

**2024 Spring Semester**

**Data Mining Class**

**Term Project Report**

**Classification on Side-scan Sonar Imaging**

**Dataset for Mine Detection with Deep**

**Learning Technique**

**Emine ŞENER**

**21360859058**

**21.05.2024**

# Table of Contents

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Investigation of Side-scan sonar imaging for Mine Detection Dataset.....</b>	<b>3</b>
<b>3. Transfer Learning Technique.....</b>	<b>5</b>
<b>4. YoloV4 Architecture.....</b>	<b>6</b>
<b>5. YoloV4 Algorithm.....</b>	<b>7</b>
<b>6. Examination of the Results Obtained.....</b>	<b>14</b>
<b>7. Code and Representation Links.....</b>	<b>17</b>
<b>8. References.....</b>	<b>17</b>

## 1.Introduction

This report details the process of classifying the Side-scan sonar imaging dataset for mine detection using deep learning techniques. Given the large-scale nature of the project, each phase has been carefully outlined. The project explores various methods, including the investigation of the dataset, the application of transfer learning, and the implementation of the YoloV4 architecture and algorithm. Results are analyzed and compared with existing literature, and links to the code and presentation are provided.

## 2.Investigation of Side-scan Sonar Imaging for Mine

# Detection Dataset

## 2.1 Examining the Dataset

The dataset used in this project consists of images and labels collected over several years using a specialized hardware device. It includes data from 2010, 2015, 2017, 2018, and 2021, capturing a diverse range of samples across different time periods.

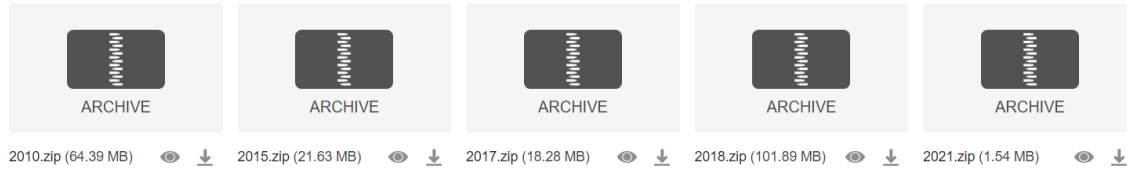


Image 2.a:Parts of Dataset

The data is organized into five distinct files, each named according to the year of collection. For example, image 2.a illustrates images and labels gathered in 2015.

0001_2015.jpg	19.05.2023 13:34	JPG Dosyası	323 KB
0001_2015.txt	19.05.2023 13:34	Metin Belgesi	1 KB
0002_2015.jpg	19.05.2023 13:34	JPG Dosyası	36 KB
0002_2015.txt	19.05.2023 13:34	Metin Belgesi	1 KB
0003_2015.jpg	19.05.2023 13:34	JPG Dosyası	304 KB
0003_2015.txt	19.05.2023 13:34	Metin Belgesi	1 KB
0004_2015.jpg	19.05.2023 13:34	JPG Dosyası	317 KB
0004_2015.txt	19.05.2023 13:34	Metin Belgesi	1 KB
0005_2015.jpg	19.05.2023 13:34	JPG Dosyası	36 KB
0005_2015.txt	19.05.2023 13:34	Metin Belgesi	1 KB



Image 2.b:Images and Labels in 2015 Folder

The dataset used in this project is formatted to be compatible with the YOLO Algorithm, which will be discussed in the fifth stage of the report. The datasets for the YOLO Algorithm are organized sequentially, with images and labels stored in .jpg and .txt files, respectively.

The images and their corresponding labels share the same name but are differentiated by their file extensions. Files with the .jpg extension represent the images to be classified, while the .txt files contain manually specified properties related to these images.

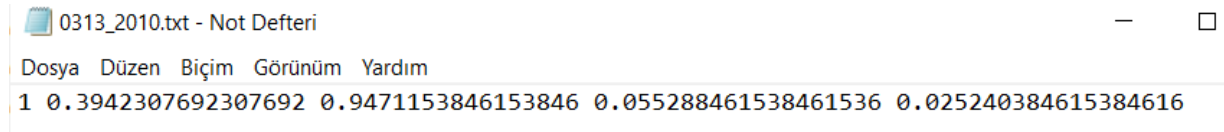
The datasets consist of paired .txt and .jpg files. Each .jpg file contains an image, and its corresponding .txt file provides information about the coordinates and dimensions of any detected mines. There are three different states for each

mine in the set, which is crucial when transitioning between different models. If no mines are detected in a given image, the corresponding .txt file will be empty. For example, the images in Figure 2.c illustrate this structure.

