OIL SPILL DETECTION USING 2DCNN and GAN

A PROJECT REPORT

Submitted by

MUKESH S (312419205064)
RUDRESH S (312419205087)

in partial fulfilment for the requirement of award of the degree

of

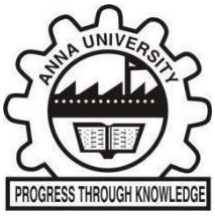
BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

St. JOSEPH'S INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)



ANNA UNIVERSITY, CHENNAI 600 025

APRIL 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"OIL SPILL DETECTION USING 2D CNN AND GAN"** is the bonafide work of **MUKESH S (312419205064)** who carried out the project work under my supervision, for the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Information Technology.

Submitted for the Viva-Voce held on _____.

SIGNATURE

Dr. S.KALARANI M.E.,Ph.D.,

HEAD OFTHEDEPARTMENT

Department of Information Technology

St. Joseph's Institute of Technology

Old Mamallapuram Road

Chennai- 600119

SIGNATURE

Dr. S.NIKKATH BUSHRA

M.E., (Ph.D).,

ASSISTANT PROFESSOR

Department of Information Technology

St. Joseph's Institute of Technology

Old Mamallapuram Road

Chennai- 600119

(EXTERNAL EXAMINER) **(INTERNAL EXAMINER)**

CERTIFICATE OF EVALUATION

College Name **:** St. Joseph's Institute of Technology

Branch & Semester **:** Information Technology (VIII)

S.NO	NAME OF STUDENT	TITLE OF THE PROJECT	NAME OF THE SUPERVISOR WITH DESIGNATION
1	**Mukesh S (312419205064)**	Oil Spill Detection using 2D CNN and GAN	Dr. S.NIKKATH BUSHRA Associate Professor M.E., (Ph.D)

The report of the project work submitted by the above students in partial fulfilment for the award of Bachelor of Technology degree in Information Technology of Anna University were evaluated and confirmed to be reports of the work done by the above students and then evaluated.

(EXTERNAL EXAMINER) **(INTERNAL EXAMINER)**

ACKNOWLEDGEMENT

The contentment and elation that accompany the successful completion of any work would be incomplete without mentioning the people who made it possible.

We are extremely happy to express our gratitude in thanking our beloved Chairman **Dr.B.Babu Manoharan M.A.,M.B.A.,Ph.D.,** who has been a pillar of strength to this college.

Words are inadequate in offering my sincere thanks and gratitude to our respected Managing Director **Mrs.B.Jessie Priya M.Com.,** heartfelt gratitude **to** our respected Executive Director **Mr.B.SashiSekar M.Sc.,** and our beloved Principal **Dr.P.Ravichandran M.Tech.,Ph.D.,** for having encouraged us to do our under graduation in Information Technology in this esteemed college.

We also express my sincere thanks and most heartfelt sense of gratitude to our eminent Head of the Department **Dr.S.Kalarani M.E.,Ph.D.,** for having extended her helping hand at all times.

It is with deep sense of gratitude that we acknowledge our indebtedness to our beloved supervisor **Dr. S. Nikkath Bushra M.E., Ph.D.,** a perfectionist for her expert guidance and connoisseur suggestion.

Last but not the least, we thank our family members and friend who have been the greatest source of support to me.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENT

ABSTRACT

The project deals with the oil spill detection where the satellite or aerial images are taken into a deep learning model which uses 2D CNN (Convolution Neural Network) - one of the powerful models that can be used for volumetric data. In 2D-CNN the kernel moves in 2 directions-mostly used in image data. The 2D CNN detects the oil spill that occurred in the oceans. By detecting the oil spill, we can able to save the marine living creatures and maintain the quality of the water. The deep learning models work in an efficient way when they are trained using large amount of datasets but in our case it was difficult to collect the datasets so we are using Generative Adversarial Network to create new images from existing image. Thus this increases the number of images in testing and training of the model.

CHAPTER 1

INTRODUCTION

1.1 SYSTEM OVERVIEW

Deep learning is a branch of Artificial Intelligence (AI) where it deals with creating a model which can decide on its own. Deep learning model resembles the human brain in order to take decisions. In oil spill detection, deep learning model is used in order to find the oil spills in the ocean. The 2D Convolution Neural Network separates the images as frames and deducts the oil spill, and then Satellite images are taken and these images are used for training and testing of the deep learning model. The satellite images are classified using the 2D CNN model whether the image consists of oil spill and the Generative Adversarial Network is used to generate more images in order to increase the number of images in training and testing dataset which eventually increase the accuracy of the model.

1.2 SCOPE OF THE PROJECT

The aim of the project is to detect the oil spills that occurs in the oceans using the satellite images or aerial images of the oceans. This project is done in order to protect the marine living creatures and quality of water in the ocean. This detection leads to know the spread of oil spill in the ocean. This project uses 2D CNN and GAN would be to develop robust and accurate model for detecting oil spills in various environmental conditions. The model would be able to classify images as either containing oil or not using a 2D CNN and generate realistic images of oil spills in various environments using a GAN. The overall aim of the project would be to create a tool that can be used for environmental monitoring and disaster response, improving the ability to detect and respond to oil spills and other environmental hazards.

