



**Faculty of Computers &  
Artificial Intelligence**



**Benha University**

# **Disasters Detection in Industrial field**

**Using IOT and Machine Learning.**

A senior project submitted in partial fulfillment of the requirements for the  
degree of Bachelor of Computers and Informatics.

**Computer Science Department,**

## ***Project Team***

- 1- Hagar Nabil Eid
- 2- Khadija Serag Hamd
- 3- Mahmoud Basuni Mansour
- 4- Mahmoud Hamdy Mahmoud
- 5- Mahmoud Reda Farouk
- 6- May Esmail Mohamed

## ***Under Supervision of***

**Prof. Dr. Hala Zayed  
Dr. Mustafa Abdul-Salam  
Benha, Aug 2020**

## **Declaration**

We hereby certify that this material, which we now submit for assessment on the program of study leading to the award of Bachelor of Computers and Informatics in *computer science* is entirely our own work, that we have exercised reasonable care to ensure that the work is original, and does not to the best of our knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of our work.

**Signed:** \_\_\_\_\_

**Date:** TUE, Aug 2020.

## **Dedication**

Oil leakage and fire detection system is an important safety control and monitoring system in industry that gives the earliest possible warning of fire and oil leakage.

For our beloved country Egypt, this hard work to make it in the right way to progress for limiting these disasters.



## **Acknowledgment**

### **The contributions of people in our work are:**

- Prof. Dr/ Hala Zayed.
- Dr. Mustafa Abdul-Salam.
- Our supervisor Eng./ Mayada Mostafa.
- Our friends.
- Members of our team.

## **Abstract**

Disasters happen every day in our lives, especially in industrial field, which cause great damage and loss of lives. Like Fire disasters that cause distortion of the balance of earth ecosystem. Leakage of hazardous substances are one of the most frequent accidents in chemical installations and often generate serious damage to equipment, people who exposed to these disasters and the environment. Another significant impact is the interruption of the production process, including, in some cases evacuation of civilian areas located near it.

There are many types of industrial disasters that can cause many damages and hazards like:

Oil stains/leakage on a driveway or garage floor, this oil leak can cause fire hazard and can cause your vehicle to fail without warning and there is potential for injury to yourself and others. Another example, Fire disasters that can cause houses, trees, and many other things to burn into ashes. They can destroy a huge area in a matter of minutes. Every year people die by accident from fire. There are many other reasons like hazardous gases leakage...etc.

Our project Focus in two cases to find best solutions for them 1- Fire 2- oil leakage with real time detection for each case. We modify YOLOv3 algorithm and create our own dataset. In case of fire, we reach 88.75 % accuracy, 90.46% F1score of model with 1200 images dataset. In case of oil, we reach 80.7% accuracy, 96.5% F1score with 5000 images dataset. In addition, Apply and Evaluate Federated Learning with EMNIST dataset (250,000 images of handwritten digits) with 94 % accuracy for test set and 100 % accuracy for Train set to approve power of federated learning model in Strengthening Model accuracy from all users

# Table of Contents

## Table of Contents

<b>1</b>	<b>Chapter One: Introduction .....</b>	<b>1</b>
1.1	Motivation.....	1
1.2	Problem Statement .....	1
1.3	Objective .....	2
<b>2</b>	<b>Chapter Two: Background.....</b>	<b>3</b>
2.1	What is CV? .....	3
2.1.1	Applications of CV .....	4
2.1.1.1	Fire .....	5
2.1.1.2	Oil Leakage .....	6
2.2	Machine Learning .....	7
2.2.1	Applications of ML .....	7
2.2.2	Fields of ML .....	8
2.2.3	FL .....	9
2.3	What is YOLO? .....	9
2.3.1	What is YOLOv3? .....	11
2.4	Fire and Oil Leakage Detection Existing System.....	12
2.4.1	Fire Detection .....	12
2.4.2	Oil Leakage Detection .....	14
<b>3</b>	<b>Chapter Three: Methodology .....</b>	<b>15</b>
3.1	Hardware Setup.....	15
3.2	SOFTWARE TOOLS .....	15
3.3	YOLOv3 Classification Model Implementation .....	
3.3.1	Environment Setup.....	25
3.3.2	First Step: Dataset Preparation .....	25
3.3.3	Second Step: Training Model.....	27
3.3.4	Third Step: Evaluation Model .....	27
3.4	Federated Learning Algorithm .....	38
3.4.1	MNIST Dataset .....	38
3.4.2	TensorFlow Model .....	39
3.4.3	Evaluating Model .....	42
3.5	SYSTEM FEATURES .....	43
3.5.1	Technical Features.....	43
3.5.2	Non-Technical Feature .....	45
3.5.3	Used Tools .....	45
<b>4</b>	<b>Chapter Four: Conclusion.....</b>	<b>46</b>
4.1	FUTURE WORK .....	46
4.2	References to Electronic Sources .....	47

## List of figures

Figure 1-1 Fire Detection .....	2
Figure 2-1 Fire Detection .....	5
Figure 2-2 Oil Leakage .....	6
Figure 2-3 Oil Leakage .....	6
Figure 2-4 CNN .....	8
Figure 2-5 Federated Learning.....	9
Figure 2-6 YOLO .....	9
Figure 2-7 YOLO Detection .....	10
Figure 2-8 YOLO Accuracy .....	10
Figure 2-9 COCO Dataset.....	11
Figure 2-10 Fire Detection.....	14
Figure 3-1 YOLO Object Detection .....	16
Figure 3-2 YOLO Layers .....	16
Figure 3-3 Model Process .....	23
Figure 3-4 Label Image .....	24
Figure 3-5 Box Directions .....	31
Figure 3-6 Testing Fire .....	31
Figure 3-7 Testing Fire .....	31
Figure 3-8 Fire Dataset .....	31
Figure 3-9 Non-Fire Dataset .....	31
Figure 3-10 Oil Leakage Dataset .....	33
Figure 3-11 Oil Dataset testing .....	34
Figure 3-12 Non-Oil Dataset Testing .....	34
Figure 3-13 Federated Architecture .....	38
Figure 3-14 MINIST dataset.....	38

## **LIST OF ACRONYMS/ABBREVIATIONS**

CFG	Configuration (File Name Extension)
CNN	Convolution Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CV	Computer Vision
EMNIST	Enhanced Modified National Institute of Standards and Technology
FL	Federated Learning
FLOPS	Floating Point Operations per Seconds
FP	False Positive
FN	False Negative
GICA	Geometrical Independent Component Analysis
GPU	Graphical Processing Unit
HSI	Hue-Saturation-Intensity
LDS	Leak Detection Systems
LFRL	Lifelong Federated Reinforcement learning
MAP	Mean Average Precision
MIC	MethylIsoCyanate
ML	Machine Learning
R-CNN	Region Convolution Neural Network
RGB	Red-Green-Blue
TP	True Positive
TN	True Negative
VM	Virtual Machine
XML	Extensible Markup Language
YCbCr	Luminance; Chroma: Blue; Chroma: Red
YOLO	You Only Look Once



# **1 CHAPTER ONE: INTRODUCTION**

## **1.1 Motivation**

Disasters occur several times in the daily industrial fields, which may cause death and high cost damage, and it is mentioned that every twenty seconds of every work minute of every hour around the world, a person dies due to an industrial accident. In June 2019 – May 2020 ,where the most famous of the devastating fires in Australia left at least 28 people dead, about 3000 homes destroyed and up to a billion animals damaged, and one of the most important events in thirty years was thirty years ago, the worst industrial disaster in the world occurred, on Friday night. December 2, 1984, an accident at the Union Carbide pesticide plant in Bhopal, India, released at least 30 tons of a highly toxic gas called methylisocyanate, in addition to several other toxic gases. The pesticide plant was surrounded by shantytowns, exposing more than 600,000 people to the deadly gas cloud that night. The cause of the previous disaster was that water inadvertently entered the MIC storage tank, where more than 40 metric tons of MIC were stored. And cause an explosion. We can understand from this that early detection before disasters occur plays an important role for alerting about disaster before it occurs is very important to reduce losses.

## **1.2 Problem Statement**

In this regard, we will talk about the state of the problem. We searched for many works and projects using the detection system for many use cases and found that there are many projects around the world that use a fire detection system and oil leakage, which helps to reduce disasters that occur from oil spills or fire.

However, we found that most projects, alerting for the disaster happened after a period of time depending on eye vision system with monitoring or ordinary fixed camera with safety team monitor what happening in each section ;so we need to add more features to allow it to detect the real time of the disaster and alert the user.



Figure 1-1 Fire Problem

### 1.3 Objective

Our application aims to Save peoples and facilities from these kinds of disasters by reliable detection within seconds as early detection .Also plays a significant role in protecting the safety of emergency response personnel. while the fire is still small, property loss can be reduced and downtime for the operation minimized through early detection. High accuracy for each case detection with area covered restrictions . Get benefits from all places that use our system to reach high accuracy with preserving each facility's privacy

## **2 CHAPTER TWO: BACKGROUND**

### **2.1 What is CV?**

Even though early experiments in computer vision started in the 1950s and it was first use commercially to distinguish between typed and handwritten text by the 1970s, today the applications for computer vision have grown exponentially. By 2022, the computer vision and hardware market are expected to reach \$48.6 billion. It is such a part of everyday life you likely experience computer vision regularly even if you do not always recognize when and where the technology is deployed.

