Machine learning

4.1 Machine learning

Machine learning is a subset of AI (Artificial intelligence) which allows a machine to learn from past data without programming explicitly or automatically. ML (Machine learning) can also be defined as solving a practical problem by gathering a dataset and algorithmically building a statistical model based on that dataset. That statistical model is assumed to be used somehow to solve the practical problem [8].

4.1.1 Advantage of Machine Learning:

There is an endless number of advantages of ML. We can look at the helpful ones. The advantages of Machine Learning tell us how using ML would benefit us. So, let us have a look at the advantages of ML [8].

- **Automation of Everything:** Machine Learning is responsible for cutting the workload and time. We let the algorithm do the hard work for us by automating things. The reason is that it is very reliable. Also, it helps us to think more creatively.
- **Wide Range of Applications:** ML has a wide variety of applications. This means that we can apply ML in any of the major fields. As a result, ML has its role everywhere, from medical, business, and banking to science and tech.
- **Scope of Improvement:** This helps us to improve both hardware and software. In hardware, we have various laptops and GPUs. These have various ML and DL networks in them. These help in the faster processing power of the system. Regarding software, we have various UIs and libraries in use. These help in designing more efficient algorithms.
- **Efficient Handling of Data:** Machine Learning has many factors that make it reliable. One of them is data handling. ML plays the most prominent role when it comes to data currently. It can handle any type of data

4.1.2 Type of Machine learning

There are three main types of machine learning, as illustrated in Figure below supervised, unsupervised, and reinforcement learning [8].

4.1.2.1 Supervised learning:

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labeled data. Even though the data needs to be labeled accurately for this method to work, supervised learning is potent when used in the right circumstances. The ML algorithm is given a small training dataset to work with in supervised learning. This training dataset is a minor part of the bigger dataset and gives the algorithm a basic idea of the problem, solution, and data points to deal with. The training dataset is also very similar to the final dataset in its characteristics and provides the algorithm with the labeled parameters required for the problem. The algorithm then finds relationships between the parameters given, establishing a cause-and-effect relationship between the variables in the dataset. At the end of the training, the algorithm learnt how the data works and the relationship between the input and the output.

There are two types of supervised machine learning:

- Classification: is the task of predicting a discrete class label.
- Regression: is the task of predicting a continuous quantity.

Some algorithms can be used for classification and regression with slight modifications, such as decision trees and artificial neural networks. On the other hand, some algorithms cannot or cannot easily be used for both problem types, such as linear regression for predictive regression modeling and logistic regression for classification predictive modeling.