CHAPTER 2

LITERATURE SURVEY

2.1 AUTOMATIC TARGET DETECTION FROM SATELLITE IMAGERY USING MACHINE LEARNING

Faster – RCNN, SSD and YOLO. YOLO (you only look once) is state-of-the-art real-time object detection framework based on a CNN (convolutional neural network) algorithm targeted remote sensing and localization by employing region-based techniques and classification algorithms, and showed better results for object localization. Performs poorly when predicting fresh data. However, the results show that this does not happen in techniques so that the models can be adequately trained.

2.2 MARINE OIL SPILL DETECTION FROM SAR IMAGES BASED ON ATTENTION U-NET MODEL USING POLARIMETRIC AND WIND SPEED INFORMATION

The deep-learning-based semantic segmentation algorithms took into account the contextual information of the imagery as well as the polar metric information, which reduced the influence of the speckle noise. The existence of lookalikes exerts more difficulties when detecting oil spills. To better detect oil spills, especially from look-alikes, more information should be utilized. Oil spill detection using SAR (Synthetic Aperture Radar) images is the image that is captured by the radar of the satellites. They used sematic segmentation to point out the oil spill in the terms of the boundary over them that reduced the influence of the noise. The look-alike images are harder to detect the oil spill, in order to detect the oil spill in look-alike more images is needed.

2.3 A DEEP LEARNING BASED OIL SPILL DETECTOR USING SENTINEL-1 SAR IMAGERY

The object detector was trained with a total of 5930 Sentinel-1 images-9768 oil spills from different sources and with different sizes collected and labelled as oil objects. Using large dataset, the capability of detecting deliberate oil spills is highlighted. Oil spills near look-alikes were also distinguished well, but oil spills covered by lookalikes were not always clearly detected. It detected the whole look-alike and the oil spill covered by it as one big oil spill. The Eastern Mediterranean Sea has been known as an oil pollution hotspot due to its heavy marine traffic and an increasing number of oil and gas exploration activities. To provide automatic detection of oil pollution from not only maritime accidents but also deliberate discharges in this region, a deep learning-based object detector was developed utilizing freely available Sentinel-1 Synthetic Aperture Radar (SAR) imagery. A total of 9768 oil objects were collected from 5930 Sentinel-1 scenes from 2015 to 2018 and used for training and validating the object detector and evaluating its performance. The trained object detector has an average precision (AP) of 69.10% and 68.69% on the validation and test sets, respectively, and it could be applied for building an early-stage oil contamination surveillance system.

2.4 DETECTION AND ANALYSIS OF OIL SPILL USING IMAGE PROCESSING

Every image will be resized and segmented on board. When oil is detected, the system will gather all related information. On board cameras, will capture raw aerial images and UAV hardware will provide altitude, flight speed, flight path and GPS coordinates to help identify oil spills.The datasets for the model should be increased in order to increase the accuracy of the detection, if not the model gets confused to detection of the oil spills across the oceans and to maximize the detection. Thousands of oil spills occur every year on offshore oil production platforms. Moreover, ships that crosses rivers to reach the harbor cause spills each year. The current study focuses on IRAQI marine and rivers, especially Al-Bakr, Khor al-Amaya, ABOT oil terminal and SHAT AL-Arab river inside Al Başrah oil terminal. In order to mitigate and manage oil spill impacts, an unmanned aerial vehicle has proven to be a valuable tool in mitigating and managing incidents. To achieve high accuracy, the objective of the current research is to analyze captured images for rivers, identify oil pollution and determine its location. The images were taken from the Iraqi Regional Organization for the Protection, General Company for Ports of Iraq, Iraqi Ministry of Environment and online websites. In the current paper is presented a software framework for detecting oil spills, pollution in rivers and other kinds of garbage. The framework based on artificial intelligence is divided into two parts: a training model and an operational model. In the training model part, a machine learning model is applied, which is one of the fastest and most accurate methods, integrated inside PipelineML ML. Thus, the object detection technique used can identify one or more categories of objects in a picture or video. Furthermore, the locations of objects can be identified with the help of neural networks. In the operational mode, models can identify oil spills in images.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

To build YOLO (You Only Look Once) RCNN(faster region-based convolutional neural network), SSD (single-shot detector), SIMRDWN (satellite imagery multiscale rapid detection with windowed networks).The results showed that SIMRDWN has an accuracy of 97% on high-resolution images, while Faster RCNN has an accuracy of 95.31% on the standard resolution (1000 X 600). YOLOv3 has an accuracy of 94.20% on standard resolution (416 X 416) while on the other hand SSD has an accuracy of 84.61% on standard resolution (300 X 300). When it comes to speed and efficiency, YOLO is the obvious leader. In real-time surveillance, SIMRDWN fails. When YOLO takes 170 to 190 milliseconds to perform a task, SIMRDWN takes 5 to 103 milliseconds.