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, machines can accurately identify the detected objects and classify these objects and then react to what they “see”. From that we can identify Computer Vision in machine learning as it is the process of understanding digital images and videos using computers. It seeks to automate tasks that human vision can achieve. This involves methods of acquiring, processing, analyzing visual data and then makes decisions from it or gain understanding about the environment and situation, and understanding digital images, and extraction of data from the real world to produce information. One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better.

Along with a huge amount of visual data (more than 3 billion images are shared online every day), the computing power required to analyze the data is now accessible and more affordable. As the field of computer vision has grown with new hardware and algorithms so has the accuracy rates for object identification

Through a small period of time, today's systems have reached 99 percent accuracy from 50 percent making them more accurate than humans at quickly reacting to visual inputs.

Top 6 Computer Vision Techniques and algorithms that are widely used today:

- Image Classification.
- Object Detection
- Object Tracking.
- Semantic Segmentation.
- Instance Segmentation.
- Image Reconstruction.

### **2.1.1 Applications of CV**

- Autonomous vehicles
- Google Translate app
- Facial recognition
- Healthcare
- Real-time sports tracking
- Agriculture
- Manufacturing

### **2.1.1.1 Fire detection**

Fire detection technologies play important safety role in extreme environments. Cameras works much like a flame or smoke detector notifying the existence of fire in the very early stages of the ignition process. In addition, thermal cameras detect what are known as slow fires, extremely dangerous because they do not produce light emission. A security system with thermographic cameras allows rapid detection of any fire in isolated areas, far from populated areas and operating 24 hours a day.

If oil or chemicals are catching fire, the flames will often spread very quickly. With these conditions, any kind of firefighting can just lead to a controllable burning but not to a limitation of the damage. To avoid such situations, potential fire risks need to be identified before a fire outbreak occurs. To detect potential fire hazards at a very early stage, infrared cameras mean a reliable solution because they can continuously scan large areas.



**Figure 2-1 Fire Detection**



### **2.1.1.2 Oil Leakage**

Oil leakage is one of the most dangerous and expensive problem in many industries like petrol station, factories, and big plant. While most of the available Leak Detection Systems (LDS) can detect pipeline leaks, leak localization is still an unresolved problem, especially when the leakage is in a long pipeline that buried under the ground, then we discover many solutions that seek to invent and improve the existing leak localization and detection methods.



**Figure 2-2 Oil leakage**



**Figure 2-3 Oil leakage**

## **2.2 What is Machine Learning?**

Machine learning (ML) is a part of artificial intelligence, and is a study of computer algorithms that try to simulate the human brain learning ability, Machine learning algorithms can be used in various applications, such as spam filtering, computer vision, banking and many other apps. Machine learning can be useful in many situation where other algorithms can't solve the problem or take very long time solving it. Such as weather predictions that computed by a very complex calculations that would take longer than the day that we want to predict it's weather, or banking and stock market financial transactions that are very complex and change very rapidly, so machine learning algorithms are used to handle these type of difficult problems. The way the machine learning algorithms work is to train a model using a huge date, the data used to build the final model usually comes from multiple datasets, and the data are splitting to two parts the first part is training dataset. Training dataset is a dataset of examples used during the learning process and is used to fit the parameters, the training dataset is consists of pairs of an input vector (or scalar) and the corresponding output vector (or scalar), where the answer key is commonly denoted as the target (or label). The current model is run with the training dataset and produces a result, which is then compared with the target, for each input vector in the training dataset. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation. The second part is testing dataset. The testing dataset is a dataset that is independent of the training dataset, but that follows the same probability distribution as the training dataset. In our project we use machine learning algorithms in computer vision to detect and avoid industrial disasters, such as convolutional neural network (CNN).

### 2.2.1 What is CNN?

It is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs are regularized versions of multilayer perceptron's. Multilayer perceptron's usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer.

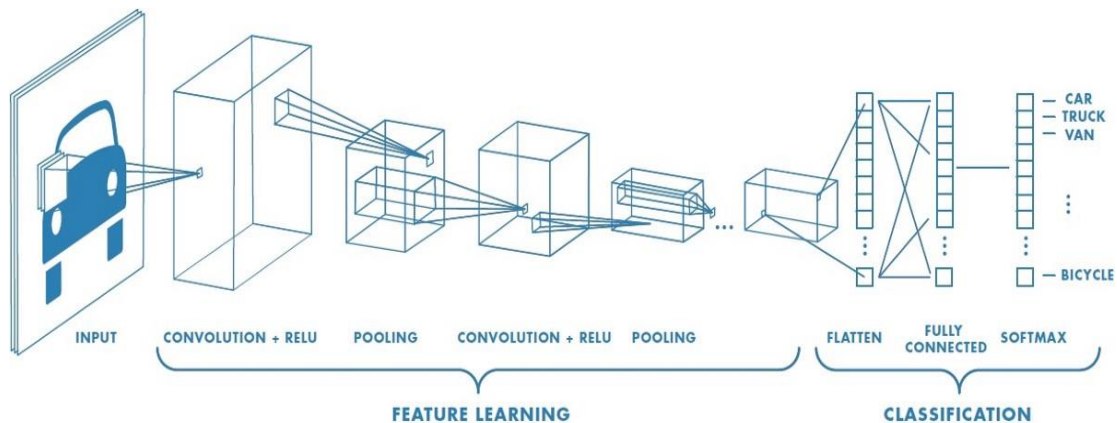


Figure 2-4 CNN

### 2.2.2 What is Federated Learning?

Federated learning as is a distributive machine learning framework developed for keeping privacy of the user data by learning only from models of the user in its local host leaving raw data and train and update a global model then resend it to user again appear in figure. federated learning is a reliable to be used in applications such as healthcare, energy demand prediction, fire detection with higher accuracy than traditional models using centralized learning also we can use LFRL (Lifelong Federated Reinforcement learning) to learn from more than one environment and more than one field and share learning together proven that it gives better performance. We can reduce the effect of other popular problems in update the model such as (uplink communication cost - bandwidth limitations) using different algorithm such as structured update and sketched update by



Compress the transmitted data and model size. We prove that federated learning is better than other traditional centralized algorithm. Although federated learning offer high level of privacy. There is a drawback that sometimes keeping privacy decrease the predication scalability. So we use other sections hoping helping solving this problem.

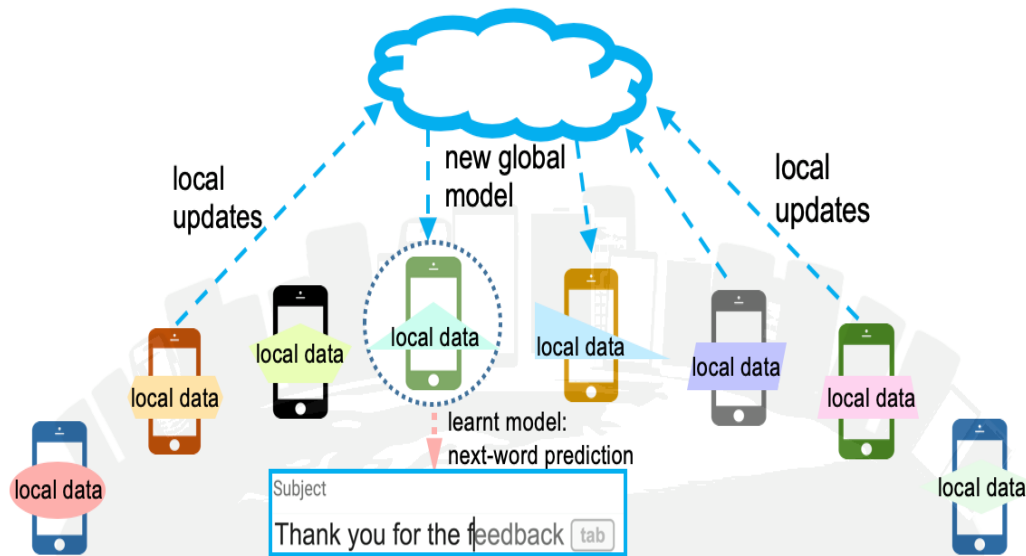


Figure 2-5 Federated Learning

### 2.3 What is YOLO?

YOLO (“You Only Look Once”) is an effective real-time object recognition algorithm.



Figure 2-6 YOLO

YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. The biggest advantage of using YOLO is its super speed, it's incredibly fast and can process 45 frames per second. Most of bounding boxes will not contain an object after detection, so we use the non-max suppression process that serves to remove boxes with low object probability and bounding boxes with the highest shared area in a process.