 0007_2010.jpg	19.05.2023 13:33	JPG Dosyası	312 KB
 0007_2010.txt	19.05.2023 13:33	Metin Belgesi	0 KB

*Image 2.c: Images and Labels in 2015 Folder*

If there is only one mine in the image, the corresponding .txt file contains only one line. This line includes the coordinates and dimensions of the detected mine.



*Image 2.d: Label of image data*

Image 2.d illustrates the label information for an image containing a single mine. Each mine is represented by a line, which consists of five pieces of data separated by spaces. The first value is either 0 or 1: 0 indicates that the drawn box contains a mine, while 1 indicates it does not. The next two values provide the x and y coordinates, respectively, and the final two values specify the height and width of the bounding box.

If the image contains multiple mines, the .txt file will have a corresponding number of lines, each representing a different mine. This data is essential for the process of drawing bounding boxes around the mines. Since the dataset includes both images and labels, the data classification process can proceed without additional labeling.

## **2.2 Evaluation of the Dataset**

Since time information does not contribute to the feature extraction process for classification, data from five different files collected over various years in the downloaded dataset are merged into a single file. This consolidated file is then split into 'obj' and 'valid' subsets to be used with the YOLO algorithm, and it is compressed for easy uploading to Google Drive. During the data split into 'obj' and 'valid', an 80% training and 20% test ratio is applied.

Due to its size, the dataset performs inefficiently in local environments, necessitating the use of the Google Colab platform. Code execution on Google Colab involves accessing data stored on Drive and saving outputs back to Drive.

## **3. Transfer Learning**

In this project, deep learning techniques were employed instead of creating the base network layers from scratch. Transfer learning involves reusing a model trained on a larger dataset for a smaller dataset.

There are two primary approaches to transfer learning: freezing weights and starting with learned values. Freezing weights in transfer learning typically involves using layers (feature extractors) closer to the input, as these layers learn lower-level and more generalizable features. When the initial training dataset differs significantly from the current dataset, the pre-trained model is utilized mainly for feature extraction, and additional layers are added specifically for the current dataset.

On the other hand, starting with learned values is particularly effective when the initial training dataset and the current dataset are similar, resulting in enhanced performance.

In this project, the YOLOv4 algorithm, originally trained on the COCO dataset, is utilized with the transfer learning technique of starting with learned values.

## **4. YoloV4 Architecture**

YOLOv4, a state-of-the-art object detection algorithm, excels in single-stage object tracking and is specifically optimized for efficient execution on a single GPU. Its design not only enables rapid and accurate detection of objects but also ensures optimal performance, making it suitable for real-time applications.

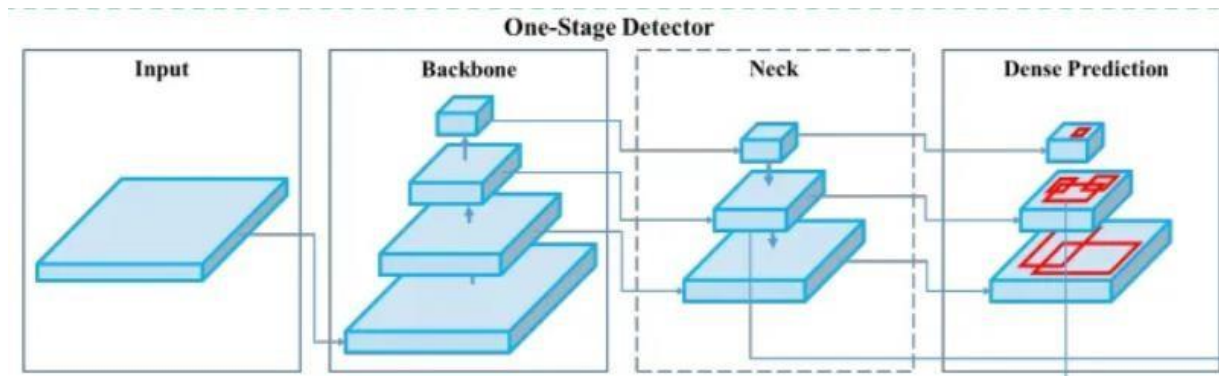


Image 4.a: YoloV4 Architecture

YOLOv4 is a CNN-based algorithm, renowned as one of the most successful techniques in Deep Learning for Object Detection. It shortens long training processes and enhances performance.