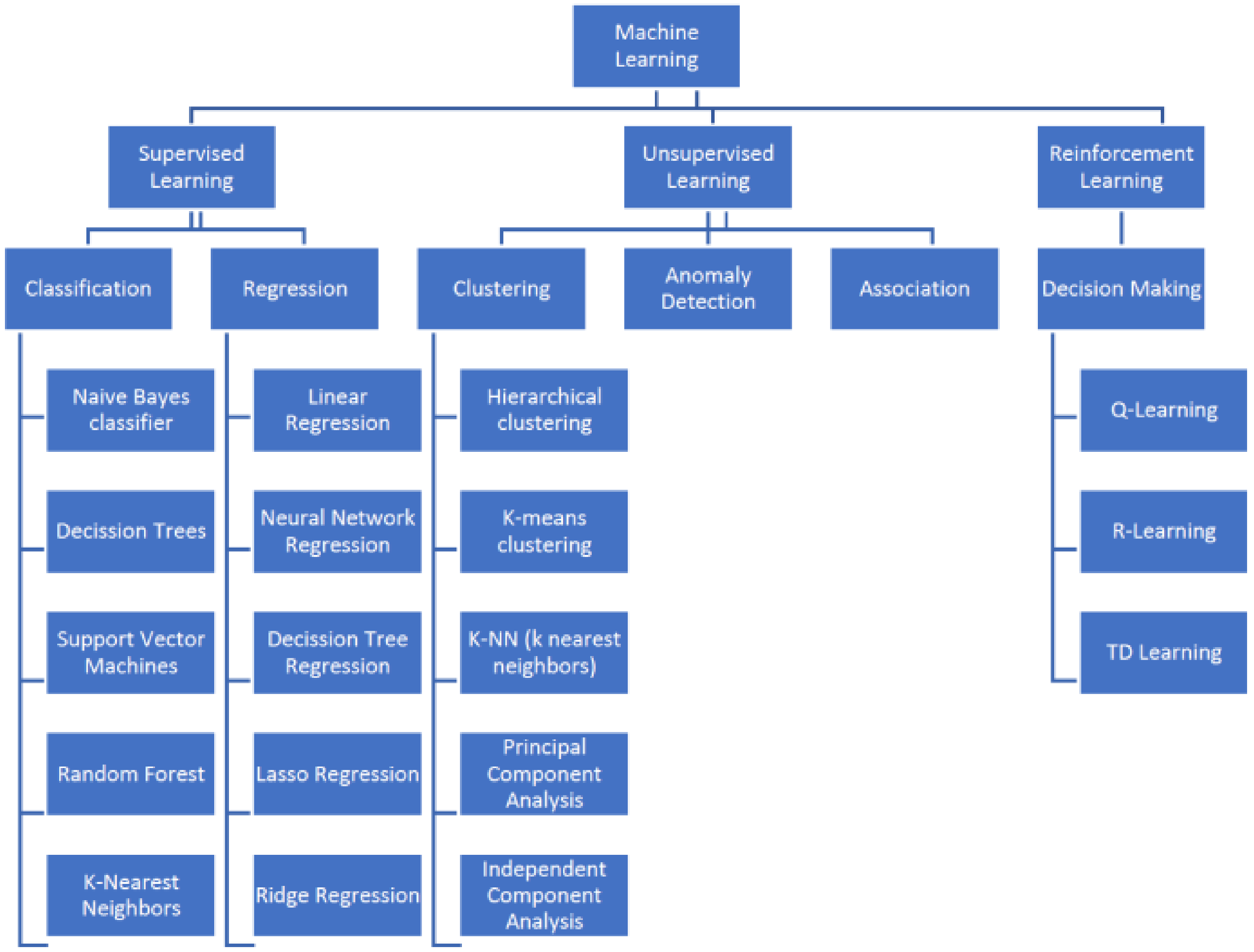


Figure 1: type of machine learning

4.1.2.2 Unsupervised learning

Unsupervised machine learning holds the advantage of being able to work with unlabeled data. This means that human labor is not required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program.

In supervised learning, the labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work off, creating hidden structures. As a result, the algorithm perceives relationships between data points abstractly, with no input required from human beings.

The creation of these hidden structures is what makes unsupervised learning algorithms versatile. Instead of a defined and set problem statement, unsupervised learning algorithms can adapt to the data by dynamically changing hidden structures. This offers more post-deployment development than supervised learning algorithms.

There are three types of unsupervised machine learning:

- **Clustering: mainly deals with finding a structure or pattern in a collection of uncategorized data. The number of clusters that the algorithm should identify can be modified.**
- **Anomaly Detection: It can discover unusual data points (Outliers) in your dataset. It helps find fraudulent transactions.**
- **Association: It identifies sets of items that often occur together in the dataset.**

4.1.2.3 Reinforcement learning:

Reinforcement learning directly inspires how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged, or reinforced, and non-favorable outputs are discouraged or punished.

4.1.3 Why is machine learning important?

Resurging interest in machine learning is due to the same factors that have made data mining and Bayesian analysis more popular than ever. Things like growing volumes and varieties of available data, computational processing that is cheaper and more powerful, and affordable data storage. All these things mean it's possible to quickly and automatically produce models that can analyze bigger, more complex data and deliver faster, more accurate results – even on a very large scale. And by building precise models, an organization has a better chance of identifying profitable opportunities – or avoiding unknown risks [6].

What's required to create good machine learning systems [7]?

- Data preparation capabilities.
- Algorithms – basic and advanced.
- Automation and iterative processes.
- Scalability.
- Ensemble modelling.

4.2 CV (computer vision)

4.2.1 What is computer vision?

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs and take actions or make recommendations based on that information [28]. If AI enables computers to think, computer vision enables them to see, observe and understand. Computer vision works much the same as human vision, except humans have a head start. Human sight has the advantage of lifetimes of context to train how to tell objects apart, how far away they are, whether they are moving and whether there is something wrong in an image. Computer vision trains machines to perform these functions, but it has to do it in much less time with cameras, data and algorithms rather than retinas, optic nerves and a visual cortex. Because a system trained to inspect products or watch a production asset can analyses thousands of products or processes a minute, noticing imperceptible defects or issues, it can quickly surpass human capabilities [9].

4.2.2 How does computer vision work?

Computer vision needs lots of data. It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects[30].

Two essential technologies are used to accomplish this: a type of machine learning called deep learning and a convolutional neural network (CNN).

Machine learning uses algorithmic models that enable a computer to teach itself about the context of visual data. If enough data is fed through the model, the computer will “look” at the data and teach itself to tell one image from another. Algorithms enable the machine to learn by itself, rather than someone programming it to recognize an image.

