enabling the development of advanced driver assistance systems (ADAS) and autonomous vehicles. Surveillance and security systems rely on object detection to detect and track suspicious activities, intruders, or objects of interest. In robotics, object detection enables machines to interact with their environment and manipulate objects intelligently. Augmented reality applications heavily rely on object detection to overlay virtual objects onto the real world accurately. In healthcare, object detection assists in medical imaging for identifying anatomical structures, tumors, or abnormalities.

CHAPTER 2

REVIEW OF STATE-OF-THE-ART & RELATED BACKGROUND

2.1 MobileNetV2 :

MobileNetV2 is a convolutional neural network (CNN) architecture specifically designed for efficient and lightweight image classification and object detection on mobile and embedded devices. It is an improved version of the original MobileNet architecture, aiming to achieve high accuracy while minimizing computational complexity and model size.

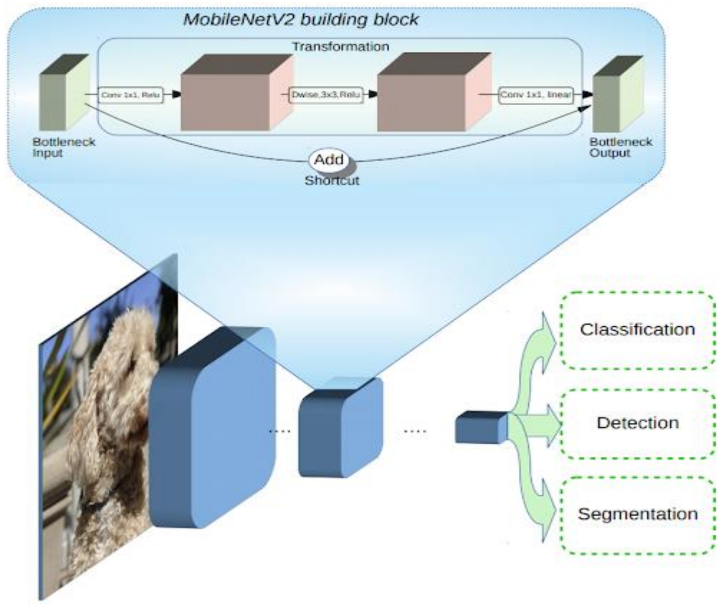


Figure 1: Overview of MobileNetV2 Architecture.

Blue blocks represent composite convolutional building blocks as shown above.

MobileNetV2 is a state-of-the-art object detection model known for its speed and accuracy. It is often used in various applications, including obstacle detection. With its efficient architecture and lightweight design, MobileNetV2 enables real-time obstacle detection in resource-constrained environments.

MobileNetV2 utilizes depthwise separable convolutions, which significantly reduces the computational complexity while maintaining high detection performance. This makes it suitable for real-time obstacle detection tasks, where quick and accurate identification of obstacles is crucial for safe navigation.

MobileNetV2's speed and accuracy make it a popular choice for obstacle detection in autonomous driving, robotics, and surveillance systems. Its ability to efficiently process images and detect obstacles in real-time contributes to enhanced situational awareness and safer navigation in dynamic environments.

Advantages and Applications:

MobileNetV2 offers several advantages that make it suitable for mobile and embedded devices:

- Efficiency: Its lightweight design makes it computationally efficient, enabling real-time inference on resource-constrained devices.

- Model Size: MobileNetV2 has a small memory footprint, making it ideal for deployment on devices with limited storage capacity.

- Accuracy: Despite its efficiency, MobileNetV2 achieves competitive accuracy on image classification and object detection tasks, making it suitable for various applications.

- Mobile and Embedded Deployment: MobileNetV2 is specifically optimized for mobile and embedded platforms, allowing for on-device inference and reducing reliance on cloud-based processing.

MobileNetV2 finds applications in a wide range of domains, including mobile image classification, object detection, semantic segmentation, and visual recognition tasks. Its efficiency and accuracy make it suitable for use in smartphones, IoT devices, drones, robotics, and other resource-constrained environments.

Overall, MobileNetV2 is a powerful architecture that strikes a balance between model size, efficiency, and accuracy, making it a popular choi

2.2 YOLOV8:

YOLOv8 (You Only Look Once version 8) is the latest iteration of the popular YOLO algorithm, which revolutionized real-time object detection with its single-shot detection approach. YOLOv8 builds upon the strengths of its predecessors and introduces several key enhancements to further improve accuracy and speed.

Features of YOLOv8 :

- YOLOv8 demonstrates improved accuracy in object detection compared to previous versions.
- It incorporates spatial attention, feature fusion, and context aggregation modules for better contextual understanding and more accurate predictions.
- YOLOv8 maintains impressive inference speeds, making it suitable for real-time applications.
- It offers flexibility by supporting multiple backbones, allowing users to choose the backbone that best suits their needs.
- YOLOv8 employs adaptive training techniques to optimize the learning rate and balance the loss function during training.
- YOLOv8's architecture is highly customizable, enabling modifications to the model's structure and parameters.
- Pre-trained models are available, facilitating ease of use and transfer learning for new datasets or specific applications.
- YOLOv8 represents a state-of-the-art object detection algorithm with improved accuracy, speed, flexibility, adaptive training, advanced data augmentation, customizable architecture, and pre-trained models.
- These advancements enable efficient and accurate handling of complex object detection challenges, opening up new possibilities in computer vision applications.