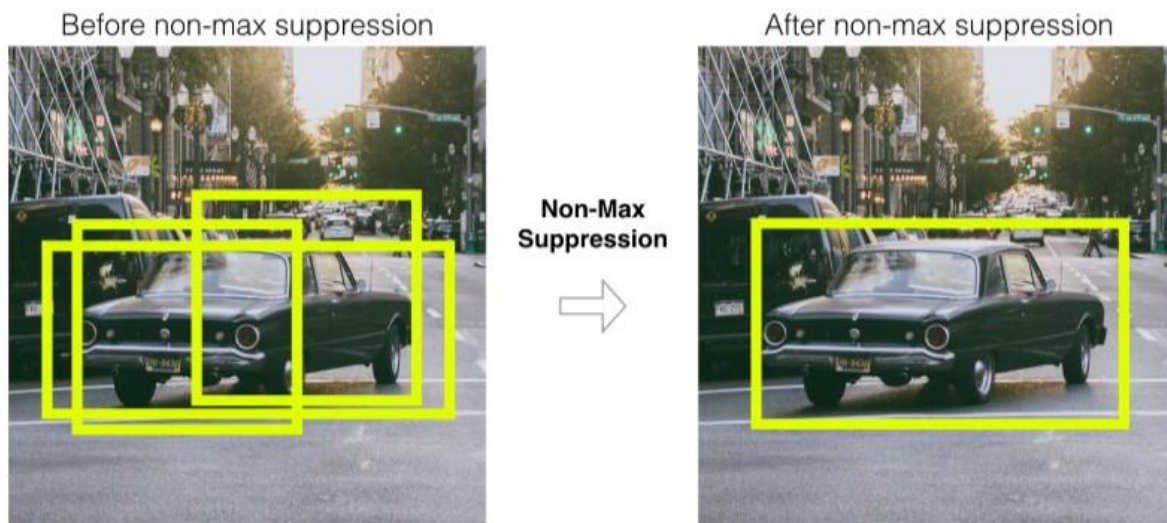


Figure 2-7 Yolo Detection

YOLO model use a huge dataset that called COCO dataset, and the model has about 57 % of this dataset that make it a very strong detection model. YOLO has many versions YOLOv1, YOLOv2 and YOLOv3.

Detection Algorithms	Accuracy (mAP)	Speed (FPS)
Fast R-CNN	70.0	0.5
Faster R-CNN w/ VGG-16 [26]	73.2	7
Faster R-CNN w/ ResNet [12]	76.4	5
YOLOv1	63.4	45
SSD300	74.3	46
YOLOv2 288 × 288	69.0	<b>91</b>
YOLOv2 416 × 416	<b>76.8</b>	67

Figure 2-8 Yolo Accuracy

### 2.3.1 What is new in version 3?

YOLOv3 uses a few tricks to improve training and increase performance, including: multi-scale predictions, a better backbone classifier and has more accuracy. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

Performance on the COCO Dataset.

Model	Train	Test	mAP	FLOPS
SSD300	COCO trainval	test-dev	41.2	-
SSD500	COCO trainval	test-dev	46.5	-
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn
SSD321	COCO trainval	test-dev	45.4	-
DSSD321	COCO trainval	test-dev	46.1	-
R-FCN	COCO trainval	test-dev	51.9	-
SSD513	COCO trainval	test-dev	50.4	-
DSSD513	COCO trainval	test-dev	53.3	-
FPN FRCN	COCO trainval	test-dev	59.1	-
Retinanet-50-500	COCO trainval	test-dev	50.9	-
Retinanet-101-500	COCO trainval	test-dev	53.1	-
Retinanet-101-800	COCO trainval	test-dev	57.5	-
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn

Figure 2-9 COCO Dataset

As we see YOLOv3 has the max FLOPS (floating point operations per second) and max mAP (mean average precision).

## 2.4 Fire and Oil Leakage Detection Existing Systems

### 2.4.1 Fire detection system

Fire is a very serious disaster, when a fire occurs, it seriously threatens people's lives and causes major economic losses, destroy the balance of earth ecosystem and worst of all they frequently cost human and animal live.

- **In 1658s**, New York's finest deployed men to walk around the streets looking for fires, with buckets on ladders and ringing bells to warn the community.
- **In 1800s**, fire alarms became a little more advanced with the placement of bell towers around cities to warn off people of a fire.
- **In 1852**, where it reached a new level of technology. Using the telegraph system, two alarm boxes with a telegraphic key were used to report neighborhood fires. One man would crank the handle that was attached to the box, releasing the key to send out a message to the central alarm station. The telegrapher at the central station would then send out the address of the location to the fire department.
- **By the late 1800s**, the electric fire alarm system was invented. This was the first time a thermostat could detect heat and trigger the sprinkler system to displace a fire. This was also the birth of fire protection services. As the protection services grew, so did the technology of the fire alarm system.
- **By the late 1990s**, these moves into the world of new technology took another significant step with the introduction of intelligent fire detectors, One of the systems used is an intelligent detectors that can analyze the signals from their

smoke or heat sensors and decide whether the source is likely to be smoke from a real fire or a false reason, such as cooking fumes; but in some situations it is difficult as one of the most devastating characteristics of a fire is its ability to spread, as in high ceilings are present, detecting a fire before it reaches a distant smoke detector can save valuable reaction time.

Just as important is the ability to correctly distinguish fire and smoke from other disturbances that trigger false alarms – and lead to financial ramifications. Another system that using fast, reliable detection is paramount by using computer vision algorithms like Otsu's method, Rayleigh distribution analysis algorithms. (modified segmentation algorithm) and NN algorithms to detect the fire and smoke indoor and outdoor also we can enhance the detection using coloring model rules like(RGB, HIS, GMM, GCIA and YCbCr) that make a higher affect in the result accuracy and make it easier to find the fire in morning and night.

- **Todays,** The biggest immediate trend in the industry is the move towards better detection, and better differentiation by using the correct system and solving other systems problems. This fire detection system will be more and more reliable and will emphasize reducing the losses by real-time fire detection and designed to minimize the risk of false alarms as our system introduce taking a video as an input from camera-based fire monitoring system can monitor the specified area in real time through video processing with combine motion detection based on frame difference and from the training dataset analyzing the input video and detecting the fire in a real-time with the addressable head that display at the control panel that indicated what zone was being affected. With the addressable head, the location can be pinpointed directly.





Figure 2-10 Fire Detection

### 2.4.2 Oil Leakage system

The United States bears fuel losses of \$10 billion annually due to leaks and thefts. The companies, operating in the midstream segment of the oil and gas industry, foresee significant upside in improving pipeline safety and reliability. The statistics measured that around 2.5 million miles of pipelines in the USA have an average of 93 leaks per mile.

The world relies on the Oil and Gas Industry heavily as the industry has contributed more than 61% in global energy production.

- Leaks were a pressing problem for Sextus Julius Frontenus, At that time, the only form of oil leak detection was by examining pipes suspected of being tapped, the only method of detecting oil leaks was by visual inspection. oil bursts are obvious and easy to find, but background losses often begin as pin-point sized leaks which remain undetectable until they become large enough to draw attention
- in 1800, Thermal Imaging for oil Leak Detection Infrared imaging was adapted during the 1950s and 1960s to produce line images. The invention of digital technology made it possible to take photos of a section of wall or roof and then use infrared to locate isothermal images on the photographs. This enables the identification of oil leaks without the use of expensive, bulky equipment. In 1879, Professor A. M. Mayer invented the “topophone,” a device that helped the oil to identify the source of any sound.
- They are now used to detect the sound of escaping oil, which creates an acoustic signal as it passes through holes and cracks in the pipes. This method relies on having a baseline acoustic fingerprint taken when the pipe is undamaged.

## **3 CHAPTER THREE: METHODOLOGY**

### **3.1 Hardware Setup**

- Laptop
- Projector
- Obelisk

### **3.2 SOFTWARE TOOLS**

- Unity engine
- Mono develop
- Blender

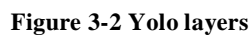
### **3.3 YOLOv3 Classification Model Implementation**

We faced some problems because there is no enough material and little knowledge about algorithms available in object recognition so we search with the aid of our DRs who suggest to work with YOLO or (“You Only Lock Once”) which is one of the best object detection algorithm and is an open source, that’s mean that anyone can make his YOLO. YOLO depend on CNN in the training process. YOLO is fast, has high accuracy and work in real time. YOLOv3 is an updated version of YOLO can be used to classify objects and give more accurate results. In industries there is a lot of objects that may fall or leak some of them are very important and have a high economic cost, other may be chemical material which are very dangerous (toxic or flammable) so we need a way to detect this danger as fast as we can.

## YOLO: You Only Look Once



### Figure 3-1 YOLO Object Detection





### 3.3.1 Environment setup

- **Development tool**

- Anaconda
- Colab

- **Framework**

- Darknet

- **Libraries**

- Keras
- Opencv
- Pytorch
- Numpy

- **Anaconda**

Anaconda is a free and open-source distribution of the Python for (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

- **Colab**

Colaboratory, or "Colab" for short, is a development tool designed by Google allows you to write and execute Python in your browser, with Zero configurations required, free access to GPUs and Easy sharing, whether you're a student, a data scientist or an AI researcher, Colab can make work easier.

- **Darknet**

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation, and is necessary to run YOLO Algorithm.

- Download and install the darknet on colab

```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet

# change makefile to have GPU and OPENCV enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

# verify CUDA
!/usr/local/cuda/bin/nvcc -version

# make darknet (build)
!make
```

## Uploading Google Drive Files to Use

This step will show you how to upload Google Drive files to the cloud VM.

We will be creating a symbolic link between '/content/gdrive/My\ Drive/' and '/mydrive'.