A CNN helps a machine learning or deep learning model “look” by breaking images down into pixels that are given tags or labels. It uses the labels to perform convolutions (a mathematical operation on two functions to produce a third function) and makes predictions about what it is “seeing.” The neural network runs convolutions and checks the accuracy of its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images in a way similar to humans.

Much like a human making out an image at a distance, a CNN first discerns hard edges and simple shapes, then fills in information as it runs iterations of its predictions. A CNN is used to understand single images. A recurrent neural network (RNN) is used in a similar way for video applications to help computers understand how pictures in a series of frames are related to one another[31].

4.2.3 RoboFlow

Roboflow is a computer vision platform that provides tools and services to help developers and businesses build and deploy computer vision models more easily and efficiently. The platform offers a suite of tools for data labeling, data preprocessing, model training, and deployment.

Roboflow supports a wide range of computer vision tasks, including object detection, image segmentation, classification, and more. It also provides a variety of pre-trained models that can be used as a starting point for your own projects, as well as tools for fine-tuning these models to better suit your specific needs. One of the key features of Roboflow is its data labeling tool, which allows users to annotate and label their images and videos quickly and accurately. This tool includes a variety of annotation types, including bounding boxes, polygons, and semantic segmentation masks.

In addition to its core features, Roboflow also integrates with popular machine learning frameworks and libraries, such as TensorFlow, PyTorch, and Keras. This integration allows developers to easily incorporate Roboflow into their existing machine learning workflows.

Overall, Roboflow is a powerful platform that can help developers and businesses streamline their computer vision projects and accelerate the development of their computer vision applications.

4.2.4 YOLO

YOLO (You Only Look Once) is a popular real-time object detection algorithm that can detect objects in images and videos with high accuracy and speed. YOLO divides an image into a grid of cells and predicts bounding boxes and class probabilities for each cell. The algorithm then applies non-max suppression to remove redundant detections and outputs the final set of bounding boxes and class probabilities.

The original YOLO algorithm was introduced in a paper by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in 2016. Since then, several versions of YOLO have been developed, including YOLOv2, YOLOv3, and YOLOv4. Each version has improved on the previous one in terms of accuracy and speed.

One of the key benefits of YOLO is its speed, as it can detect objects in real-time on a GPU. This makes it ideal for applications such as autonomous vehicles, Smart Biking system July/2023 78 | P a g e security cameras, and robotics. YOLO is also highly accurate, particularly for small and medium-sized objects, which can be challenging for other object detection algorithms.

While YOLO is a powerful algorithm, it does have some limitations. For example, it can struggle with detecting objects that are heavily occluded or have complex shapes. Additionally, YOLO is not optimized for detecting objects at extreme distances or in low-light conditions.

Overall, YOLO is a powerful object detection algorithm that has become widely used in computer vision applications due to its speed and accuracy.

4.2.4.1 Why the YOLO algorithm is important?

YOLO algorithm is important because of the following reasons:

- Speed: This algorithm improves the speed of detection because it can predict objects in real-time.
- High accuracy: YOLO is a predictive technique that provides accurate results with minimal background errors.
- Learning capabilities: The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

4.2.4.2 How the YOLO algorithm works?

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)
- Combination of the three techniques

4.2.4.2.1 Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S. The following image shows how an input image is divided into grids.