With its improved accuracy, enhanced speed, flexibility, adaptive training, advanced data augmentation, customizable architecture, and pre-trained models, YOLOv8 stands as a state-of-the-art object detection algorithm. Its advancements empower researchers and developers to tackle complex object detection challenges efficiently and accurately, opening up new possibilities in computer vision applications.

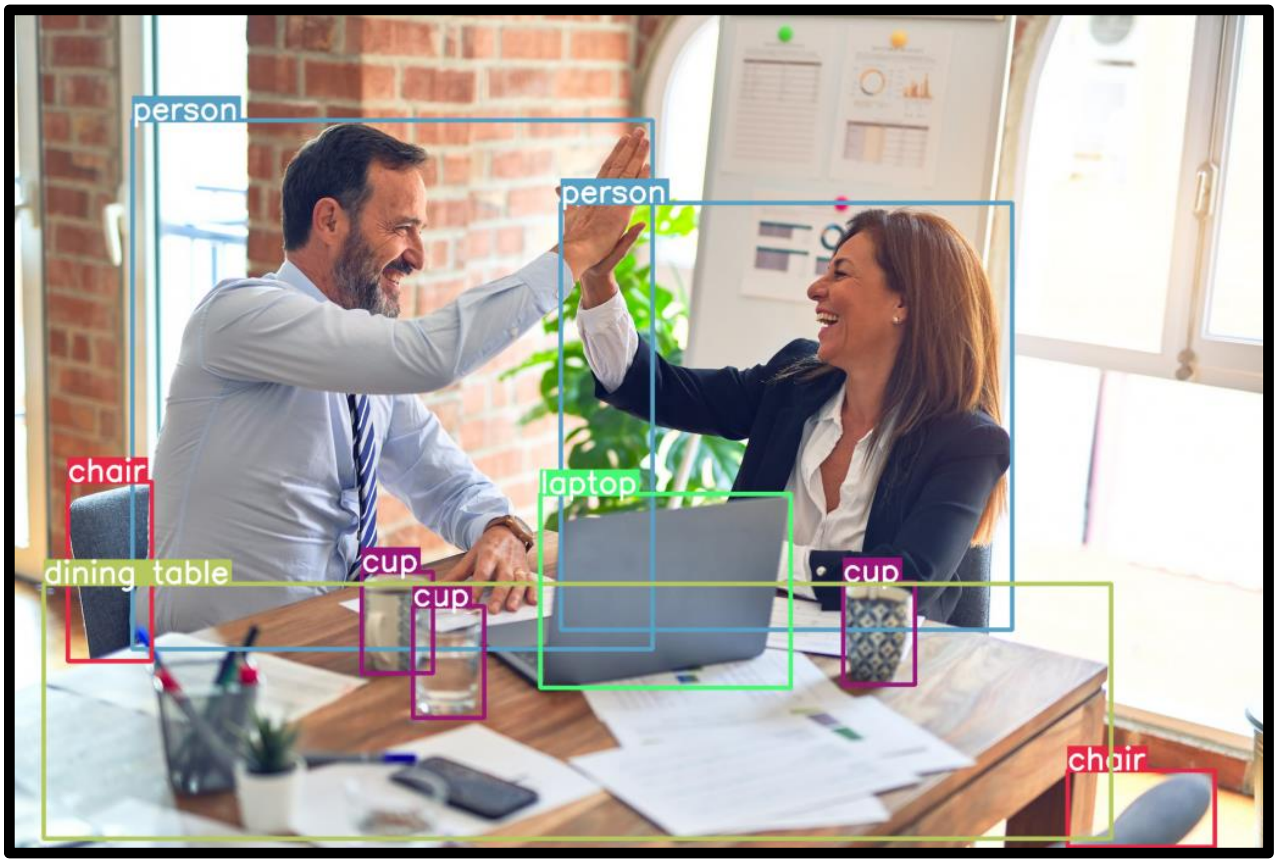


Figure 2: YOLOv8 Object Detection

2.3 Loss functions:

Loss functions play a crucial role in training machine learning models by quantifying the discrepancy between predicted values and ground truth labels. In the context of object detection, two commonly used loss functions are Mean Squared Error (MSE) with sigmoid activation and Smooth L1 loss. Let's explore each of them in more detail:

1. **Mean Squared Error (MSE) with Sigmoid Activation:**

   - MSE is a popular loss function for regression tasks.
   - In object detection, it is used to predict the probability of object presence and bounding box coordinates.
   - Sigmoid activation function maps the predicted probability between 0 and 1.
   - MSE computes the squared difference between predicted probability and ground truth, emphasizing larger discrepancies.
   - Provides a smooth and continuous loss function for gradient-based optimization.