This means we are just creating a shortcut '/mydrive' to map to the contents within the folder '/content/gdrive/My\ Drive/'.

The reason for this is that sometime having the space in 'My Drive' folder path can cause issues when running certain commands. This symbolic link will stop this from happening!

Now you can run YOLOv3 with images from Google Drive using the darknet command:

```
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
# this creates a symbolic link so that now the path /content/gdrive/My\ Drive/ is equal to /mydrive
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive
```

### 3.3.2 Data Preparation

Data Preparation is the most important part to the training process as the pre-processing is an improvement of the image data that suppresses unwilling distortions or enhances some image features important for further processing.

Preparation data takes many steps to be ready for the training:

- **Gathering the data**

Collecting data from many resources to achieve the best training for the data for the result is important. We collect the data from GitHub and many other resources for:

- **Fire dataset**

- Fire dataset collecting from many resources as GitHub [FireNET-master] and Kaggle [Fire dataset] and many different resources.
    - Dataset was about 1200 image that dividing for
      - 80% for training data [960 image].
      - 20% for the testing data [240 image].

- **Oil leakage dataset**

- We face problem with collecting oil leakage dataset so we make our own dataset depending on recording videos from different places with different situations which is a self-created diverse dataset with images randomly sampled from our self-shot oil.
    - Dataset was about 5000 image that dividing for
      - 80% for training image [4000 image].
      - 20% for the test data [1000 image].

- **split videos into image or frame:**

We face a big problem that we couldn't find any oil leakage dataset, so we create our dataset.

First, we shoot a video of our chemicals on different backgrounds.

Second, we use an algorithm to split our video into frames.

Third, we delete some similar frames that may affect training of our model.

Fourth, split the dataset that we create into two parts of images, 80% of images for training, and 20% of images for testing.

And there is the algorithm we use to split the video into frames:

```
import cv2
import numpy as np
import os

# Playing video from file:
cap = cv2.VideoCapture('Video.mp4')
```

Here we import some library for the algorithm and put the name of the video that we want to split it into frames.

```
try:
    if not os.path.exists('data'):
        os.makedirs('data')
except OSError:
    print ('Error: Creating directory of data')
```

Then we check if the folder or directory that the algorithm will save the frames into exists or not. If not, the algorithm will create a folder and name it 'data'.

```
currentFrame = 0
while(True):
    ret, frame = cap.read()

    name = './data/f' + str(currentFrame) + '.jpg'
    print ('Creating...' + name)
    cv2.imwrite(name, frame)

    currentFrame += 1
```

Here we start the number of created frames by zero. Then tell him to capture frame by frame, save image of the current frame in jpg file, and every frame increase one to the name of current frame to avoid duplicating images.

```
cap.release()  
cv2.destroyAllWindows()
```

When everything done, release the capture.

```
(C:\Users\m7amd\Anaconda3) C:\Users\m7amd\OneDrive\Desktop\SplitProj>python split-video-by-frame.py  
Creating.../data/frame0.jpg  
Creating.../data/frame1.jpg  
Creating.../data/frame2.jpg  
Creating.../data/frame3.jpg  
Creating.../data/frame4.jpg  
Creating.../data/frame5.jpg  
Creating.../data/frame6.jpg
```

While running the splitting process and creating the frames.

- **Data Augmentation**

It is an integral process in deep learning, as in deep learning we need large amounts of data and in some cases it is not feasible to collect thousands or millions of images, so data augmentation comes to the rescue.

It helps us to increase the size of the dataset and introduce variability in the dataset. To make augmentation we need to do these **operations**:

1. *Rotation*
2. *Shearing*
3. *Zooming*
4. *Cropping*
5. *Flipping*
6. *Changing the brightness level*

We make Data augmentation using **Augmentor** by this code:

```
# Importing necessary library
import Augmentor
# Passing the path of the image directory
p = Augmentor.Pipeline("image_folder")

# Defining augmentation parameters and generating 5 samples
p.flip_left_right(0.5)
p.black_and_white(0.1)
p.rotate(0.3, 10, 10)
p.skew(0.4, 0.5)
p.zoom(probability = 0.2, min_factor = 1.1, max_factor = 1.5)
p.sample(5)
```

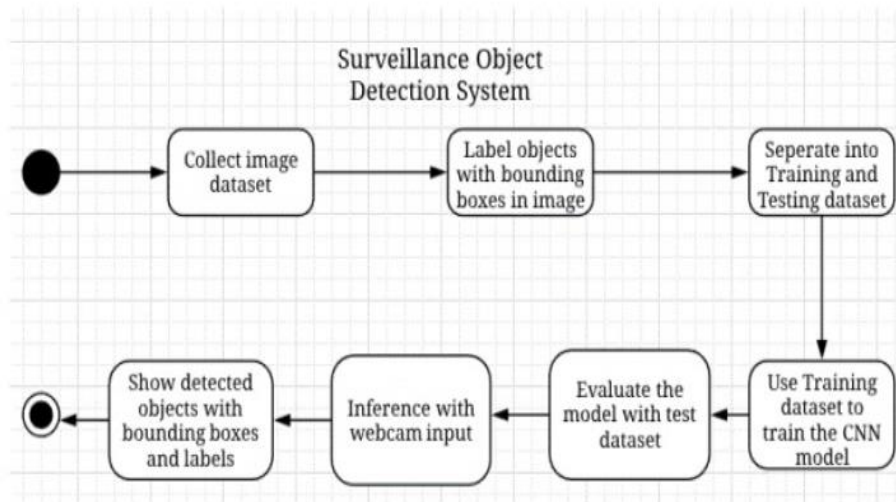
The above code snippet allows you to generate 5 augmented images

Ex : for **oil dataset** we enter 920 images and out 4600 augmented images (4000 for training , 600 for testing oil ) and add 400 images for test non-oil, thus total dataset is 5000 images.

- **Reduce dataset**

It is useful for the training process to remove irrelevant data for a good result. In terms of machine learning, choosing assumed or approximated objects must be “more right” for your dataset.

*This is a diagram for describe the process from the beginning:*



**Figure3-3 Model Process**

- **Labeling dataset**

- Labeling each image in the dataset to define each object in the dataset using “Labeling” software
- Drawing a bounding box around the object then write its class name belongs to the object.
- Saving image with its annotation with extension of (.txt ) containing its class number and the directions for the drawing rectangle that will be used later for the training process.

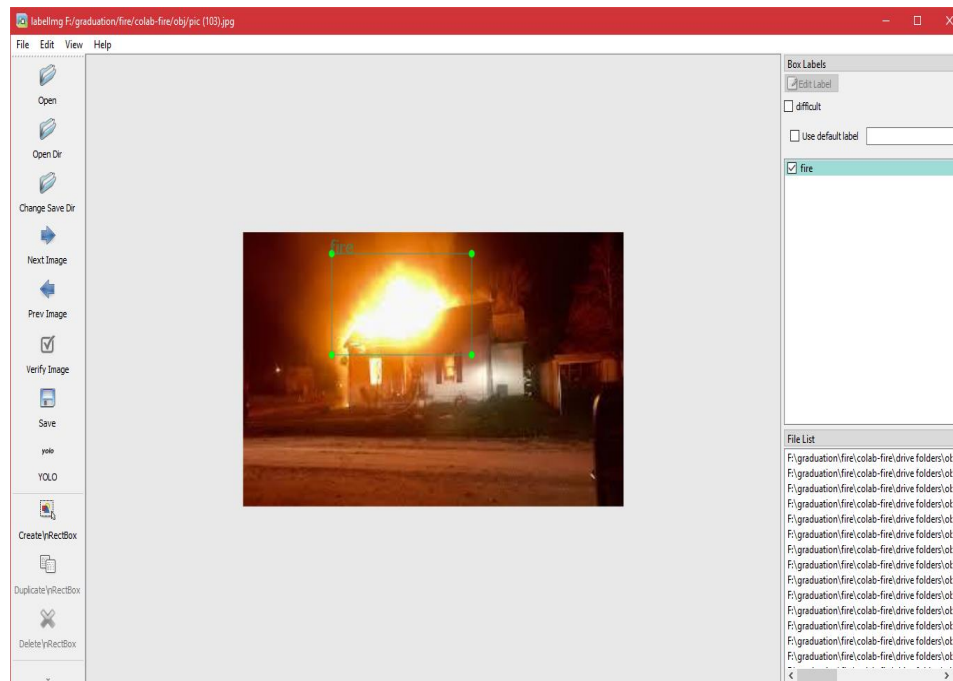


Figure 3-4 Labeling images

0 0.579866 0.393861 0.178523 0.294485

Figure 3-5 Box Direction

## Finally

Data has been prepared for the next process and it is ready for the training process.



### 3.3.3 Training Model

#### ❖ *Training a Custom YOLOv3 Object Detector in the Cloud:*

In order to create a custom YOLOv3 detector we will need the following:

- Labeled Custom Dataset
- Custom .cfg file
- obj.data and obj.names files
- train.txt file (test.txt is optional here as well)

#### **Step 1: Gathering and Labeling a Custom Dataset**

when we have successfully generated a custom YOLOv3 dataset and make Manually Labeling Images with Annotation Tool.