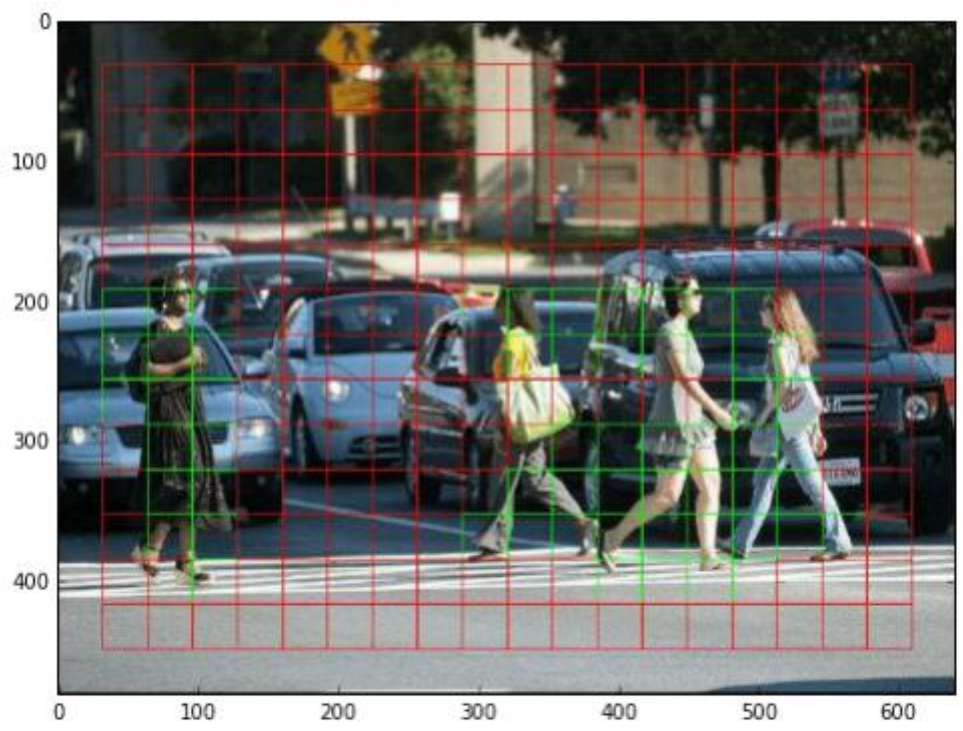


Figure :image divided into grids

In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

4.2.4.2.2 Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

- Width (b_w)
- Height (b_h)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.
- Bounding box center (b_x, b_y)

The following image shows an example of a bounding box. The bounding box has been represented by a yellow outline.

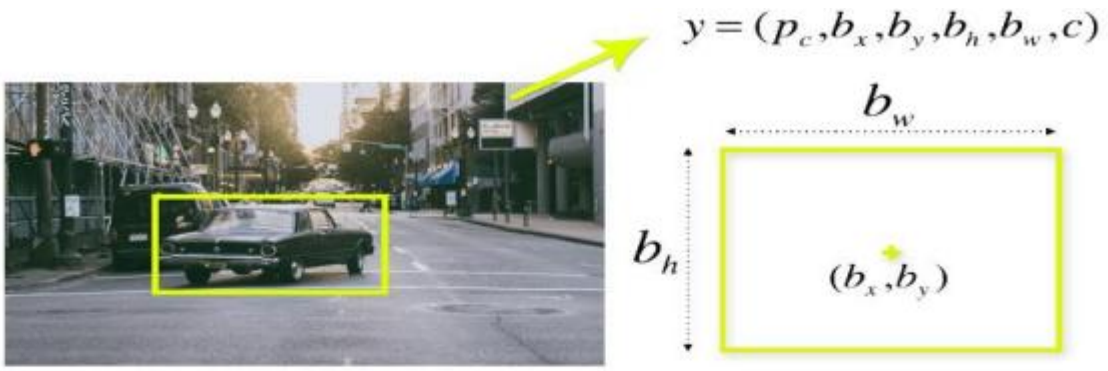


Figure Bounding box in image

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, represents the probability of an object appearing in the bounding box.

4.2.4.2.3 Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly. Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

The following image provides a simple example of how IOU works.

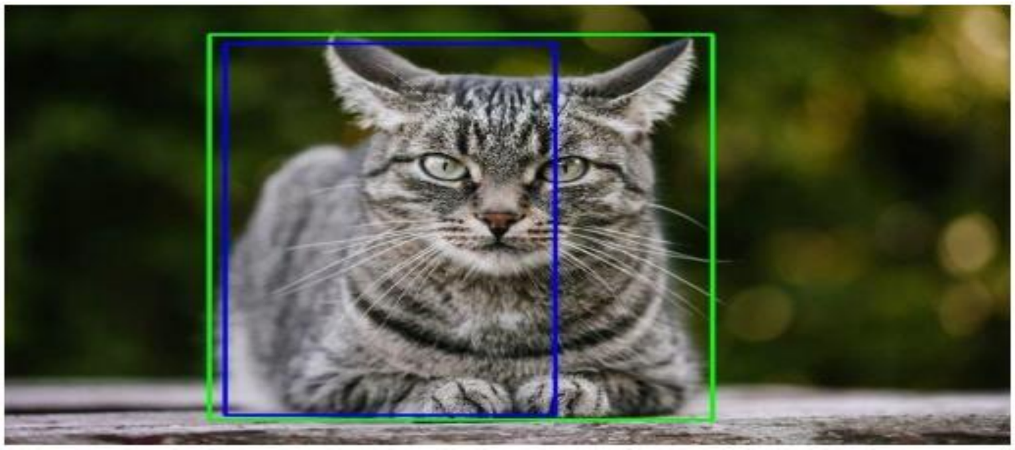


Figure Intersection over union

In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

4.2.4.2.4 Combination of the three techniques

The following image shows how the three techniques are applied to produce the final detection results.