2. **Smooth L1 Loss:**

   - Robust alternative to MSE, especially for bounding box regression.
   - Uses a piecewise function instead of squaring the difference.
   - Absolute difference for small differences and squared difference for larger differences.
   - More resilient to outliers and stable training.
   - Effective when ground truth bounding boxes have a wide range of scales and aspect ratios.

Both MSE with sigmoid activation and Smooth L1 loss have their advantages and are suitable for different scenarios in object detection. The choice of loss function depends on task requirements, dataset nature, and desired balance between accuracy and robustness. Experimentation and evaluation help determine the better-performing loss function.

2.4 Intersection Over Union (IoU):

The Intersection over Union (IoU) is a widely used evaluation metric in computer vision, particularly in tasks such as object detection, segmentation, and instance recognition. IoU measures the spatial overlap between predicted and ground truth regions of interest. Here's some information about IoU:

1.Definition:

IoU is defined as the ratio of the intersection area between the predicted and ground truth regions to the union area of those regions. It quantifies how well the predicted region aligns with the ground truth region.

2.Calculation:

To calculate IoU, the following steps are typically performed:

- Compute the intersection area: Determine the overlapping region between the predicted bounding box/segmentation mask and the ground truth bounding box/segmentation mask.

- Compute the union area: Determine the combined area of both the predicted and ground truth regions.

- Calculate the IoU: Divide the intersection area by the union area.

- IoU Score Interpretation:

  The IoU score ranges from 0 to 1, where:

  IoU = 0 indicates no overlap between the predicted and ground truth regions.
  IoU = 1 indicates a perfect match between the predicted and ground truth regions.

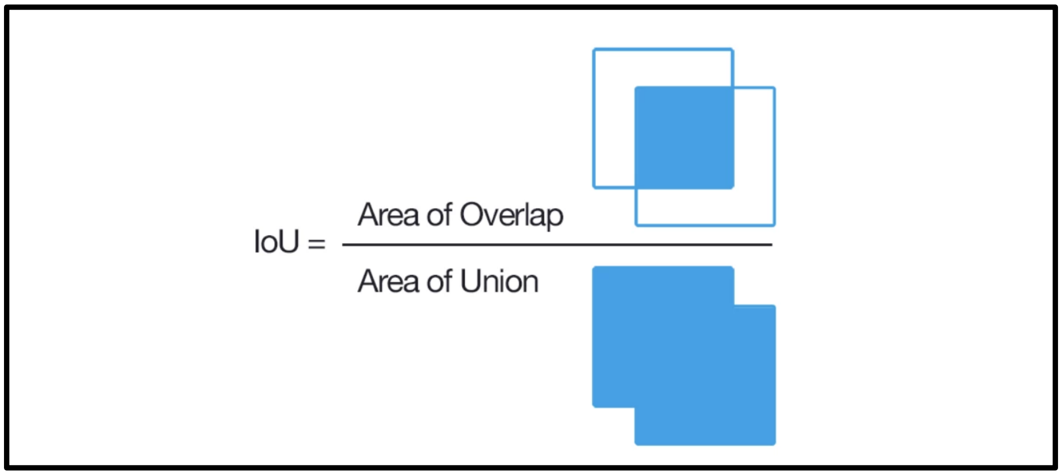


Figure 3: Intersection Over Union

3.Usage and Application:

IoU is commonly used in object detection and segmentation tasks to assess the accuracy of models. It helps evaluate how well the model localizes objects and captures their boundaries. In object detection, the IoU metric is often used to determine true positive, false positive, and false negative predictions based on predefined thresholds. It can also be used to calculate the mean Average Precision (mAP) for object detection models.

Overall, IoU is a valuable metric for assessing the performance of computer vision models. It provides a quantitative measure of the spatial agreement between predicted and ground truth regions, enabling researchers and practitioners to evaluate and compare different models and techniques.