#### **Step 2: Moving Your Custom Dataset Into Your Cloud VM**

So now that we have our dataset properly formatted to be used for training we need to move it into this cloud VM so that when it comes the time we can actually use it for training. Then we renaming the folder with our images and text files on our local machine to be called '**obj**' and then creating a .zip folder of the 'obj' folder. Then we upload the zip to our Google Drive. So we should now have obj.zip someplace in our Google drive. Now we can copy in the zip and unzip it on your cloud VM.

```
# this is where my zip is stored (I created a yolov3 folder where I will get my required files from)
!ls /mydrive/yolov3

# copy the .zip file into the root directory of cloud VM
!cp /mydrive/yolov3/obj.zip ../

# unzip the zip file and its contents should now be in /darknet/data/obj
!unzip ../obj.zip -d data/
```

### Step 3: Configuring Files for Training

This step involves properly configuring your custom .cfg file, obj.data, obj.names and train.txt file.

```
# download cfg to google drive and change its name
!cp cfg/yolov3.cfg /mydrive/yolov3/yolov3_custom2.cfg

# to download to local machine (change its name to yolov3_custom.cfg once you download)
download('cfg/yolov3.cfg')
```

Now you need to edit the .cfg to fit your needs based on your object detector.

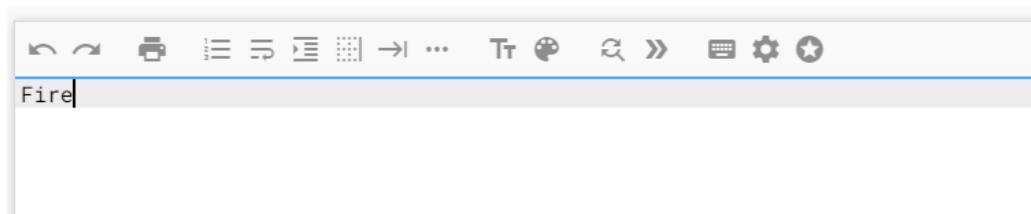
We having batch = 64 and subdivisions = 16 for ultimate results, set my max\_batches = 10000, steps = 8000, 9000, I changed the classes = 5 in the three YOLO layers and filters = 30 in the three convolutional layers before the YOLO layers, in each of the three Yolo layers in the cfg, we change one line from random = 1 to random = 0 to speed up training but slightly reduce accuracy of model.

```
# upload the custom .cfg back to cloud VM from Google Drive
!cp /mydrive/yolov3/yolov3_custom.cfg ./cfg

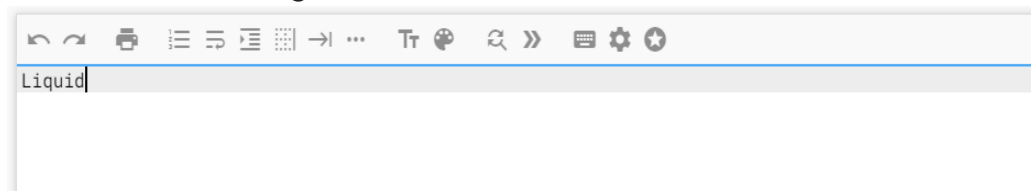
# upload the custom .cfg back to cloud VM from local machine (uncomment to use)
#%cd cfg
#upload()
#%cd ..
```

- 1- Then we create a new file within a code or text editor called **obj.names** and this file exactly the same as our class.txt in the dataset generation step.

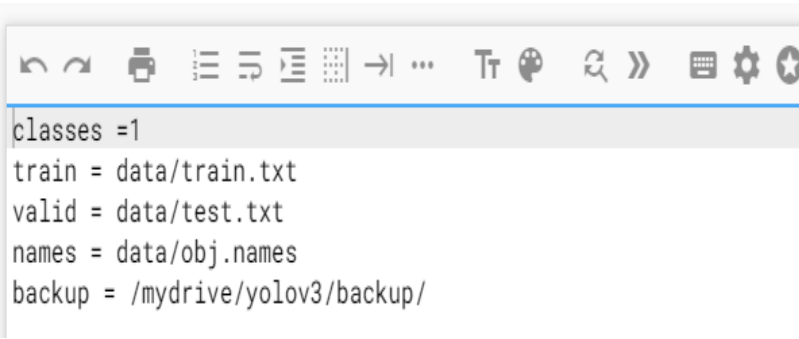
- In fire detect :



- In oil leakage detect :



- 2- we create a new file within a code or text editor called **obj.data** file and fill it in like this



```
classes = 1
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/yolov3/backup/
```

*Then*

```
# upload the obj.names and obj.data files to cloud VM from Google Drive
!cp /mydrive/yolov3/obj.names ./data
!cp /mydrive/yolov3/obj.data ./data

# upload the obj.names and obj.data files to cloud VM from local machine (uncomment to use)
#%cd data
#upload()
#%cd ..
```

Now we simply run the python script to do all the work for us.

```
!python generate_train.py
```

```
# verify train.txt can be seen in our darknet/data folder
!ls data/
```

#### Step 4: Download pre-trained weights for the convolutional layers.

This step downloads the weights for the convolutional layers of the YOLOv3 network. By using these weights it helps our custom object detector to be way more accurate and not have to train as long.

```
# upload pretrained convolutional layer weights
!wget http://pjreddie.com/media/files/darknet53.conv.74
```

### Step 5 & Finally: Train our Custom Object Detector

- 1- To avoid this hold (CTRL + SHIFT + i) at the same time to open up the inspector view on your browser.

Paste the following code into our console window and hit **Enter**

```
function ClickConnect(){  
  console.log("Working");  
  document.querySelector("colab-toolbar-button#connect").click()  
}  
setInterval(ClickConnect,60000)
```

- 2- We then run this code

```
# train your custom detector  
!./darknet detector train data/obj.data cfg/yolov3_custom.cfg darknet53.conv.74 -dont_show
```

- 3- If for some reason we get an error or our Colab goes idle during training, we have not lost your weights! Every 100 iterations a weights file called yolov3\_custom\_last.weights is saved to mydrive/yolov3/backup/ folder, Then we run this code.

```
!./darknet detector train data/obj.data cfg/yolov3_custom.cfg /mydrive/yolov3/backup/yolov3_custom_last.weights -dont_show
```

### 3.3.4 System Evaluation

Using new data when evaluating our model to prevent the likelihood of overfitting to the training set.

#### Fire Dataset

- Our testing dataset that acted as 20% of the dataset.
- Then, we divided the testing dataset into 60% for the same class ,40% for other images do not belong to the class.
- Evaluating the confusion matrix for the accuracy with
  - Accuracy

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

- Precision is defined as the fraction of relevant examples (true positives) among all the examples which were predicted to belong in

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

a certain class

- Recall is defined as the fraction of examples which were predicted to belong to a class with respect to all the examples that truly belong in the class.

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- F1-Score is defined to combine the precision and recall metrics.

$$F_{\beta} = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

## **Steps for the testing the dataset using the cfg**

- I. set our custom cfg to test mode

```
# need to set our custom cfg to test mode
%cd cfg
!sed -i 's/batch=64/batch=1/' yolov3_custom.cfg
!sed -i 's/subdivisions=16/subdivisions=1/' yolov3_custom.cfg
%cd ..
```

- II. Testing an image by uploading it from google drive and then re-upload the detected images again to your Google Drive, display the detected image.

```
[ ] # run your custom detector with this command (upload an image to your google drive to test, thresh flag sets accuracy that detection must be in order to show it)
!./darknet detector test data/obj.data cfg/yolov3_custom.cfg /mydrive/yolov3/backup/yolov3_custom_last.weights /mydrive/images/1.jpg -thresh 0.3
imshow('detection1.jpg')
```

### **For the Fire Dataset:**

- Our testing dataset consists of 240 images that acted as 20% of the dataset.
- Then, we divided the testing dataset into:
  - 60% for Fire data as 144 images:
    - True positives are 128 images.
    - False negatives are 16 images.
  - 40% for Normal data (Non-Fire) as 96 images:
    - True negatives are 85 images.
    - False positives are 11 images.

The result will be the detected image as shown in this figure:



Figure 3-6 Testing result



Figure 3-7 Testing result

III. Calculating the number of the images needing for the matrix by collecting the images from the drive being saved from the previous step.