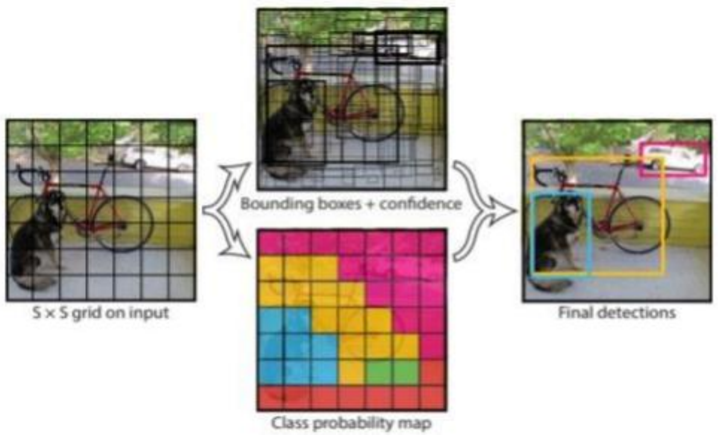


Figure combined three techniques

First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object.

For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single convolutional neural network.

Intersection over union ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly.

For example, the car is surrounded by the pink bounding box while the bicycle is surrounded by the yellow bounding box. The dog has been highlighted using the blue bounding box.

4.2.5 YOLOv5

YOLOv5 is a widely adopted deep learning model for real-time object detection, recognized for its high speed, accuracy, and ease of deployment. Although it was not officially released by the original YOLO authors, YOLOv5 has gained significant popularity due to its implementation in PyTorch, modular architecture, and active open-source development. The model incorporates advanced techniques such as auto-learning anchor boxes, mosaic data augmentation, and model quantization to enhance both performance and adaptability. YOLOv5 is particularly effective in various real-world applications including industrial inspection, autonomous navigation, and medical imaging analysis [1].

4.2.5.1 Library: Ultralytics YOLOv5

Ultralytics YOLOv5 is a highly optimized and modular deep learning library for real-time object detection, built using the PyTorch framework. As one of the most widely used versions of the YOLO (You Only Look Once) family, YOLOv5 offers a balance between speed and accuracy, making it suitable for deployment across a wide range of environments, including edge devices, embedded systems, and cloud-based platforms.

The architecture of YOLOv5 supports key innovations such as auto-anchor learning, mosaic augmentation, and model quantization, which enhance training efficiency and inference speed. Its user-friendly design and extensive documentation make it accessible to both novice and experienced practitioners in machine learning and computer vision.

The official Ultralytics YOLOv5 repository provides comprehensive resources, including pretrained models, tutorials, and API references, enabling users to fully leverage its capabilities in tasks such as object detection, tracking, and image analysis.

4.2.5.3 YOLOv5 Algorithm

YOLOv5 (You Only Look Once version 5) is a real-time object detection algorithm developed by Ultralytics, implemented in PyTorch. It processes entire images in a single forward pass, predicting bounding boxes and class probabilities simultaneously. YOLOv5 introduces several architectural enhancements over its predecessors, including the Cross Stage Partial (CSP) backbone and the Path Aggregation Network (PANet) for feature fusion. These improvements contribute to a balanced trade-off between detection accuracy and computational efficiency, making YOLOv5 suitable for deployment on resource-constrained devices [2].

Key features of YOLOv5 include:

- **CSP Backbone**: Enhances learning capability by partitioning feature maps and merging them through a cross-stage hierarchy.
- PANet: Facilitates efficient feature aggregation from different layers, improving object localization.
- **Mosaic Data Augmentation**: Combines four training images into one, enriching the dataset and improving generalization.
- **AutoAnchor**: Automatically calculates optimal anchor boxes, streamlining the training process.
- **Export Flexibility**: Supports conversion to formats like ONNX, CoreML, and TensorRT for diverse deployment scenarios.

These features collectively enable YOLOv5 to achieve high-speed and accurate object detection, making it a preferred choice for various real-world applications.

4.3 Programming used

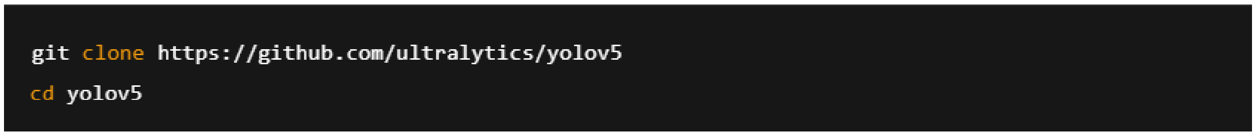
In order to process the captured data, we used two programming languages, which are python. Like a normal language, each programming language has a set of rules and a vocabulary that it uses. Processors, however, cannot understand them directly, thus a human-readable program must be translated into binary values that a computer or a computer-like device can comprehend it [3].