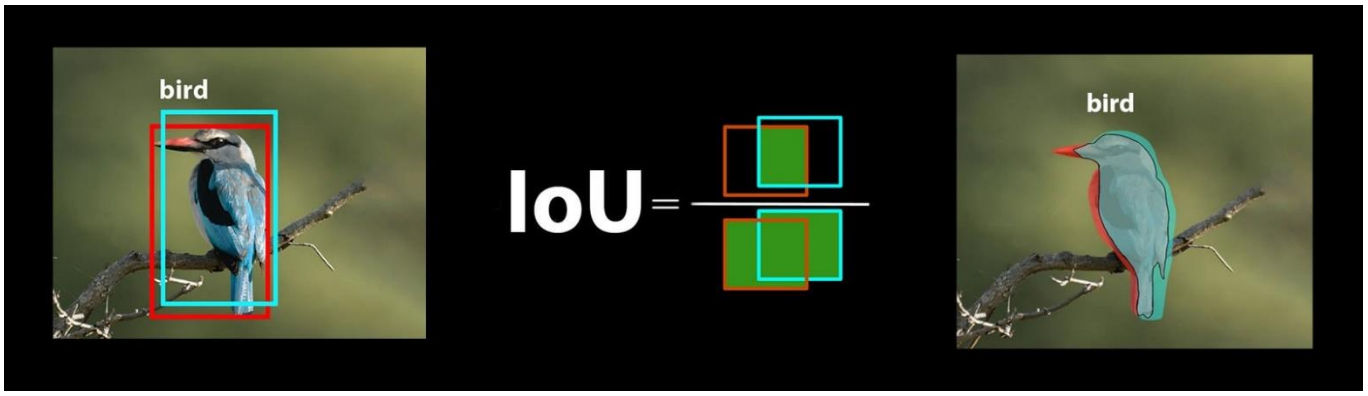


Figure 4: IoU Example

CHAPTER 3

PROJECT IMPLEMENTATION

3.1 YOLO V8 Implementation:

Ultralytics YOLOv8 is a cutting-edge, state-of-the-art (SOTA) model that revolutionizes computer vision with its speed, accuracy, and flexibility. It builds upon the success of previous YOLO versions and introduces new features and improvements. YOLOv8 has achieved significant progress in terms of speed, accuracy, and architectural design, surpassing YOLOv7. It has set a new benchmark with an outstanding Mean Average Precision (MAP) score of 53.7.

1. **Module Installation**
2. **Pretrained Object Detection**
3. **Custom Data Training**
4. **Inference with Custom Weights**

By following these steps diligently, I have successfully trained YOLOv8 on custom data, ensuring reliable performance on diverse operating system.

1. Module Installation:

The first step involves installing the necessary modules required for YOLOv8.To leverage the capabilities of YOLOv8, then utilized the "ultralytics" package, which was specifically developed for this purpose. The package installation was straightforward and involved executing the following command:

```
!pip install ultralytics
```

This command ensured the installation of all the necessary packages required to utilize YOLOv8 for detection and training on custom data.

After the successful installation, the next step is to import the YOLO module from the "ultralytics" package using the following command:

```
from ultralytics import YOLO
```

By importing YOLO from "ultralytics", one can gain access to the functionalities and features provided by YOLOv8 for efficient object detection and training.

3. Pretrained Object Detection:

For efficient object detection with improved accuracy and faster speed, a single command can be executed to achieve optimal results. By running the following command in the terminal or command, object detection can be performed using pre-trained weights and YOLOv8:

!yolo task=detect mode=train model=yolov8l.pt

data='/content/drive/MyDrive/Internship/moon_new/data.yaml'epochs=30 imgsz=640

Upon executing this command, the detection process will commence, utilizing the specified pre-trained weights and processing your chosen video or image. If everything functions as expected, the results will be saved in the **"runs/detect/train"** directory within the current location.

4. Custom Data Training

The steps for training a YOLOv8 object detection model on custom data can be summarized as follows,

- Collect data
- Label data
- Split data (train, test, and val)
- Creation of config files
- Start training

Step-1: Collect Data

To train YOLOv8 on custom data, I prepared a dataset following the required format. YOLOv8 expects the label data to be stored in text (.txt) files, adhering to the following format:

<object-class-id> <x> <y> <width> <height>

Step 2: Label Data

In dataset, the images were already labeled with bounding box coordinates in the specified format (<object-class-id> <x> <y> <width> <height>). However, the labels were initially provided in a CSV (.csv) file. To convert the CSV file to the required text (.txt) format, a Python script utilizing libraries such as os, pandas, and PIL. This script facilitated the conversion process seamlessly.

Once the labels were successfully converted to the .txt format, then next step is to conduct a comparison between the labels in the .txt files and the corresponding images in the training image folder. The comparison was performed by considering the filenames without file extensions. This analysis aimed to identify any discrepancies between labeled data and image files.

Upon inspection, it was discovered that a few images within the training image folder were not labeled. To maintain consistency, those particular images were removed from the corresponding image folder. This step ensured that the dataset contained accurately labeled images, aligning with the provided bounding box coordinates in the text files.

Step-3: Split Data (Train, Test, and Val)

To ensure effective training of a computer vision model on custom data, it is crucial to partition the dataset into distinct subsets: the training set and the test set. The training set is utilized to educate the model on making accurate predictions, while the test set evaluates the model's performance and accuracy. While a common practice is to split the data in an 80-20% ratio, the exact ratio may vary depending on the dataset size and the specific task at hand. For instance, with a small dataset, a higher percentage might be allocated for training, while a larger dataset can afford a smaller training percentage.

In the provided dataset, there was no dedicated validation dataset available. To address this, the validation dataset was created by selecting 7% of the training data. This was accomplished using Python libraries such as os, random, and shutil. Subsequently, the the same code is applied which was used earlier to convert the labels of the validation dataset from CSV format to the required TXT format.

Dataset Folder Structure

```
YOLOv8 Dataset Folder
  ├── Train Folder
  │     ├── Image Files (folder including all training images)
  │     └── Label Files (in TXT format)
  ├── Test Folder
  │     ├── Image Files (folder including all training images)
  │     └── Label Files (in TXT format)
  └── Validation Folder
        ├── Image Files (folder including all training images)
        └── Label Files (in TXT format)
```

By adhering to this structure, the dataset was well-organized, facilitating smooth training and evaluation processes for the computer vision model.