3.1.1 DISADVANTAGES

- Lack of datasets
- Model collapses at a point
- Complicated
- Compromised accuracy
- Separating Characters
- No Circumference

3.2 PROPOSED SYSTEM

The proposed system uses 2D Convolutional Neural Network to detect the oil spills across the water bodies, consisting 2D convolutional operation with kernel size (k) and number of feature maps (m) at each convolutional layer for hyperspectra image classification. The convolutional kernel moves in 2-direction (x, y) to calculate the convolutional output, Uses: Image Classification, Generating New Images, Image Inpainting, Image Colorization. The system also uses GAN (Generative Adversarial Networks) the powerful class of neural networks for un-supervised learning. GAN can analyze, capture and copy or create alternative data within the dataset.

3.2.1 ADVANTAGES

- Increase in Dataset
- Classes can be added according to the requirement
- Approximate Oil Spill spread across the ocean is estimated
- Boundary of Oil spill is measured

3.3 SYSTEM REQUIREMENTS

A 64-bit PC can handle larger amounts of information than a 32-bit system. Since it can use RAM-4GB and up-a 64bit computer can be more responsive when you are running lots of programs at once. 1GB of RAM is considered low and most desktop and laptop computers comes stock with at least 2GB, but usually more. Most software defines two sets of system requirements i.e., minimum and recommended.

3.3.1 SOFTWARE REQUIREMENTS

A software requirement specification is a description of a software system to be developed. It lays out functional and non-functional requirements and it also describes the operating system and tool used in the system and they are:

REQUIREMENT	SPECIFICATION
Free Space	200GB
GPU	12 hours
IDE	Google Colab/ Jupyter Notebook

TABLE 3.1- SOFTWARE REQUIREMENTS

3.3.2 HARDWARE REQUIREMENTS

Hardware specifications are technical description of the computer's components and capabilities. Processor speed, model and manufacturer, etc. So the hardware components required for the proposed system are:

REQUIREMENT	SPECIFICATION
Processor	Intel® Core™ i5 or AMD
Clock Speed	3.40 GHz
RAM	16 GB
System Type	64 Bit OS
Operating System	Linux / Mac OS / Windows 10

TABLE 3.2- HARDWARE REQUIREMENTS

3.4 LANGUAGE SPECIFICATION

3.4.1 The PYTHON Programming Language:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and

procedural, it also has a comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is opensource software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

3.4.2 Features of Python

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small

core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the C Python reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. C Python is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar. Some other important features are:

- Beginner's Language
- Simple and Easy to Learn
- Interpreted Language
- Cross-platform language

- Free and Open Source
- Object-Oriented language
- Extensive Libraries
- Integrated

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called un-pythonic.

3.4.3 Google Colab

Google is quite aggressive in AI research. Over many years, Google developed AI framework called TensorFlow and a development tool called Colaboratory. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply Colab.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.

Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

CHAPTER 4

SYSTEM ARCHITECHTURE

4.1 ARCHITECTURE DIAGRAM

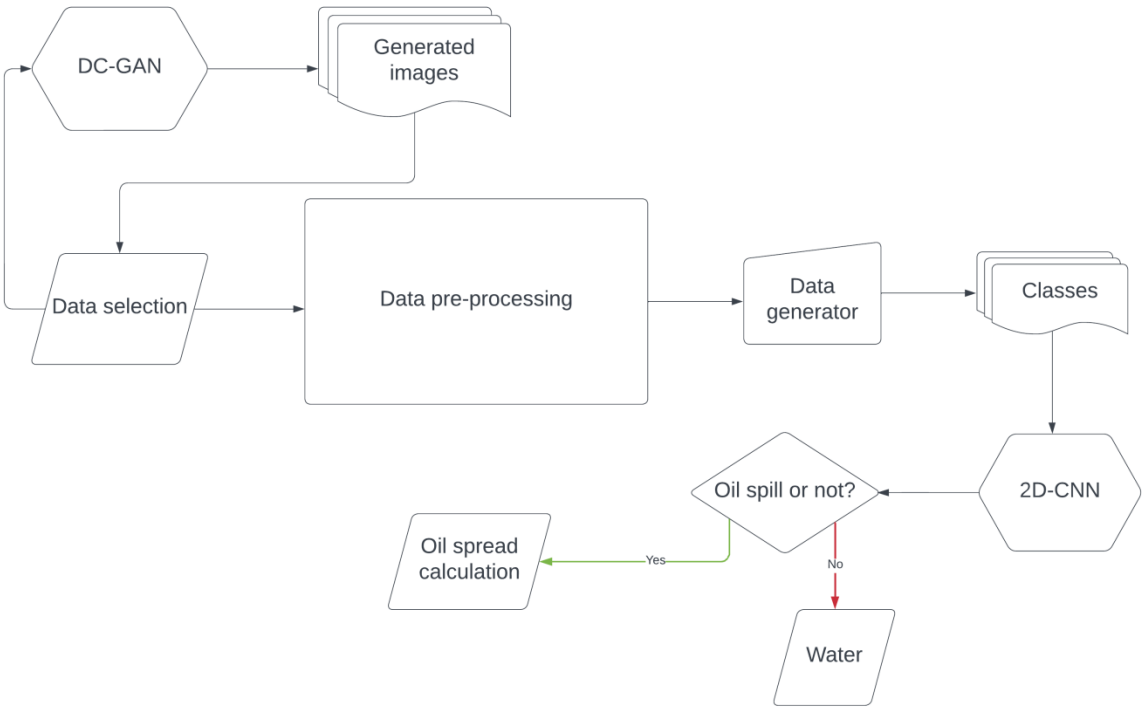


Fig 4.1 ARCHITECTURE DIAGRAM FOR OIL SPILL DETECTION USING 2d CNN AND GAN

4.2 ARCHITECTURE DESCRIPTION

Data plays a vital role in detecting any type of problem or required target. We have to collect sufficient amount of data to train and test our models. These collected data are imported to the oil spill detection and generative models in order to detect oil spill and generate new images for the available datasets respectively. In comparing to the overall studies, everyone who proposed their model for oil spill have faces a tough situation while collecting datasets and we are trying to generate the datasets on our own to overcome this difficulty. In a 2D CNN, the input data is processed through a series of convolutional layers, pooling layers, and fully connected layers. Convolutional layers use a set of filters or kernels to scan the input data and identify features or patterns in the data. Pooling layers downsample the output of the convolutional layers to reduce the dimensionality of the feature maps and help prevent overfitting. Fully connected layers are used for classification or regression tasks. The filters used in convolutional layers are learned through training the network on a large dataset of input-output pairs. The goal is to learn filters that can accurately detect the relevant features in the input data that are important for the task at hand. The generator takes a random noise vector as input and generates a new sample that is intended to be similar to the input data. The discriminator takes both real and generated samples as input and attempts to distinguish between them. The two networks are trained together in a process called adversarial training, in which the generator tries to fool the discriminator, and the discriminator tries to correctly identify the real samples from the generated ones. GANs have been used to generate a wide variety of data types, including images, music, and text. They have been particularly successful in generating photo-realistic images, and have been used in applications such as image synthesis.

4.3 NETWORK ARCHITECTURE

The Convolution Neural Networks are mostly used in the deep learning enthusiasts in recent years. Their main purpose is classification of images and videos. Kernels are the filter which acts as the building blocks of the CNN's. Kernels are used to extract the dominant features or required segment of the images using the convolution layers present in it. There are 1D-CNN, 2D-CNN and 3D-CNN. For our oil spill detection model we are using 2D-CNN. 2 Dimensional Convolutional Neural Networks are the type of neural networks in which they deals with the 2D images for detection, classification etc. There are filters called kernel which moves in 2 directions of the input images. Basically CNN consists of three layers they are: convolutional layer, pooling layer and fully connected layer.

Type	Kernel size/stride	Input size
Convolution	3 × 3/2	299 × 299 × 3
Convolution	3 × 3/1	149 × 149 × 32
Convolution	3 × 3/1	147 × 147 × 32
Pooling	3 × 3/2	147 × 147 × 64
Convolution	3 × 3/1	73 × 73 × 64
Convolution	3 × 3/2	71 × 71 × 80
Convolution	3 × 3/1	35 × 35 × 192
Inception module	Three modules	35 × 35 × 288
Inception module	Five modules	17 × 17 × 768
Inception module	Two modules	8 × 8 × 1,280
Pooling	8×8	8 × 8 × 2,048
Linear	Logits	1 × 1 × 2,048
Softmax	Output	1 × 1 × 1,000

Fig 4.2 Architecture of Inception V3

.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 LIST OF MODULES

5.1.1 Data Selection

5.1.2 Data pre-processing

5.1.3 Build and Train model(2DCNN-InceptionV3)

5.1.4 Image Validation

5.1.5 Performance Evaluation

5.1.6 Oil spread calculation

5.1.1 DATA SELECTION

According to the type of models and processes, we have to select the type of data which can be used to train and test the model. The data might be CSV (excel sheet), images (jpg, jpeg, png) files and these can be collected from the various Github repositories, Kaggle, Google datasets and Roboflow. For our detection models and generative models we are using image files. From the above mentioned websites we have retrieved datasets and we also used python Bing image downloader where the images can be downloaded easily in higher quantity. The only problem with Bing downloader is they download all the images that have been labeled as oil spill.

DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator consists of strided convolution layers, batch normalization layers, and LeakyRelu as activation function. It takes a 3x64x64 input image. The generator consists of convolutional-transpose layers, batch normalization layers, and ReLU activations. DCGAN that uses Deep Conv Nets to have a stable architecture and better results. The Generator in GAN uses a fully connected network, whereas DCGAN uses a Transposed Convolutional network to up sample the images.

In our project, we have used this DCGAN to increase the number of images in our datasets so that the deep learning models accuracy increases in detecting the oil spills. The **accuracy of the models = Number of training and testing images or number of images in the dataset**. Both InceptionV3 and VGG16 detects oil spill but the efficiency is totally depends on the datasets, in our case it will be tough to collect datasets so to overcome this; we are trying to use GAN in order to create more images with the already available images.