The input layer, where the image is fed into the model, is followed by the Backbone layer, which is responsible for feature extraction.

Between the Backbone and the Head lies the Neck layer, added to gather more information for object estimation. This layer derives detailed information from neighboring feature maps using both bottom-up and top-down flows.

The final layer, the Head layer, predicts the bounding boxes and the class of each detected object.

## 5. YoloV4 Algorithm

### 5.1 Configuring YoloV4 Basic

Several important configurations are necessary for the execution of the YOLOv4 algorithm. The `yolov4.cfg` file contains the essential configurations for YOLOv4. Within this file, the fundamental architecture and training parameters of the model are defined, including the added layers, each convolutional layer, and the number of neurons per layer. Additionally, it specifies parameters such as Intersection over Union (IoU), which is used to evaluate the model's performance. Another critical aspect of the `yolov4.cfg` file is the specification of the number of target classes for the classification process. The number of classes must be accurately defined in this file for proper model training and performance.

The number of target classes in the classification process should be specified in the `yolov4.cfg` file.

Another important configuration file is `obj.names`. This file must contain as many different names as the number of classes specified in the `yolov4.cfg` file. For example, in our code, the names 'MILCO' and 'NOMBO' are used. 'MILCO' indicates that the detected object is a projectile, while 'NOMBO' signifies that it is not a bullet.

Additionally, the `train.txt` and `test.txt` files are crucial configuration files that store the locations of the training and test data. These files were created in the project using the following code snippets.

The code that creates the `train.txt` file is shown in Image 5.a.

```
import os
images = []
os.chdir(os.path.join("data", "obj"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        images.append("data/obj/" + filename)
os.chdir("..")

with open("train.txt", "w") as outfile:
    for img in images:
        outfile.write(img)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

Image 5.a: Part 1 of Code

The code that creates the `test.txt` file is shown in Image 5.b.

```
import os
images = []
os.chdir(os.path.join("data", "valid"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        images.append("data/valid/" + filename)
os.chdir("../")

with open("test.txt", "w") as outfile:
    for img in images:
        outfile.write(img)
        outfile.write("\n")
    outfile.close()
os.chdir("../")
```

Image 5.b: Part 2 of Code

The final configuration step is to save all these configuration files in the `obj.data` file. The contents of `obj.data` are shown in Image 5.c.

```
1 classes = 2
2 train = data/train.txt
3 valid = data/test.txt
4 names = data/obj.names
5 backup = /mydrive/yolov4/training
```

Image 5.c: Part 3 of Code

## 5.2 Importing the YoloV4 Dataset Into The Code Environment

Since the dataset is stored in Google Drive, it is first connected to the drive as shown in Figure 3.d.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Image 5.d: Part 4 of Code

The training and test data uploaded to Google Drive are then imported into the code environment.

```
!cp /content/gdrive/MyDrive/obj.zip /content/
!cp /content/gdrive/MyDrive/valid.zip /content/
```

Image 5.e: Part 5 of Code

The files in zip format are then extracted.

```
!unzip obj.zip -d ./data
!unzip valid.zip -d ./data
```

Image 5.f: Part 6 of Code

Since all data in the Colab environment will be stored in the content directory, the



directory is navigated to. The codes for generating train.txt and test.txt are then executed.

During the training process, the weights of the YOLOv4 model trained on the COCO dataset are utilized for optimal performance. These weights, deemed suitable for the project, are downloaded from the Darknet page and saved to Google Drive.

```
!wget
https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/
yolov4.conv.137
```

*Image 5.g: Part 7 of Code*

### 5.3 YoloV4 Training Process

The drive is reconnected for the training process, and the Darknet framework is downloaded to the Content directory.

```
from google.colab import drive
drive.mount('/content/gdrive')
cd /content/
!git clone https://github.com/AlexeyAB/darknet
```

*Image 5.h: Part 8 of Code*

When loading Darknet, certain configuration changes are necessary.