Step-4: Creation of Config Files

To create a custom configuration file for organizing and storing important parameters of a computer vision model, follow these steps:

1. Create a file named "data.yaml" within the current directory, where the terminal or command prompt is open.
2. Copy and paste the provided code into the "data.yaml" file.
3. Set the correct paths for the dataset folders by replacing the placeholders with the appropriate paths:

```
train: /content/drive/MyDrive/Internship/moon_new/train
val: /content/drive/MyDrive/Internship/moon_new/val
test: /content/drive/MyDrive/Internship/moon_new/test

nc: 1
```

```
names: ['0']
```

If the dataset consists of a single class, modify the "nc" (number of classes) value to reflect the actual number of classes. In this case, it is set to 1. Update the "names" field with the class names. If there is only one class, replace the placeholder '0' with the appropriate class name. For multiple classes, list the names accordingly. Finally save the "data.yaml" file. It's crucial to ensure that the paths for the training and testing directories are set correctly, as the training process relies on this configuration file.

Step-5: Start Training

Once the necessary preprocessing steps, such as data collection, data labeling, data splitting, and creating a custom configuration file, have been completed, the training of YOLOv8 on custom data can be initiated using the following command in the terminal or command prompt:

```
!yolo task=detect mode=train model=yolov8l.pt

data='/content/drive/MyDrive/Internship/moon_new/data.yaml' epochs=30
imgsz=640
```

In this command, the parameters are defined as follows:

- task: detect; (Specifies the task to be performed (detect, segment, or classify)).
- mode: train; (Determines the mode of operation (train, predict, or val)).
- model: YOLOv8l; (Specifies the YOLOv8 model to be used (yolov8n, yolov8s, or yolov8x)).
- epochs: 30; (Sets the number of training epochs).
- imgsz: 640; (Defines the image size for training (it should be a multiple of 32, such as 320, 416, etc.)).

During the training process, YOLOv8 ensures that corrupt images do not impede the training. If any image or label file is corrupted, YOLOv8 gracefully ignores them without causing any issues.

Upon successful training, the custom-trained weights will be saved in the following folder path:

/runs/detect/train/weights/best.pt

By executing this command, YOLOv8 will initiate the training process on custom data, allowing the model to learn and generate optimized weights for object detection tasks.

Inference with Custom Weights

Once the model is trained, it becomes possible to utilize it for making predictions on new data. The following command was employed to perform object detection using the custom weights:

```
best = YOLO("/content/runs/detect/train/weights/best.pt")

#test images with trained model

best.predict("/content/drive/MyDrive/Internship/moon_new/test/images", save=True)
```

In this code snippet, the "best" object of the YOLO class is created, and the trained model's custom weights are loaded from the specified path:

("/content/runs/detect/train/weights/best.pt").

Subsequently, object detection is performed on the test images located in the directory:

"/content/drive/MyDrive/Internship/moon_new/test/images".

Additionally, by setting the "save" parameter to True, the results of the predictions can be saved. Considering the specific task at hand, The obtained results, generated using the custom weights can be found in the results section of the report.

3.2 MobileNet V2 implementation

1. MobileNet V2:

SSD Mobilenet V2 is a one-stage object detection model which has gained popularity for its lean network and novel depth wise separable convolutions. It is a model commonly deployed on low compute devices such as mobile (hence the name Mobilenet) with high accuracy performance.

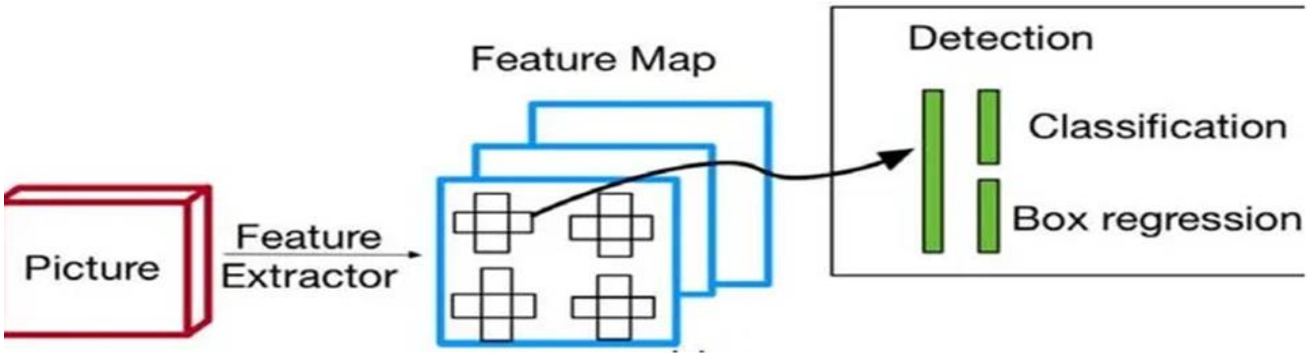
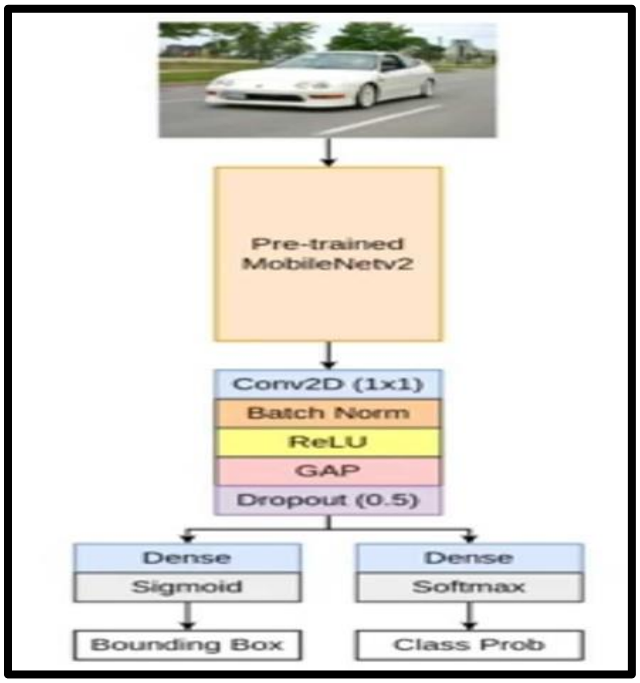