In DCGAN, there are two types of models in them they are generator and discriminator: The generators takes care of generating or creating new images from the existing images and the discriminator finds out whether the image is generated or from the real dataset. The generator keeps generating the images of the imported dataset until the discriminator finds out that the images are generated by the generator.

The oil spill images of the datasets are imported to the DCGAN then the generator starts generating new images after every epoch, the discriminator starts detecting the generated images or real image. Once the discriminator detects the generated oil spill image the generator stops generating the oil spill images.

```
1 generator.summary()
```

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 32768)             4227072

re_lu_12 (ReLU)              (None, 32768)             0

reshape_3 (Reshape)          (None, 8, 8, 512)         0

conv2d_transpose_9 (Conv2DT  (None, 16, 16, 256)       2097408
ranspose)

re_lu_13 (ReLU)              (None, 16, 16, 256)       0

conv2d_transpose_10 (Conv2D  (None, 32, 32, 128)       524416
Transpose)

re_lu_14 (ReLU)              (None, 32, 32, 128)       0

conv2d_transpose_11 (Conv2D  (None, 64, 64, 64)        131136
Transpose)

re_lu_15 (ReLU)              (None, 64, 64, 64)        0

conv2d_9 (Conv2D)            (None, 64, 64, 3)         3075

=================================================================
Total params: 6,983,107
Trainable params: 6,983,107
Non-trainable params: 0
```

Fig 5.11 Architecture of GAN

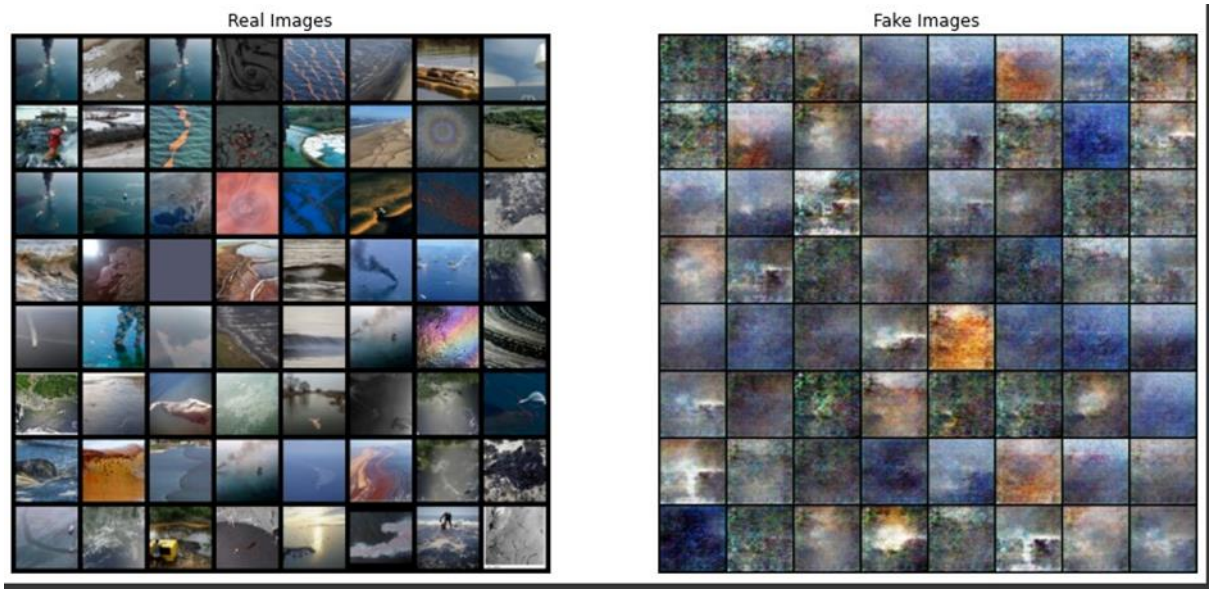


Fig 5.12 Generated images by GAN

Fig 5.12 shows that the DCGAN generates images after high amount of epoch and time but the images are not clear and they are noisy. This can be tried with even higher epoch and fine tuning the DCGAN model.

5.1.2 DATA PRE-PROCESSING

Data pre-processing is the process in which the datasets used in the model is cleaned that is un-related images are removed and related images are alone taken into the account. In this phase, data generators are used to label the images into several classes such as [‘Oil’,’ Water’] and many more classes can also be added. The classes specify the different types of images used for the training of the model. If we add more classes to them, we can be able increase the accuracy of the model.

5.1.3 BUILD AND TRAIN MODEL

2D CNN two pre-trained models for oil spill detection they are InceptionV3. Inception V3 has a deeper network in image classification, less computational costs. Building 2-Dimensional Convolutional Neural Networks layer by layer like convolutional layer, pooling layer, and fully connected layer. In 2D CNN there are