```
cd /content/darknet
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
!apt update
!apt-get install libopencv-dev
```

*Image 5.i: Part 9 of Code*

OpenCV, GPU, and CuDNN are activated, and Darknet is compiled using the make command. The training data must then be transferred to Darknet.

```
!cp /content/gdrive/MyDrive/obj.zip /content/darknet/data/
cd /content/darknet/data/
!unzip obj.zip
```

*Image 5.j: Part 10 of Code*

The same applies to the test data.

```
!cp /content/gdrive/MyDrive/valid.zip /content/darknet/data
cd /content/darknet/data/
!unzip valid.zip
```

*Image 5.k: Part 11 of Code*

All configuration files and pre-trained weights, as defined in the introduction of the report and uploaded to Google Drive, are transferred to Darknet.

```
!cp /content/gdrive/MyDrive/yolov4-custom.cfg /content/darknet/cfg
!cp /content/gdrive/MyDrive/obj.names /content/darknet/data
!cp /content/gdrive/MyDrive/obj.data /content/darknet/data
!cp /content/gdrive/MyDrive/train.txt /content/darknet/data
!cp /content/gdrive/MyDrive/yolov4.conv.137 /content/darknet/data
```

Image 5.l: Part 12 of Code

Darknet starts training with the training data.

```
!./darknet detector train /content/darknet/data/obj.data
/content/gdrive/MyDrive/yolov4-custom.cfg
/content/darknet/data/yolov4.conv.137 -dont_show -map
```

Image 5.m: Part 13 of Code

Due to the complexity of the training dataset and the large-scale nature of the model, the training process takes considerably longer compared to other object detection models. Figure 3.d illustrates a small segment of the training process.