- **For Fire dataset testing:**

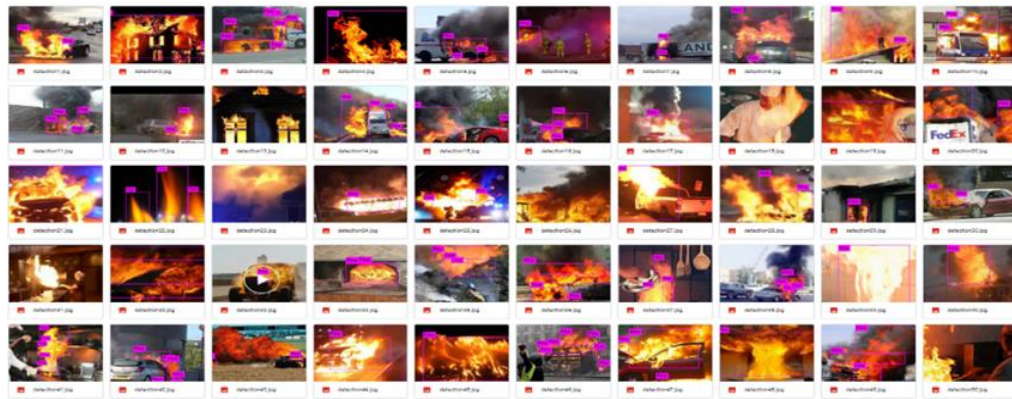


Figure 3-8 Fire dataset

- **For Non-fire dataset testing:**

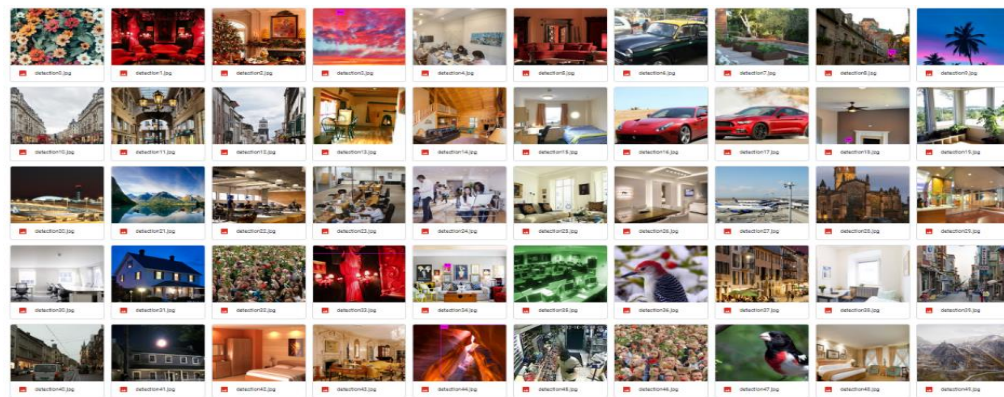


Figure 3-9 Non-Fire dataset

IV. Constructing the confusion matrix with parameters calculated from the previous step as this:

CM		Predicted	
		1	0
Actual	1	128	16
	0	11	85

**From the previous matrix calculating the following:**

- Accuracy =  $(TP+TN)/(TP+FP+FN+TN)$   
 $= (128+58) / (128+11+16+85) = 88.75\%$
- Precision =  $TP/(TP+FP)$   
 $= 128 / (128+11) = 92.09\%$
- Recall =  $TP/(TP+FN)$   
 $= 128 / (128+16) = 88.89\%$
- F1Score =  $2*((Precision*Recall) / (Precision + Recall))$   
 $= 2*((92.09*88.89) / (92.09+88.89))$   
 $= 90.46\%$

**Finally**, we obtained enough good result accuracy according to the minimized dataset with **88.75% accuracy**.



### **Oil Dataset**

We worked iteratively with different number of the dataset to enhance the accuracy so firstly:

- Our testing dataset consists of 146 images that acted as 20% of the dataset.
- Then, we divided the testing dataset for:
  - 60% for Oil data as 88 images:
    - True positives are 88 images.
    - False negatives are 0 images.
  - 40% for Normal data (Non-Oil) as 58 images:
    - True negatives are 36 images.
    - False positives are 22 images.

The result will be the detected oil image as shown in this figure:

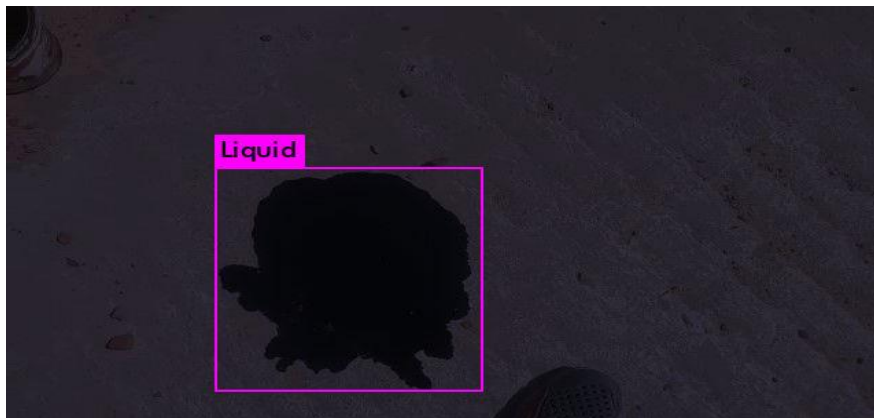


Figure 3-10 Oil leakage detection

- V. Calculating the number of the images needing for the matrix by collecting the images from the drive being saved from the previous

- **For Oil dataset testing:**

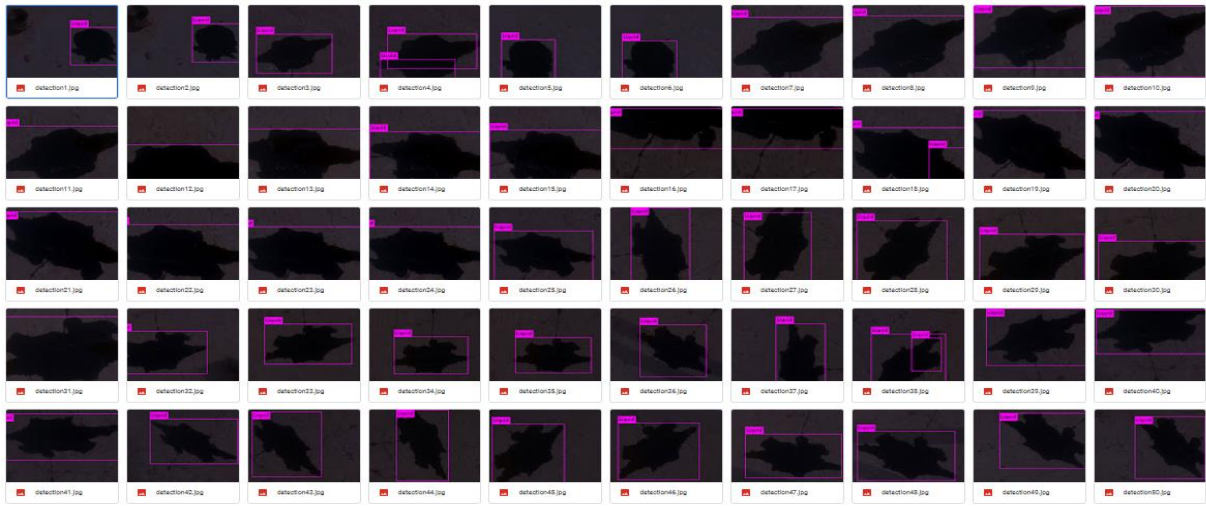


Figure 3-11 oil dataset testing

- **For Non-Oil dataset testing:**

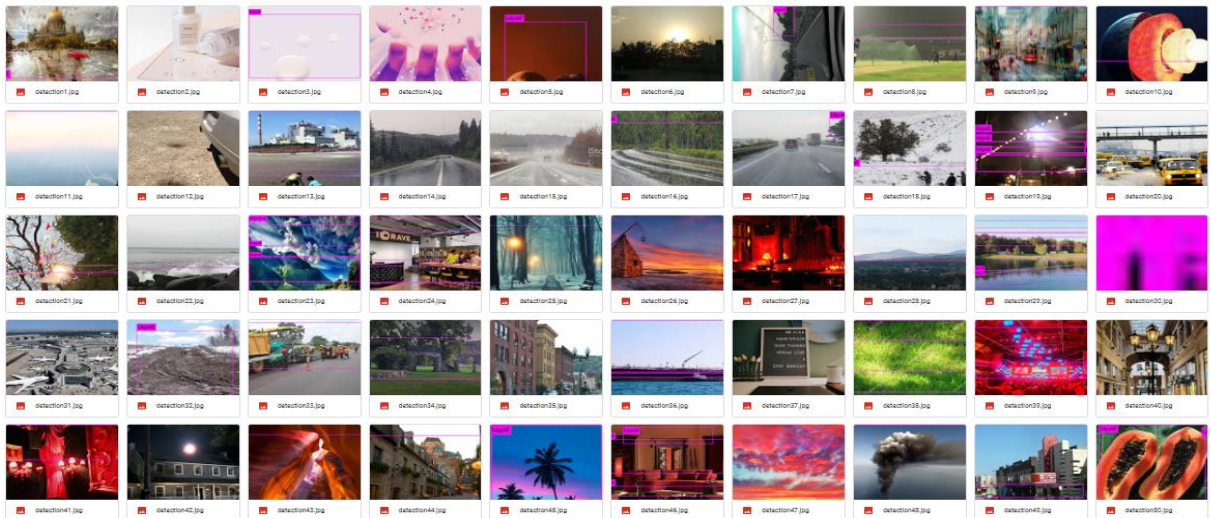


Figure 3-12 non-oil dataset testing

VI. Constructing the confusion matrix with parameters calculated from the previous step as this:

CM		Predicted	
		1	0
Actual	1	88	0
	0	36	22

**From the previous matrix calculating the following:**

- Accuracy =  $(TP+TN)/(TP+FP+FN+TN)$   
 $= (88+22) / (88+0+36+22) = 75.34\%$
- Precision =  $TP/(TP+FP)$   
 $= 88 / (88+36) = 70.97\%$
- Recall =  $TP/(TP+FN)$   
 $= 88 / (88+0) = 100\%$
- F1Score =  $2*((Precision*Recall) / (Precision + Recall))$   
 $= 2*((70.97*100) / (70.97+100))$   
 $= 83.02\%$

For this result with accuracy of 75.34% need to be enhanced so we can make an augmentation to increase the dataset so we will construct a new dataset with the augmented dataset from 146 testing image to 400 testing images.