many pre-trained models, in this project-InceptionV3 is used, which is one of the efficient models for image classification.

Even though we execute two different models in 2D-CNN, the process occurs in the both the models are the same. We are going to check which provides high accuracy for our problem. In the convolutional layer, the image features are extracted and converted into a matrix called as kernel consists of values such as height and width of the images. After this, comes the pooling layer where the down sampling of images takes place for every image that comes from the convolutional layer. These two layers form a feature map that consists of values of the particular feature of the images which can be used for the training purposes. The fully connected layers in the 2D-CNN are the place where the decision making occurs whether the image consists of features of oil. If the image consists of oil features then the oil spread calculation is done in order to find the volume of oil spread along the ocean. If the image doesn't consist of features of oil spill then it is detected as water and no spill calculation takes place.

5.1.4 IMAGE VALIDATION

The image features are extracted by the convolutional neural networks in which the 2D-CNN model learns from the training images. In the validation state, the 2D-CNN model detects if the testing set of images consists of oil spill or not. If the oil spill is present then volume of oil spread across the ocean is calculated. The validations results are mentioned below:

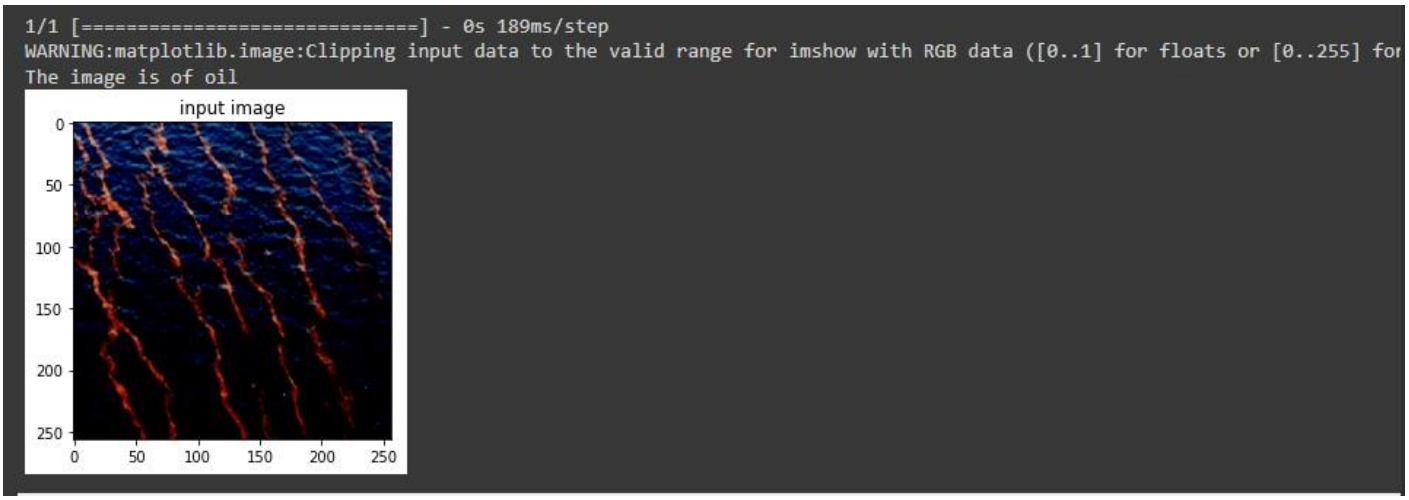


Fig 5.13 Validation Result 1

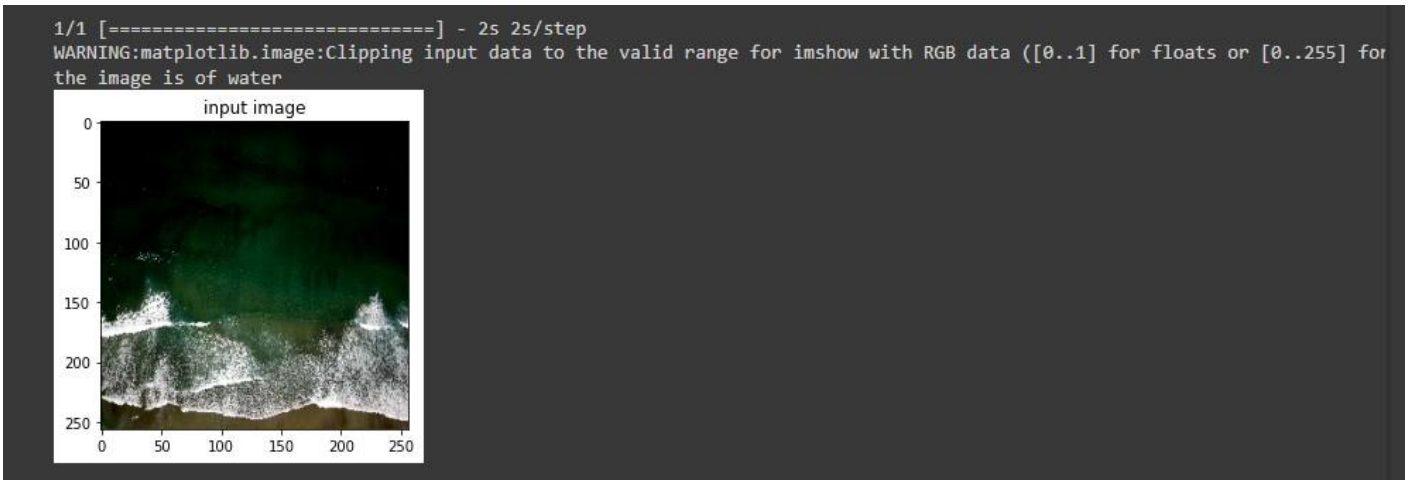


Fig 5.14 Validation Result 2

5.1.5 PERFORMANCE EVALUATION

Performance evaluation is the process in which the models efficiency is calculated and measured if the models can be further improved this helps us to improve the detection of the oil spill models. This can be measured in several ways; we can see them in depth below.

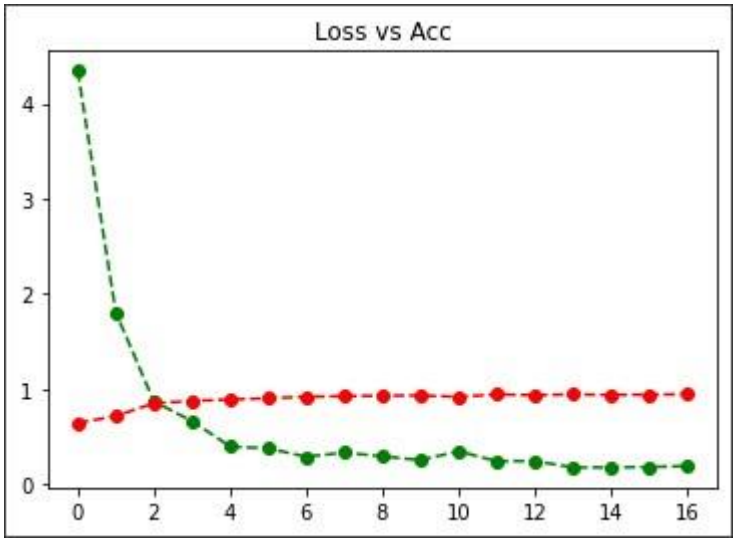


Fig 5.15 Loss vs. Accuracy graph of InceptionV3

Confusion matrix measures the quality of predictions from a classification or detection models by looking at how much detection are true and how much detection is false. Confusion matrix consists of

- True positives(TP)
- True negatives(TN)
- False positives(FP)
- False negatives(FN)

True positives: The model detects oil spill and it is true, the model detects the image consists of oil spill and the image consists of the oil spill.

True negatives: The model detects there is no oil spill and it is true, the model detects the image consists of water and the image consists of the water.

False positives: The model detects there is no oil spill and it is false, the model detects the image consists of water and the image consists of the oil spill.