```
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.182845, iou_loss = 0.000000, total_loss = 0.182845
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.180197, iou_loss = 0.000000, total_loss = 0.180197
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.289487, iou_loss = 0.000000, total_loss = 0.289487
total_bbox = 15867, rewritten_bbox = 0.252096 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.209945, iou_loss = 0.000000, total_loss = 0.209945
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.170482, iou_loss = 0.000000, total_loss = 0.170482
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.321357, iou_loss = 0.000000, total_loss = 0.321357
total_bbox = 15867, rewritten_bbox = 0.252096 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.223864, iou_loss = 0.000000, total_loss = 0.223864
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.177669, iou_loss = 0.000000, total_loss = 0.177669
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.309622, iou_loss = 0.000000, total_loss = 0.309622
total_bbox = 15867, rewritten_bbox = 0.252096 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.384805), count: 5, class_loss = 9.201688, iou_loss = 11.240891, total_loss = 20.442579
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.347671), count: 6, class_loss = 9.536484, iou_loss = 5.007107, total_loss = 14.543591
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.309144, iou_loss = 0.000000, total_loss = 0.309144
total_bbox = 15878, rewritten_bbox = 0.251921 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.198061, iou_loss = 0.000000, total_loss = 0.198061
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.171763, iou_loss = 0.000000, total_loss = 0.171763
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.354450, iou_loss = 0.000000, total_loss = 0.354450
total_bbox = 15878, rewritten_bbox = 0.251921 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.160940, iou_loss = 0.000000, total_loss = 0.160940
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.171432, iou_loss = 0.000000, total_loss = 0.171432
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.273020, iou_loss = 0.000000, total_loss = 0.273020
total_bbox = 15878, rewritten_bbox = 0.251921 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.240688, iou_loss = 0.000000, total_loss = 0.240688
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.157443, iou_loss = 0.000000, total_loss = 0.157443
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.297866, iou_loss = 0.000000, total_loss = 0.297866
total_bbox = 15878, rewritten_bbox = 0.251921 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class_loss = 0.187443, iou_loss = 0.000000, total_loss = 0.187443
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.000000), count: 1, class_loss = 0.171389, iou_loss = 0.000000, total_loss = 0.171389
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.299655, iou_loss = 0.000000, total_loss = 0.299655
```

Image 5.n: Part 14 of Code

The YOLOv4 algorithm is evaluated using the IOU metric and class prediction accuracy. To achieve this, two distinct loss values are computed: `iou_loss` and `class_loss`, which together contribute to the `total_loss`. Given that the dataset's `.txt` files contain both class (0/1) and bounding box (location and size) information, separate loss calculations are performed for these distinct properties.

## 5.4 YoloV4 Training Process Evaluation Metrics

As depicted in Figure 3.e, alongside the commonly used `class_loss`, there is also the IOU loss. IOU (Intersection over Union) is a widely adopted technique for evaluating algorithms that employ bounding box methods.

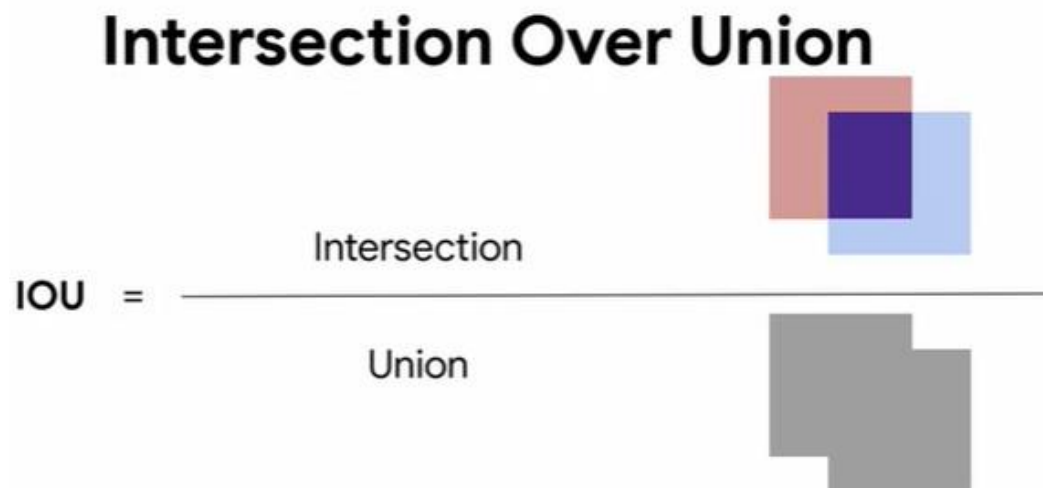


Image 5.o: Part 15 of Code

Algorithms that utilize bounding boxes involve two distinct boxes: the predicted box and the actual box. When calculating the IOU, the intersection of these two boxes is placed in the numerator, while the union is placed in the denominator. This method yields the IOU score.

The YOLOv4 training process is visualized in `echart.png`. To display this graph, a helper function like the one below is required:

```
def imShow(path):
    import cv2
    import matplotlib.pyplot as plt
    %matplotlib inline

    image = cv2.imread(path)
    height, width =
    image.shape[:2]
    resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.
INTER_CUBIC)

    fig = plt.gcf()
    fig.set_size_inches(18, 10)
    plt.axis("off")
```

Image 5.p: Part 16 of Code

## 5.5 YoloV4 Verification Process

The model is evaluated using the following command.

```
!./darknet detector map /content/gdrive/MyDrive/obj.data
/content/gdrive/MyDrive/yolov4-custom.cfg /content/gdrive/MyDrive/yolov4-
custom_last.weights
```

Image 5.q: Part 17 of Code

Validation output includes complexity matrix values and derived variables from

these values. A small excerpt from the validation output is provided below.

```
26 conv    512      3 x 3/ 1    13 x 13 x 512 -> 13 x 13 x 512 0.797 BF
27 conv    256      1 x 1/ 1    13 x 13 x 512 -> 13 x 13 x 256 0.044 BF
28 conv    512      3 x 3/ 1    13 x 13 x 256 -> 13 x 13 x 512 0.399 BF
29 conv     21      1 x 1/ 1    13 x 13 x 512 -> 13 x 13 x 21 0.004 BF
30 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
31 route   27              -> 13 x 13 x 256
32 conv    128      1 x 1/ 1    13 x 13 x 256 -> 13 x 13 x 128 0.011 BF
33 upsample          2x      13 x 13 x 128 -> 26 x 26 x 128
34 route   33 23              -> 26 x 26 x 384
35 conv    256      3 x 3/ 1    26 x 26 x 384 -> 26 x 26 x 256 1.196 BF
36 conv     21      1 x 1/ 1    26 x 26 x 256 -> 26 x 26 x 21 0.007 BF
37 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 6.789
avg_outputs = 299797
Allocate additional workspace_size = 26.22 MB
Loading weights from /content/gdrive/MyDrive/yolov4-custom_last.weights...
seen 64, trained: 57 K-images (0 Kilo-batches_64)
Done! Loaded 38 layers from weights-file