Figure 3: One Stage Detection

2. Model Fine Tuning Approach:



- The model starts with an input layer that shapes the input image. It uses a pre-trained MobileNetV2 backbone, trained on ImageNet, to extract useful features. This leverages the backbone's knowledge for effective feature capture.

- The trainable backbone fine-tunes its weights during training, adapting to the specific object detection task. The backbone's output goes through a 1x1 convolutional layer, batch normalization, ReLU activation, and dropout for enhanced learning and generalization.

- Global average pooling reduces spatial dimensions while retaining essential information. A dense layer predicts bounding box coordinates, normalized between 0 and 1 using the sigmoid activation. The output is reshaped into (max_boxes, 4), representing the boxes' coordinates.

- The final model connects the input and output layers. The input layer receives the image, and the output layer provides predicted bounding box coordinates for each detected object

3. Dataset Preparation

The dataset used for the project is Rockfall Detection on Moon ,A Labeled Image Dataset for Deep Learning-Driven Rockfall Detection on the Moon. The dataset is available on Kaggle and it can be used for carrying experiments.

Dataset in kaggle contains folder structure as:

```
rockfall-detection-on-moon
        |
        |— moon
             |
             |— test_images
             |
             |— test_labels
             |
             |— train_images
             |
             |— train_labels
```

Here test_images contains images which are 34 in total out of which 17 are labeled. These images can be used to infer the results from the trained model as this data can be termed as unseen data. Test_labels contain csv files which contain image name, bounding box coordinates in Pascal voc format and label name of each image.

The train_images contain 660 images out of which only 349 are labeled. These images are used for the training of models. The train_labels folder contains the csv files just like the test_labels one for the training images.

So, in dataset preparation only for training and testing only labeled images are extracted and all bounding box coordinates for each image are stored in a single list. It means there will be only a single list which would contain multiple bounding box coordinates related to that image.

4. Challenges:

- Data shortage
- Each Image contains distinct number of bounding boxes

To deal with the above two challenges, the optimal solution was to use data augmentation to deal with the data shortage and to deal with the variable number of bounding boxes, the tensors were padded with a list of 0s to make them of the same size.

As if one list of images contains only 2 bounding box coordinates then we add 18 extra lists of zeros to make their length 20.

5. Data Augmentation approach

To address the issue of data shortage, data augmentation approach helped to increase the data samples in the dataset. Data augmentation increases the samples by applying the transforms on images like rotation of image, scaling of image etc. So, the one more challenge regarding data augmentation was that it was easy to create samples from image by its transformation but we have to also transform the bounding box coordinates of image accordingly. So, to deal with this issue, the Albumentations library helped to achieve success in data augmentation.

Albumentations library utilizes the bbox_params parameter in the transformation pipeline. By specifying the bounding box format (e.g., Pascal VOC format) and indicating the label fields, Albumentations understands how to handle the bounding boxes during the transformation process.

When the transformation is applied to an image, Albumentations adjusts the image accordingly and updates the bounding box coordinates to reflect the changes. This ensures that the bounding boxes align with the transformed objects accurately.

Transforms used in augmentation:

- ShiftScaleRotate
- HorizontalFlip
- GaussNoise, GaussianBlur, MotionBlur
- Random Brightness Contrast, ColorJitter
- OpticalDistortion, ElasticTransform, GridDistortion, Perspective

6. Training Pipeline Implementation approach

A machine learning pipeline refers to a systematic and organized sequence of data processing and modeling steps applied to a dataset to build and deploy a machine learning model. The pipeline is defined by creating the functions and chaining them at last.

- load_data() : Extracts Images from folder along with their bounding box coordinates from csv simultaneously.