4.3.1 Python

Python is a programming language designed to make it simple for programmers of all levels to translate concepts into working code. Many developers use Python because it is able to express his/her thoughts in less code lines without sacrificing readability and also because now it is a well-developed, and well-supported programming language in the field of machine learning. It is favorable for most people because it has many great features such as fast prototyping, easy coding and free to use. In this project, we are using a python code that is added onto the Raspberry-Pi to detect the Detection and Lane Line Detection and Traffic Sign Detection [4].

4.3.1.1 Installing the Required Libraries for YOLOv5

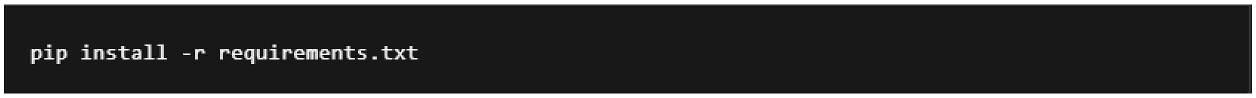
First, download the YOLOv5 code from GitHub:



```
git clone https://github.com/ultralytics/yolov5
cd yolov5
```

Figure clone our yolov5 files

Step 2: Install Required Libraries



```
pip install -r requirements.txt
```

Figure installing our libraries for our project

This will install essential libraries like PyTorch, OpenCV, NumPy, and others.

4.3.1.2 Python with OpenCv

OpenCV is a Python library that allows you to perform image processing and computer vision tasks. It provides a wide range of features, including detection, face recognition, and tracking. [5]

4.3.1.3 Thonny IDE

Thonny is an integrated development environment (IDE) that is lightweight and specifically designed for Python development. It is an excellent choice for beginners and for running machine learning projects, especially when working on platforms like Raspberry Pi.

Thonny provides features such as:

- **Code Analysis**: Thonny helps identify errors in your code, making it easier to debug.
- **Graphical Debugger**: The IDE includes a simple and user-friendly debugger for tracking down issues in the code.
- **Unit Testing**: Thonny supports unit testing, which helps you ensure that your code functions as expected.
- **Version Control Integration**: Thonny integrates with version control systems like Git, making it easy to manage project changes.

While it is less feature-rich than more heavyweight IDEs like PyCharm, Thonny is optimized for smaller, resource-constrained environments like Raspberry Pi. Its simple interface makes it a great fit for running YOLOv5 and other machine learning tasks on the Raspberry Pi.

Thonny is cross-platform and works on Microsoft Windows, macOS, and Linux. Its lightweight design is particularly advantageous when running machine learning models like YOLOv5 on Raspberry Pi, providing an efficient and straightforward development experience.

4.3.1.4 Google Colab

Google Colaboratory, or "Colab" as most people call it, is a cloud-based Jupyter notebook environment. It runs in your web browser (you can even run it on your favorite Chromebook) and lets anyone with internet access experiment with machine learning and coding for artificial intelligence. You can write and execute Python code, share your code and edit it simultaneously with other team members, and document everything by combining it into a single notebook with rich text, charts, images, HTML, and LaTeX.

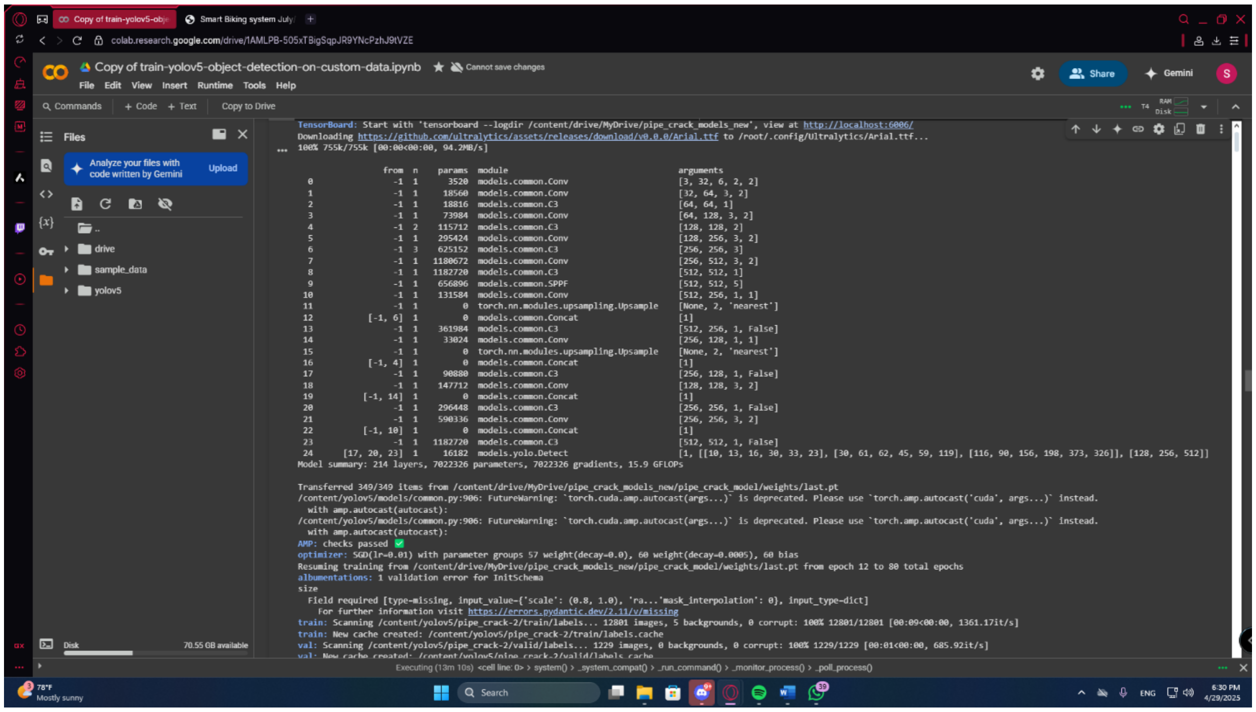
4.3.1.4.1 What can you do in Google Colab?