calculation mAP (mean average precision)...
Detection layer: 30 - type = 28
Detection layer: 37 - type = 28
1172
detections count = 3197, unique truth count = 668
```

*Image 5.r: Part 18 of Code*

## 5.6 YoloV4 Testing Process

A new image is uploaded to Google Drive for the YOLOv4 testing process. The trained model draws bounding boxes on the image and assigns probability values to them. An auxiliary function displays both the bounding boxes and associated probability values on the screen.

```
def imshow(path):
    import cv2
    import matplotlib.pyplot as plt
    %matplotlib inline

    image = cv2.imread(path)
    height, width = image.shape[:2]
```

*Image 5.s: Part 19 of Code*

```

    resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.
INTER_CUBIC)

    fig = plt.gcf()
    fig.set_size_inches(18, 10)
    plt.axis("off")
    plt.imshow(cv2.cvtColor(resized_image, cv2. COLOR_BGR2RGB))

```

Image 5.t: Part 20 of Code

The following command is employed to test the model, and the test results are displayed using a helper function.

```

!./darknet detector test /content/gdrive/MyDrive/obj.data
/content/gdrive/MyDrive/yolov4-custom.cfg /content/gdrive/MyDrive/yolov4-
custom_5000.weights /content/gdrive/MyDrive/Image01.png -thresh 0.5

imshow('predictions.jpg')

```

Image 5.u: Part 20 of Code

## 6. Examination of the Results Obtained

### 6.1 The results of the developed model

The graph illustrating the metrics obtained during the training is shown below.

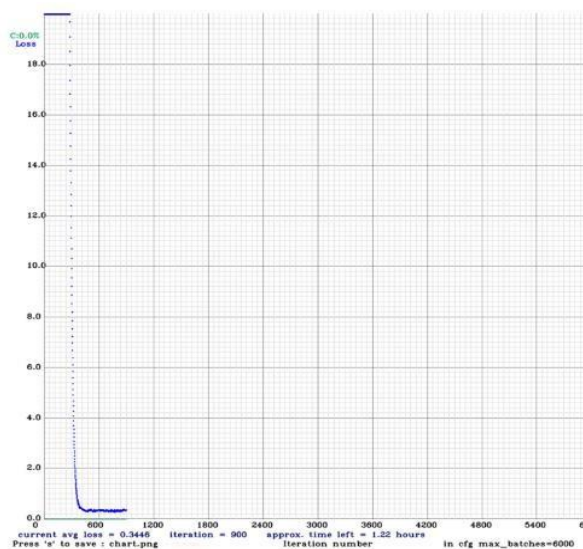


Image 6.a: graph of results

A new image has been uploaded to Google Drive for testing, and it will draw a bounding box if necessary.

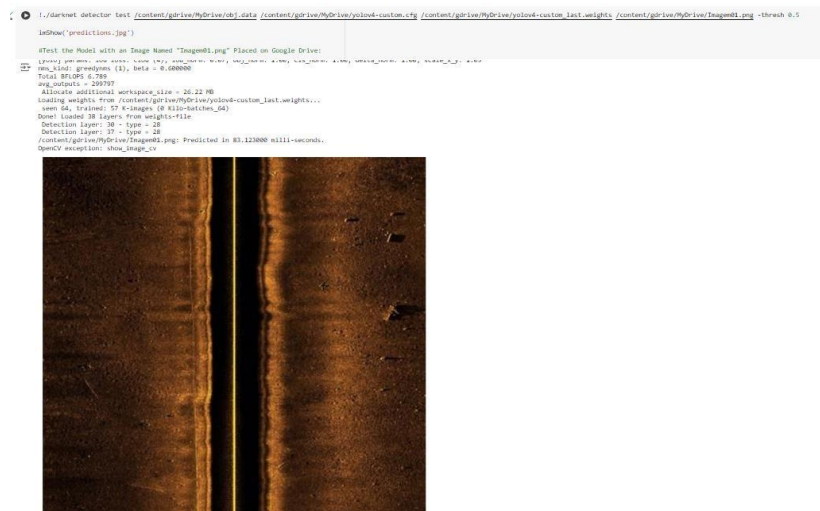


Image 6.b: predict of model

During the testing phase, upon finding that the bounding boxes were not drawn as expected, the output of the validation code was reviewed. It was concluded that the deep learning network, established using the information in the .cfg file, remained insufficiently complex for such a demanding task.