### Working with new dataset

- Our testing dataset consists of 1000 images that acted as 20% of the dataset.
- Then, we divided the testing dataset for:
  - 60% for Oil data as 600 images:
    - True positives are 579 images.
    - False negatives are 21 images.
  - 40% for Normal data (Non-Oil) as 400 images:
    - True negatives are 228 images.
    - False positives are 172 images.
- Constructing the confusion matrix with parameters calculated from the previous step as this:

CM		Predicted	
		1	0
Actual	1	579	21
	0	172	228

#### **From the previous matrix calculating the following:**

- Accuracy =  $(TP+TN)/(TP+FP+FN+TN)$   
 $= (579+228)/(579+21+172+228) = 80.7\%$
- Precision =  $TP/(TP+FP)$   
 $= 579 / (579+172) = 77.1\%$

- Recall =  $TP / (TP + FN)$   
 $= 579 / (579 + 21) = 96.5\%$
- F1Score =  $2 * ((Precision * Recall) / (Precision + Recall))$   
 $= 2 * ((77.1 * 96.5) / (77.1 + 96.5))$   
 $= 85.72\%$

Finally, we obtained enough good result accuracy according to the minimized dataset with **80.7%% accuracy**.

### 3.4 Federated Learning algorithm implementation

Federated Learning could be defined as a distributed machine-learning framework that allows a collective model to be constructed from models of data that is distributed across data owners or centers (clients).as shown in figure 4

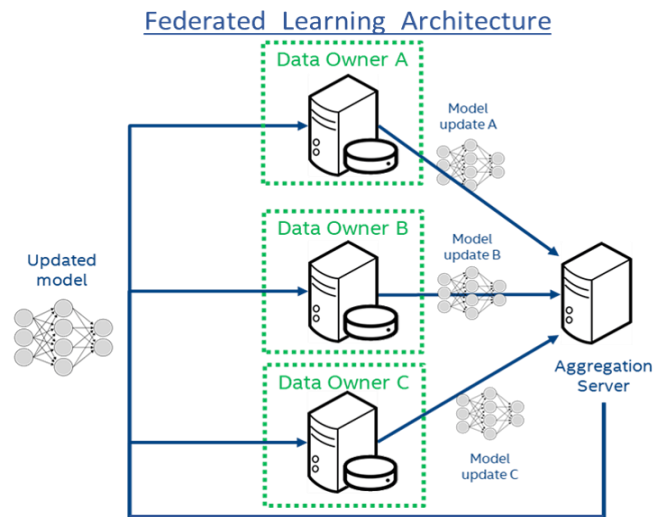


Figure 3-13 Federated architecture

#### 3.4.1 EMNIST DATASET

EMNIST is a large database of handwritten digits that is commonly used for training various image-processing systems. An extended dataset similar to MNIST has been published in 2017, which contains 240,000 training images and 40,000 testing images of digit from 0 to 9 so it has 10 classes to classify.

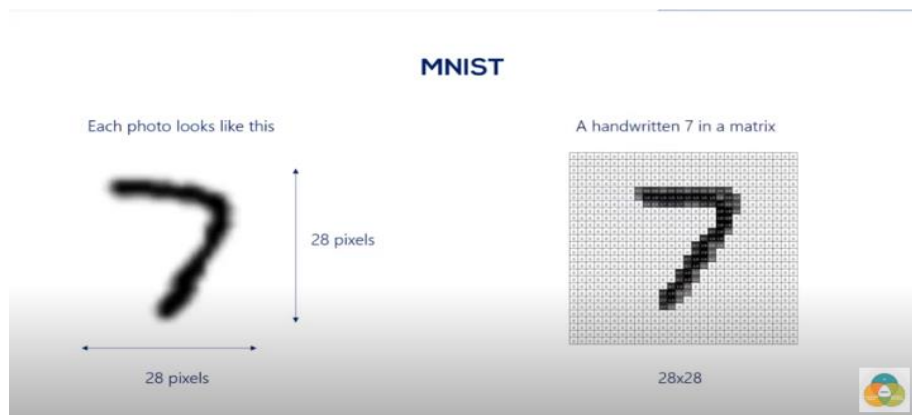


Figure 3-14 MINIST dataset

### 3.4.2 Tensorflow Federated Model

- **First step** : we initialize some values used as instructions for model creation like num\_epoch=50 means that model will make 50 rounds of federated process which take each client update model and integrate them into global model .Then distribute global model to clients(num\_clients =3).the same with other parameters that is necessary for completing the process.

```
In [32]: NUM_EPOCHS = 50
        BATCH_SIZE = 20
        SHUFFLE_BUFFER = 500
        NUM_CLIENTS = 3
```

- **Second step** : Data preparation process

1-loading EMNIST dataset which suitable for federated and distribution process with number of clients

```
In [33]: emnist_train, emnist_test = tf.fsimulation.datasets.emnist.load_data()
```

2-preprocess the data distributed to each client to suit federated model and process

```
In [34]: def preprocess(dataset):
        def element_fn(element):
            return collections.OrderedDict([
                ('x', tf.reshape(element['pixels'], [-1])),
                ('y', tf.reshape(element['label'], [1])),
            ])

        return dataset.repeat(NUM_EPOCHS).map(element_fn).shuffle(
            SHUFFLE_BUFFER).batch(BATCH_SIZE)
```

```
In [35]: def make_federated_data(client_data, client_ids):
        return [preprocess(client_data.create_tf_dataset_for_client(x))
                for x in client_ids]
```

- **Third step** : Create clients and distribute training and testing set for them and shaping data distributed over clients

```
In [36]: sample_clients = emnist_train.client_ids[0: NUM_CLIENTS]
federated_train_data = make_federated_data(emnist_train, sample_clients)
print(federated_train_data)

sample_clients_test = emnist_test.client_ids[0: NUM_CLIENTS]
federated_test_data = make_federated_data(emnist_test, sample_clients_test)
print(federated_test_data)
```

```
In [37]: # This is only needed to create the "federated" ver of the model
#just use to shape our data to federated learning process needs
sample_batch = iter(federated_train_data[0]).next()
sample_batch = collections.OrderedDict([
    ('x', sample_batch['x'].numpy()),
    ('y', sample_batch['y'].numpy()),
])
```

- **Fourth step**: creating simple model that used to process each client data on server to create global model, which distributed into clients.

In our experiment, we use sequential model that contain an input layer of 784 node of inputs (input image 28x28) ,one hidden layer , one output layer with softmax activation function with 10 classes output.

```
In [44]: # Create a new model
def create_compiled_keras_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Input(shape=(784,)),
        tf.keras.layers.Dense(10, kernel_initializer='zeros'),
        tf.keras.layers.Softmax(),
    ])

    def loss_fn(y_true, y_pred):
        return tf.reduce_mean(tf.keras.losses.sparse_categorical_crossentropy(y_true, y_pred))

    model.compile(
        loss=loss_fn,
        optimizer=gradient_descent.SGD(learning_rate=0.02),
        metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
    return model
```

```
In [45]: # Turn model into one that can be used with TFF
def model_fn():
    keras_model = create_compiled_keras_model()
    return tff.learning.from_compiled_keras_model(keras_model, sample_batch)
```



- **Fifth step:** Training model with data prepared and distributed to clients

So after creating model and preparing data, It create initialize model and send it to all clients. Begin the first round that make each user create its own model. Then send local model to server to combine all models into one global model and send it to client .then the same with each round of training.

```
In [46]: # Initialize training
iterative_process = tff.learning.build_federated_averaging_process(model_fn)
state = iterative_process.initialize()

trained_clients=[]
def get_train_data(keep_it_stupid_simple=False):
    if keep_it_stupid_simple:
        if not trained_clients:
            trained_clients.append(sample_clients)
        return federated_train_data
    sc = choices(emnist_train.client_ids, k=NUM_CLIENTS)
    for c in sc:
        while True:
            if c in trained_clients:
                sc.remove(c)
                newc=choices(emnist_train.client_ids, k=1)[0]
                if newc not in trained_clients:
                    sc.append(newc)
                    break
            else:
                break
    trained_clients.append(sc)
    new_federated_train_data = make_federated_data(emnist_train, sc)
    return new_federated_train_data
```

```
In [47]: # Training process
for round_num in range(1, NUM_EPOCHS+1):
    federated_train_data=get_train_data(True)
    state, metrics = iterative_process.next(state, federated_train_data)
    print('round {:2d}, metrics={}'.format(round_num, metrics))

    print('Trained {:2d} clients'.format(len(trained_clients)*NUM_CLIENTS))
    print(trained_clients)
```