As a programmer, these are some of the things that are possible in Colab:

1. Write, execute, and share code in Python
2. Participate in real-time collaborative coding with your team
3. Connect with GitHub to import or publish notebooks
4. Import external datasets
5. Document code that supports mathematical equations
6. Access GPU and TPU runtimes for free
7. Use preinstalled libraries like TensorFlow, Matplotlib, PlyTorch, and other ML libraries
8. Integrate with GitHub
9. Use version history similar to Google Docs
10. Train models using images, audio, and text
11. Analyze and visualize data

4.3.1.4.2 Why we used google colab?

Python and Jupyter can have intensive CPU and GPU workload requirements. Colab gives you free access to computing infrastructure to test and execute your code. Like many of Google's products, there is a free tier and paid options. The free version of Colab is for students, hobbyists, and small experimental projects. As a data scientist or AI researcher, Google's paid plans offer more compute units, faster GPUs, access to higher memory machines, and terminal access with the connected virtual machine. If you want to learn about artificial intelligence and machine learning or have some simple Python code you want to document or experiment with, Colab requires no setup and is free to use. Google also offers various Colab Pro subscriptions that give paid users access to faster NVIDIA GPUs and compute credits for more advanced tasks. At the next figure we’ve trained the model of Roboflow YOLO V5 on google colab because of the enormous power of processing on GPUs and CPUs of google servers so it saved our time instead of using our personal computers power.


Copy of train-yolov5-object-detection-on-custom-data.ipynb
Cannot save changes
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Copy to Drive
Share
Gemini
Files
Analyze your files with code written by Gemini
Upload
drive
sample_data
yolov5
Disk
70.55 GB available
Executing (13m 10s) <cell line: 0> > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()

```
TensorBoard: Start with 'tensorboard --logdir /content/drive/MyDrive/pipe_crack_models_new', view at http://localhost:6006/
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 94.2MB/s]

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 18                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 24      [17, 20, 23]  1     16182  models.yolo.Detect                      [1, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
Model summary: 214 layers, 7022326 parameters, 7022326 gradients, 15.9 GFLOPs

Transferred 349/349 items from /content/drive/MyDrive/pipe_crack_models_new/pipe_crack_model/weights/last.pt
/content/yolov5/models/common.py:906: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with amp.autocast(autocast):
/content/yolov5/models/common.py:906: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with amp.autocast(autocast):
AMP: checks passed ✅
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
Resuming training from /content/drive/MyDrive/pipe_crack_models_new/pipe_crack_model/weights/last.pt from epoch 12 to 80 total epochs
albumentations: 1 validation error for InitSchema
size
  Field required [type=missing, input_value={'scale': (0.8, 1.0), 'ra...'mask_interpolation': 0}, input_type=dict]
    For further information visit https://errors.pydantic.dev/2.11/v/missing
train: Scanning /content/yolov5/pipe_crack-2/train/labels... 12801 images, 5 backgrounds, 0 corrupt: 100% 12801/12801 [00:09<00:00, 1361.17it/s]
train: New cache created: /content/yolov5/pipe_crack-2/train/labels.cache
val: Scanning /content/yolov5/pipe_crack-2/valid/labels... 1229 images, 0 backgrounds, 0 corrupt: 100% 1229/1229 [00:01<00:00, 685.92it/s]
val: New cache created: /content/yolov5/pipe_crack-2/valid/labels.cache
```

Figure our model training on google colab

4.4 Computer Vision Project Design

The machine learning part of the project uses a YOLOv5 model to detect cracks and damage in pipes. We trained the model on a dataset of pipe images that were labeled to show different types of defects. The training was done on a PC, and the final model was moved to the Raspberry Pi for real-time use.