```

calculation mAP (mean average precision)...
Detection layer: 30 - type = 28
Detection layer: 37 - type = 28
1172
detections_count = 3197, unique_truth_count = 668
class_id = 0, name = MILCO, ap = 0.03%          (TP = 0, FP = 0)
class_id = 1, name = NOMBO, ap = 0.00%          (TP = 0, FP = 0)

for conf_thresh = 0.25, precision = -nan, recall = 0.00, F1-score = -nan
for conf_thresh = 0.25, TP = 0, FP = 0, FN = 668, average IoU = 0.00 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.000149, or 0.01 %
Total Detection Time: 17 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

```

Image 6.c: metrics of model

In the evaluation metrics from the training code output, it is evident that the loss value is notably high.

```

(next mAP calculation at 1000 iterations) H1000/6000: loss=0.2 hours left=1.2
1000: 0.224814, 0.288557 avg loss, 0.002610 rate, 0.798709 seconds, 64000 images, 1.203107 hours left
valid: Using default 'data/train.txt'
valid: Using default 'data/train.txt'

```



## 6.2 Comparison with Article Results

The study compared the performance using side-scan sonar imaging data of underwater vehicles for mine detection with an article. Unlike the article, which presents only numerical values for training results, our approach includes graphical representations.

*In this preliminary test, using our dataset of 1170 images, over the first 5000 training iterations, we achieved an average Intersection over Union (IoU) of 60%, a mean Average Precision (AP) of 75%, a Precision of 82%, and a Recall of 64% with a confidence threshold of 0.25. (Side-scan sonar imaging data of underwater vehicles for mine detection)*

It appears that the IOU and mAP values are significantly higher compared to the project's values.

During the final testing phase described in the article, it is evident that the study successfully detected the bullets.

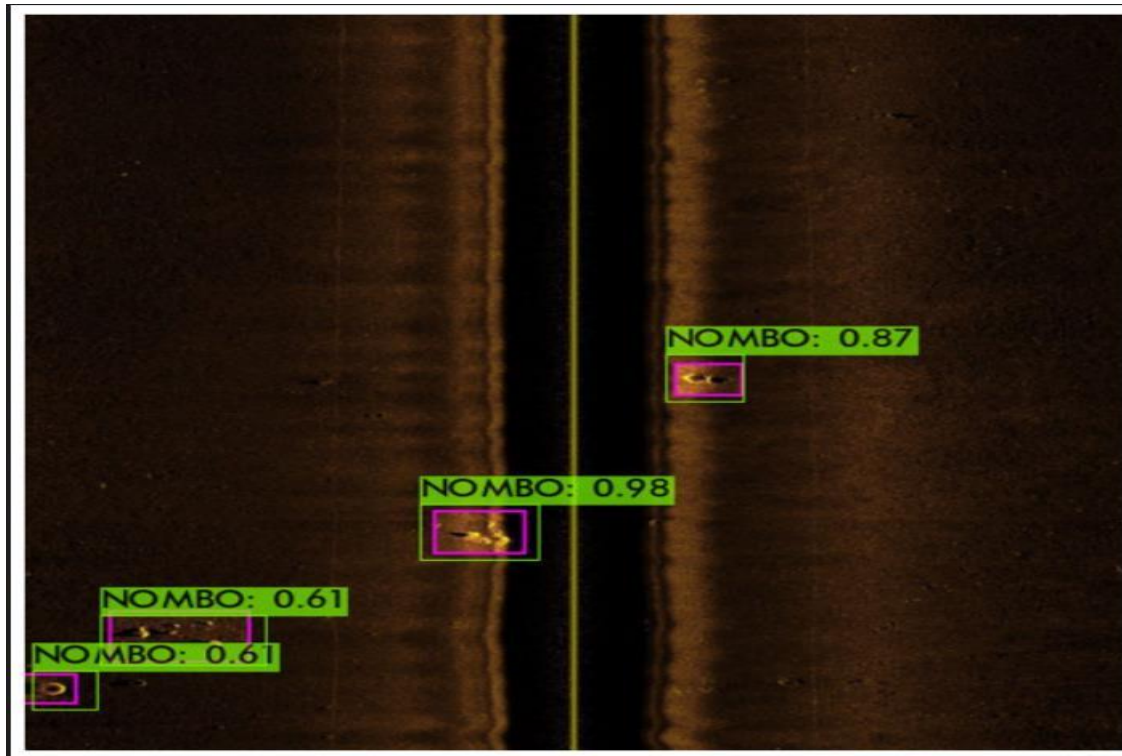


Image 6.e: results of article

When investigating the reason behind the article's excellent results, it was discovered that the same YOLOv4.cfg configuration file used in the article was also utilized in the project work.

Upon examining the .cfg file, it was found to be quite complex. Consequently, hundreds of layers were added to the .cfg file used in the project, similar to the article's approach. However, this resulted in a very slow training process, requiring approximately 15 hours. Platforms like Colab or Kaggle do not support such lengthy training sessions.

## 7. Code and Representation Links

Code Link