False negatives: The model detects there is oil spill and it is false, the model detects the image consists of oil spill and the image consists of the water.

Cm [0][0]-TP=Number of oil spill labeled as oil spill

Cm [0][1]-TN=Number of water labeled as water

Cm [1][0]-FP=Number of oil spill labeled as water

Cm [1][1]-FN=Number of water labeled as oil spill, let's see the accuracy of this inceptionV3 model.

The accuracy of the model can be calculated by the confusion matrix in which True positive, True negative, False positive, False negative. True positive- oil spill as oil spill, True negative- water as water, False positive-oil spill as water and False negative-water as oil spill.

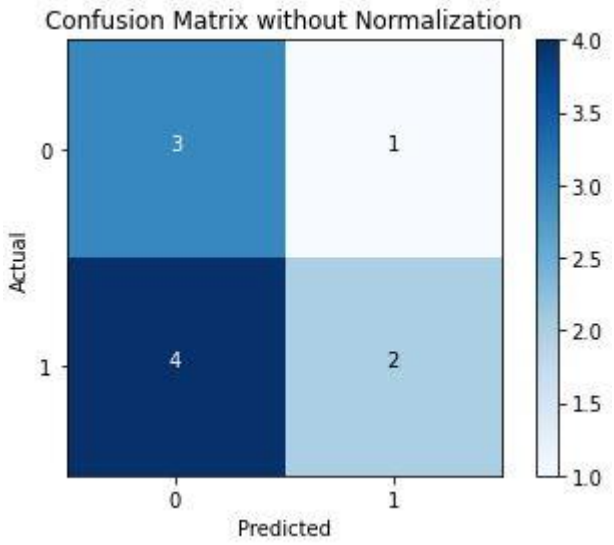


Fig 5.16 Confusion matrix of InceptionV3

ACCURACY:

Accuracy is one of the performance metrics that describes the performance of the model with all classes (oil, water). The accuracy can be used for the model which equally focuses on both the classes. In our model we consider accuracy is equal to the ratio between the numbers of predictions to the total number of predictions.

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{True_{positive} + True_{negative}}}{\boldsymbol{True_{positive} + True_{negative} + False_{positive} + False_{negative}}}$$

Using the above formula we will be able to calculate the accuracy of the model with the values in the confusion matrix. The result variable holds the sum of

the true positive and true negative over the sum of all the values in the confusion matrix. The value of accuracy, we get as result is 0.5 then the model is 50% accurate in detecting or classifying the exact classes.

PRECISION:

Precision is considered as the ratio between the numbers of positive output classified to the total number of output classified as positive (correct or incorrect). The precision is meant for calculating the accuracy of the model detecting the output as positive.

$$\boldsymbol{Precision} = \frac{\boldsymbol{True_{positive}}}{\boldsymbol{True_{positive}} + \boldsymbol{False_{positive}}}$$

When the model does any incorrect positive detections, or few proper positive detections then the denominator is increased in order to reduce the precision. The precision will be high when the model does correct positive detections or the model does little wrong positive detection. The precision mainly reveals us the truthfulness of the model. The precision of our model is 0.4285 that is the model is 42.85% accurate in detecting positive samples.

RECALL:

The recall is defined as the ratio between the numbers of positive output detected as positive to the total number of positive output. The recall reveals us the ability of model to detect positive samples. If the recall is high, then the possibilities of detecting the positive samples are high.