The Raspberry Pi captures images from the camera, and the model checks each image for any cracks or damage. If any defects are found, the system saves the result with the type of defect and its location in the image. This helps in identifying problems inside the pipe without needing human inspection.

4.5 Model Output: Trained Weights and Inference Results

The final output of our training process is a set of optimized model weights saved in a file format compatible with YOLOv5 (typically .pt). These weights encapsulate the learned features from the labeled pipe dataset, enabling the model to accurately detect cracks and other structural damages in real-time.

After training the model using Google Colab, we exported the trained weights and deployed them on the Raspberry Pi. During real-time operation, these weights are used to make predictions on images captured by the PiCamera. The model processes each image frame, identifies the presence of defects, and returns:

- The bounding box coordinates
- The class label of the defect (e.g., "crack", "hole", "leak")
- The confidence score for each prediction

These predictions are then visualized using OpenCV, and any detected defects are logged for further analysis. This process allows for automatic, efficient, and accurate pipe inspection without manual review.

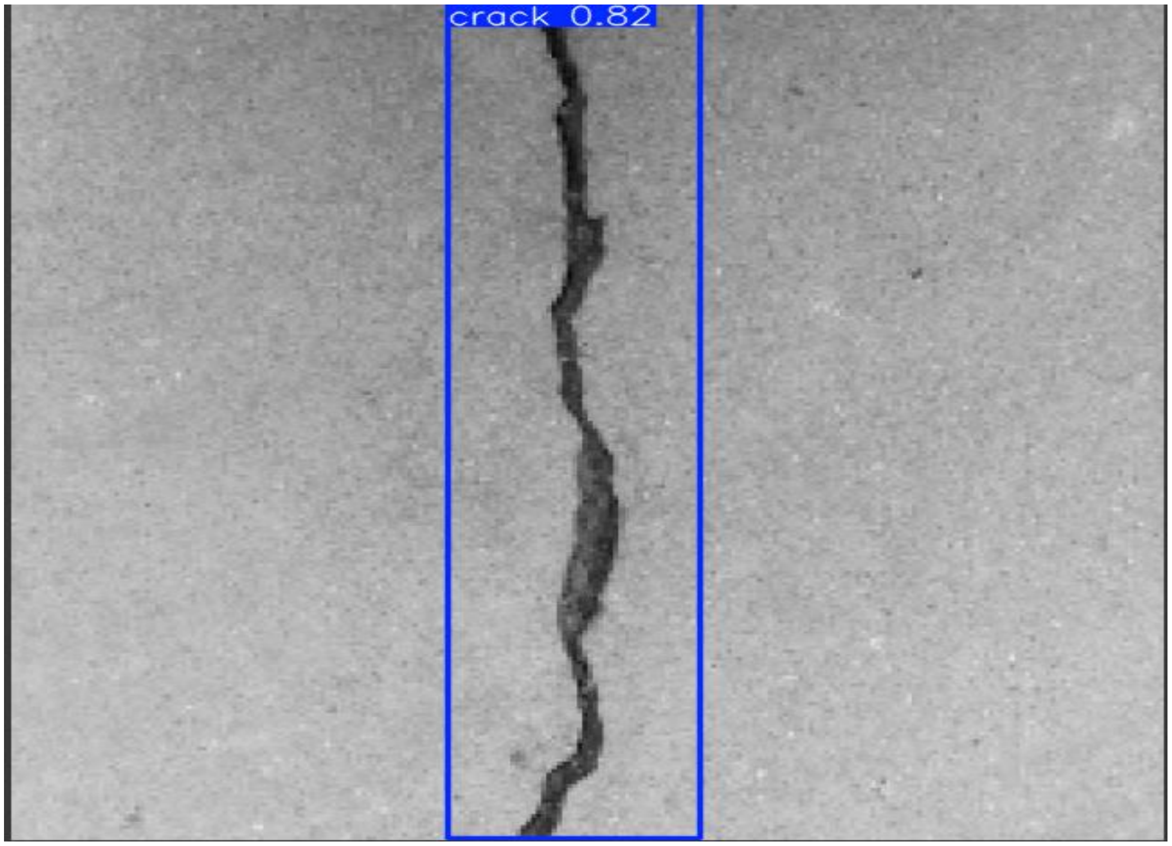


Figure model detect crack on image