- data_augmentation() : After extraction of images, we get a total of 349 images, then we apply augmentation and create 7 distinct samples for a single image and total samples become 2443. It starts by image resizing and normalizing images along with their bounding boxes after this they are augmented.

- load_dataset() : The load_dataset function loads the dataset, performs data augmentation on the training set, splits the data into training and validation sets, and loads the test set. It returns the processed training, validation, and test data along with any additional test images.

- read_image_bbox() : It is used to read image and its corresponding bounding boxes.

- parse() : It is used to parse and preprocess an image and its corresponding bounding box information. The function utilizes the tf.numpy_function method to apply a custom Python function, read_image_bbox, to the image and bbox inputs. This allows for the execution of non-TensorFlow operations within the TensorFlow computational graph.

- tf_dataset() : The tf_dataset function creates a TensorFlow dataset from images and their bounding boxes. It applies a parsing function to preprocess the data, batches it, and prefetches elements for efficient training. The function returns the processed dataset ready for use.

All above functions are chained together to form a pipeline which is used for training the model.

7. Model Training

After implementation of the pipeline, data is set to be incorporated. Now model training comes into action. Model training contains various things like hyperparameter optimization, callbacks definition, model building, model compiling, and fitting the data into the model.

- **Hyperparamters:**

Table 1: Hyperparameters

Hyperparameters	**Values**
Optimizer	Adam
Epoch	30
Learning rate	0.005
Batch size	16
Image Size	300 × 300
Dropout	0.2

- **Callbacks:**

1. ModelCheckpoint: Saves the model's weights during training, allowing for the best model to be saved based on validation loss.

2. ReduceLROnPlateau: Adjusts the learning rate if the validation loss plateaus, helping improve model performance.

3. CSVLogger: Logs training and validation metrics to a CSV file, providing a record of model performance.

4. EarlyStopping: Stops training if the validation loss does not improve within a specified number of epochs, helping prevent overfitting.

- **Loss Function and performance metric:**

1.Smooth L1 Loss: A robust loss function used in object detection to calculate the discrepancy between predicted and ground truth bounding box coordinates.

2.IoU (Intersection over Union) Metric: A performance metric in object detection that measures the overlap between predicted and ground truth bounding boxes, providing an evaluation of localization accuracy.

- **Model Building and compilation:**

Model is created with the given input shape and then the model is compiled with a custom loss function, smooth_l1_loss, and an Adam optimizer with a specified learning rate. After this data if feeded to model.