[https://github.com/EmineSener/EmineSener-Mine-Detection-with-Deep-Learning-on-Side-Scan-Sonar-Imaging-Data-of-Underwater-Vehicles/blob/main/Real\\_time\\_object\\_classifier.ipynb](https://github.com/EmineSener/EmineSener-Mine-Detection-with-Deep-Learning-on-Side-Scan-Sonar-Imaging-Data-of-Underwater-Vehicles/blob/main/Real_time_object_classifier.ipynb)

Video Link

<https://drive.google.com/file/d/1rTK3PnlWg6dFWxd%20vBLbRhL9qFOxGZOfo/view?usp=sharing>

## 8. References



[1] European Commission, "OCEAN 2020: The EU's largest collaborative defence research project under the PADR successfully completed," 2021. [Online]. Available: [https://defence-industry-space.ec.europa.eu/ocean-2020-eus-largest-collaborative-defence-research-project-under-padr-successfully-completed\\_en](https://defence-industry-space.ec.europa.eu/ocean-2020-eus-largest-collaborative-defence-research-project-under-padr-successfully-completed_en). (Accessed 6 January 2024).

[2] European Union, "Open Cooperation for European maritime awareness," 2021. [Online]. Available: [https://defence-industry-space.ec.europa.eu/document/download/f0316a47-1b9b-4a81-9424-18b6cbcd350\\_en?filename=PADR%202017%20-%20OCEAN2020.pdf](https://defence-industry-space.ec.europa.eu/document/download/f0316a47-1b9b-4a81-9424-18b6cbcd350_en?filename=PADR%202017%20-%20OCEAN2020.pdf). [Accessed 07 January 2023].

[3] N. Pessanha Santos, R. Moura, G. Sampaio Torgal, V. Lobo and M. de Castro Neto, "Side-scan sonar imaging for Mine detection," 2023. [Online]. Available: <https://dx.doi.org/10.6084/m9.figshare.24574879>.





















## 2. Comparison with Article Results

Side-scan sonar imaging data of underwater vehicles for mine detection was used for article comparison. The article does not use graphics for training results, only values are given.

*In this preliminary test, using our dataset of 1170 images, over the first 5000 training iterations, we achieved an average Intersection over Union (IoU) of 60%, a mean Average Precision (AP) of 75%, a Precision of 82%, and a Recall of 64% with a confidence threshold of 0.25. (Side-scan sonar imaging data of underwater vehicles for mine detection)*

It is seen that the IOU and mAP values are quite high compared to the project values.

In the test process at the end of the article, it is seen that the article study detected the bullets quite successfully.

When the reason why the results of the article are so good is investigated, it is seen that the yoloV4.cfg configuration file used in the article is used in the project work.

According to the .cfg file, it was found to be quite complicated.

Thus, hundreds of layers were added to the .cfg file used in the project, as in the article. When the model was trained, the training process was very slow and required an estimated 15 hours. Colab or Kaggle do not support such a long training process.