- After compile training process, each round show results of accuracy and lose function values. With Associated number of clients.

```
round 40, metrics=<sparse_categorical_accuracy=0.99978185,loss=0.08393426>
round 41, metrics=<sparse_categorical_accuracy=0.9999273,loss=0.08158986>
round 42, metrics=<sparse_categorical_accuracy=1.0,loss=0.07980117>
round 43, metrics=<sparse_categorical_accuracy=0.9999273,loss=0.07792594>
round 44, metrics=<sparse_categorical_accuracy=0.9999273,loss=0.07580581>
round 45, metrics=<sparse_categorical_accuracy=0.9999273,loss=0.07436204>
round 46, metrics=<sparse_categorical_accuracy=1.0,loss=0.07275914>
round 47, metrics=<sparse_categorical_accuracy=1.0,loss=0.07131687>
round 48, metrics=<sparse_categorical_accuracy=1.0,loss=0.069494836>
round 49, metrics=<sparse_categorical_accuracy=1.0,loss=0.0678857>
round 50, metrics=<sparse_categorical_accuracy=1.0,loss=0.06687253>
Trained 3 clients
[['f0000_14', 'f0001_41', 'f0005_26']]
```

### 3.4.3 Evaluation of Model

After training of model with determined number of rounds/epochs, we need to evaluate our model. So we test model with training and testing set used in experiment to prove our point of power of federated learning to increase accuracy of model with preserving data of users.

```
In [48]: # Evaluation
evaluation = tff.learning.build_federated_evaluation(model_fn)

train_metrics = evaluation(state.model, federated_train_data)
print('Train metrics', str(train_metrics))

test_metrics = evaluation(state.model, federated_test_data)
print('Test metrics', str(test_metrics))

Train metrics <sparse_categorical_accuracy=1.0,loss=0.09275841>
Test metrics <sparse_categorical_accuracy=0.93939394,loss=0.43198553>
```

As shown it reach 100% accuracy with training set used and 94% with testing set.

## 3.5 SYSTEM FEATURES

There are technical and non-technical features for our system:

### 3.5.1 Technical Features

The system is built using Federated learning and computer vision.

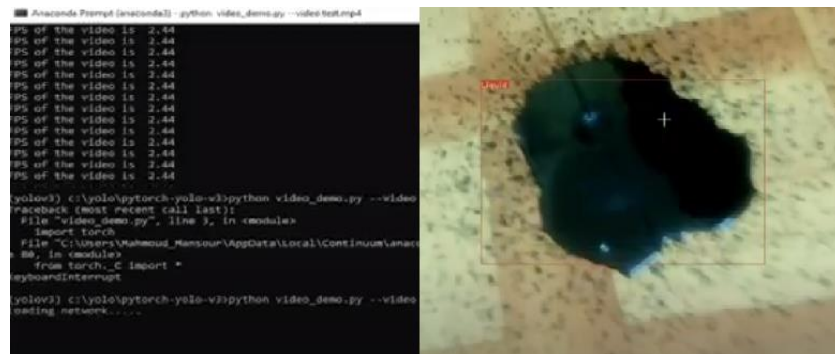
1. Federated learning is used to enable collaborative machine learning by centralizing the training model with all local datasets on one server and enables multiple distributed actors (the factories and every organization has the system) to use and modify that model.
  - we use EMNIST dataset. The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset.

Computer Vision: fire and oil leakage detection depends on computer-vision algorithms, we've used YOLO version 3 with fast R-CNN.

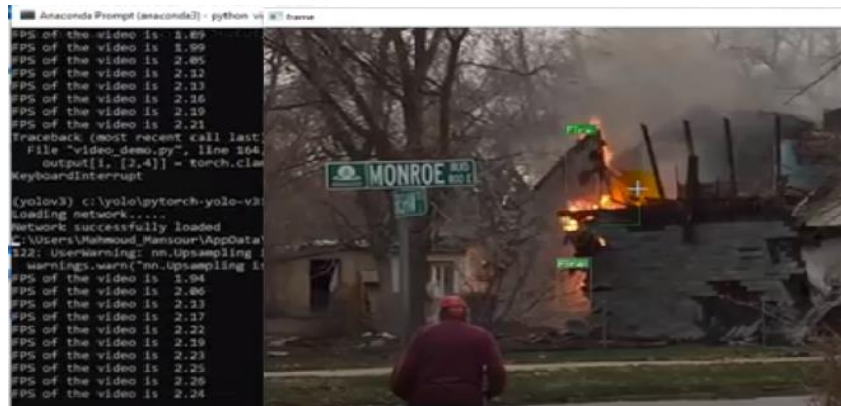
- YOLO-v3 is one of most real-time object detection algorithms in computer vision because it's a single stage architecture that goes straight from image pixels to bounding box coordinates and class probabilities so it's used for better, stronger and faster real time detection than all other versions and that makes the detection in our cases is truly efficient and fast.
- Fast R-CNN is one of the famous object detection architectures that uses YOLO to efficiently classify object proposals. Its efficiency depends on dividing the image into various regions then considering each region as a separate image and pass all regions to CNN to classify them into various classes and when it has divided each region into its corresponding class it combines all regions to get

original image with detected objects in a very fast way so it's the most convenient in our cases.

- So, that indicates the result:
  - for oil leakage detection:



-and fire detection



### 3.5.2 Non-Technical Features

We developed our project to view its contents anywhere can contain these disasters but mainly in factories. We chose factories among other places because of many key facts proves that factories are the most place in which these disasters can occur, so factories are very suitable architectural structure to display our content on it.

Our system increases safety and decreases the resulting cost of loss of employees and materials when such a disaster happens. When the employees feel safer, they can give all their intention to their work comfortably and definitely that can gain production with saving so many costs and also it provides good marketing for the organization, so this can be considered as a good revenue for business stakeholders.

### 3.5.3 Used Tools

We've used Google Colab, Darknet and TensorFlow.

- 1) Google Colab is a cloud service and it supports free GPU, we used it in training because of the incredible GPU capabilities it offers and it's free.
- 2) Darknet is a component of the greater "deep web," a network of encrypted Internet content that is not accessible via traditional search engines and it's an [overlay network](#) within the [Internet](#) that can only be accessed with specific software, configurations, or authorization.
- 3) TensorFlow is a Python-friendly open source library for numerical computation that makes machine learning faster and easier. We can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks and many other fields. TensorFlow supports production prediction at scale, with the same models used for training. TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs.

## 4 CHAPTER FOUR: CONCLUSION

Disasters happen every day of our lives, and this disaster affect our economic and ecological lives. It is possible to detect this disaster at its early stage during surveillance, as well as saving a large number of human lives which can greatly facilitate disaster management. So we think that our project has maximum importance to be available for facilities that face these disasters. Project reach very good accuracy with cases selected and with the power of federated learning approach we can get more benefits (distributed ML model, preserving privacy, Strengthening the model from all user).

### **Highlight achievements:**

- We modify YOLOv3 algorithm and create our own dataset to reach a very Good prototype accuracy for each case.
- Real time detection for each case using hardware setup and software tools
- Apply and Evaluate Federated Learning with MNIST dataset with very good accuracy and approve power of federated learning.

### **4.1 FUTURE WORK**

We are very existing to add many other extensions to our project in the future

- Improve the accuracy for each case which will reduce fault tolerance and fake alarm or fake detection.
- Apply Federated Learning to Yolo Algorithm which will be the first time this algorithm used which will enhance the accuracy, increase the performance, and save the privacy.
- Increase the number of cases like (gas leakage, object falling, electric disruption and other dangerous cases).
- Generalize our solution to use it in many places not just industrial one giving it the ability to be applied in hospitals, schools, universities, or any organizations whatever it's size.
- Provide the solution as web service to reach more users and make it easier to use.

## 4.2 References to Electronic Sources

- [1] Dataset Preparation  
<https://www.altexsoft.com/>
- [2] Fire History  
<https://ifpmag.mdmpublishing.com/>
- [3] Computer Vision history  
<https://www.forbes.com/>
- [4] TensorFlow Federated  
[https://www.tensorflow.org/federated/tutorials/federated\\_learning\\_for\\_image\\_classification](https://www.tensorflow.org/federated/tutorials/federated_learning_for_image_classification)
- [5] Darknet  
<https://pjreddie.com/darknet/>
- [6] YOLOv3  
<https://github.com/augmentedstartups/yolov3workflow>
- [7] PyTorch  
<https://pytorch.org/>
- [8] Anaconda  
<https://www.anaconda.com/>
- [9] YOLOv3 on Colab  
[https://colab.research.google.com/drive/1Mh2HP\\_Mfxoao6qNFbhfV3u28tG8jAVGk](https://colab.research.google.com/drive/1Mh2HP_Mfxoao6qNFbhfV3u28tG8jAVGk)
- [10] Fire Dataset  
<https://github.com/sulenn/fire-dataset>
- [11] Fire Dataset  
<https://www.kaggle.com/phylake1337/fire-dataset>
- [12] Oil Dataset
  - Training  
<https://drive.google.com/file/d/1VMMEno3j1sfwQu3jrltNFzDnnUF2OGn0/view?usp=sharing>
  - Testing
    - Oil  
<https://drive.google.com/file/d/1LTkcgPhFAA04pZwMIHxMmmxhWCE-RT8l/view?usp=sharing>
    - Non-oil  
<https://drive.google.com/file/d/1tr8uyfirxVOywbh10RbgqDqFzRXIcE5D/view?usp=sharing>