CHAPTER 4

RESULTS

4.1 RESULTS of YOLO V8:

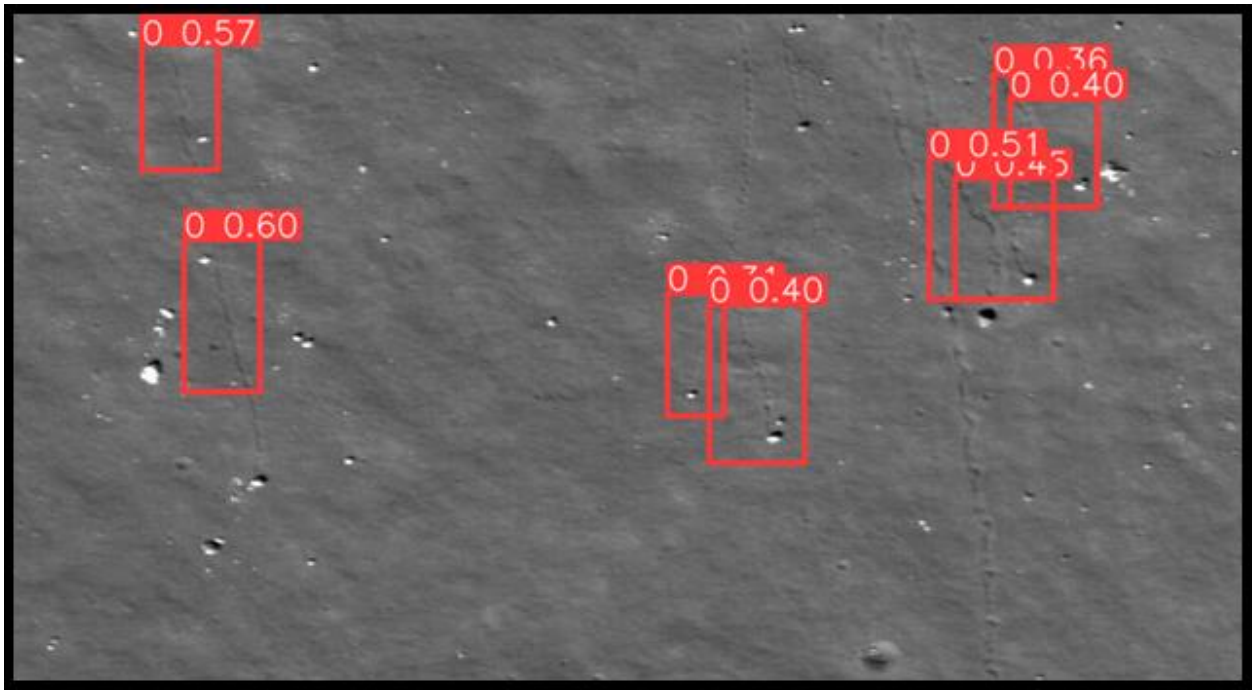


Figure 4: YOLOv8 Trained on Moon Surface Data for Rockfall Detection

4.2 Results of MobileNet V2:

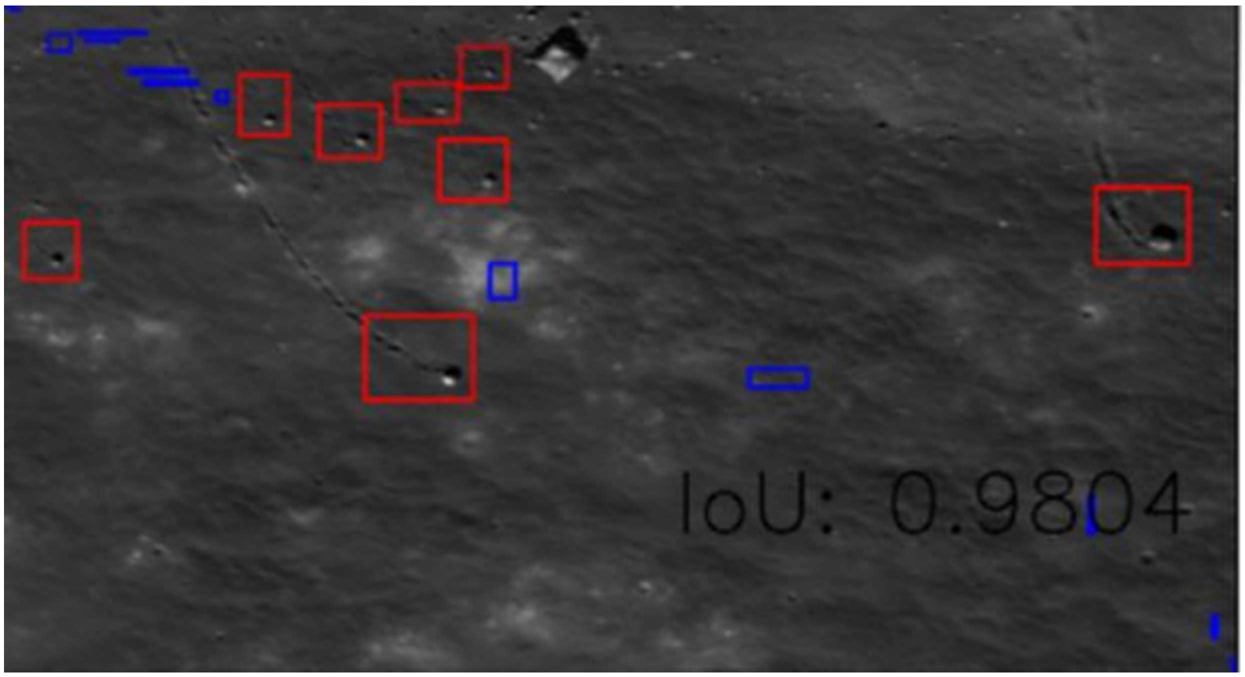


Figure 5: MobileNet V2 Trained on Moon Surface Data for Rockfall Detection

CHAPTER 5

CONCLUSION

In conclusion, this report investigates the use of MobileNet v2 and YOLO v8 object detection models for rock detection on the moon. Both models show impressive performance in accurately detecting rocks. MobileNet v2 excels in computational efficiency, making it suitable for resource-constrained lunar missions. YOLO v8 demonstrates higher precision and recall, beneficial for comprehensive rock detection. These findings contribute to lunar exploration, aiding in landing site selection and geological surveys. Integration of other advanced models and real-time data can further enhance performance in dynamic lunar environments. This study emphasizes the significance of object detection techniques in lunar exploration, advancing our understanding of lunar geology. The automated rock detection systems developed using MobileNet v2 and YOLO v8 have practical applications in future missions.

CHAPTER 6

REFERENCES

[1]https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

[2] https://learnopencv.com/ultralytics-yolov8/

[3] https://builtin.com/machine-learning/common-loss-functions

[4]https://vitalflux.com/mean-squared-error-vs-cross-entropy-loss-function/#:~:text=Mean%20squared%20error%20(MSE)%20loss%20is%20a%20widely%2Dused,to%20predict%20continuous%20numerical%20values.

[5] https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7#:~:text=Smooth%20L1%20Loss&text=It%20is%20less%20sensitive%20to,values%20result%20in%20exploding%20gradients.

[6] https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef

[7]https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[8]https://towardsdatascience.com/enhanced-object-detection-how-to-effectively-implement-yolov8-afd1bf6132ae

[9]https://www.fritz.ai/object-detection/#:~:text=Object%20detection%20is%20a%20computer,all%20while%20accurately%20labeling%20them.

[10]Rockfall detection on Moon (dataset):
https://www.kaggle.com/datasets/yash92328/rockfall-detection-on-moon