$$\boldsymbol{Recall} = \frac{\boldsymbol{True_{positive}}}{\boldsymbol{True_{positive}} + \boldsymbol{False_{negative}}}$$

The recall mainly focuses on the detection of the positive samples which is independent of detection of the negative samples. In recall we can neglect the negative samples instead we have to focus more on the positive samples. The recall of our model is 0.75 that is the model is 75% accurate in detecting positive samples but the problem is sometimes the model detects the negative samples as positive.

5.1.6 OIL SPREAD CALCULATION

Calculating the volume of oil spill over the ocean we have to add boundary to spot the oil spill, so we are adding two types of boundaries and then calculating the oil spread over the ocean.

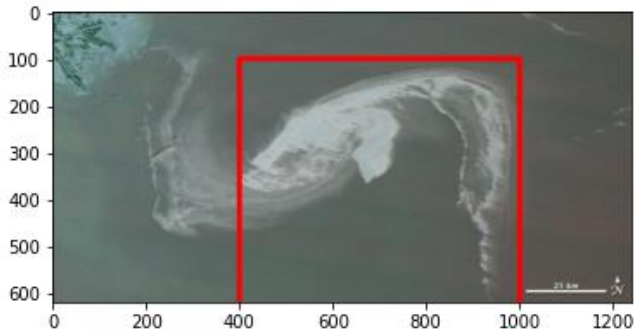


Fig 5.17 Rectangular boundary

Let us calculate the volume of oil spill using the below formula in V is the volume of the oil spread along the ocean, l is the length of the rectangular boundary and b is the breadth of the rectangular boundary, v is the volume of the oil in water and t is the time taken during oil spill,

$$V=lb\times v\times t \text{ ---------- (1)}$$

In one barrel of crude oil, 42 gallon crude oil barrel contains approximately 45 gallons of refined crude oil and oil based products per barrel.

$$\text{So } v=45 \text{ gallon---------- (2)}$$

Time taken by the oil to reach the surface of the ocean is ranging from 2 months to 15 years; we are considering it to be 2 months in which there are 60 days.

Let t= 60 we can convert it to hours in 1 day =24hours then 60 days =60 × 24=1440 hours.

t=1440 hours---------- --- (3)

As we can see the length of the rectangular boundary from the outline can be calculated such that,

l=1000-400=600 km--------- (4)

b=600-500=100 km---------- (5)

Substituting (5), (4), (3) and (2) in (1),

V=600×100×45×1440

V=3,888,000,000 kilo gallon per hour

Approximately because the rectangular boundary may miss out some oil spread over the ocean so it can be maximized in order to attain the exact volume of oil spreading over the ocean.

Let's calculate the volume of oil spread along the ocean when it is covered by the circular boundary. V is the volume of the oil spread along the ocean, r is the radius of the circular boundary, v is the volume of oil in water and t is the time taken by the oil to reach the surface. So the formula will become,

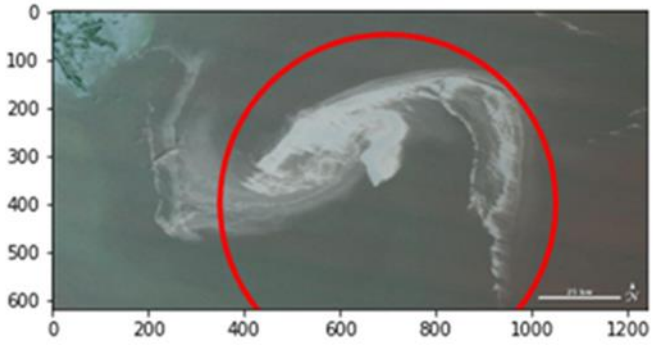


Fig 5.18 Circular Boundary

$$\boldsymbol{V = \pi r^2 \times v \times t}\text{------- (1)}$$

Let π =3.14 ----- (2)

v=45gallon ----- (3)

t=1440 hours ----- (4)

(v and t are the same values that are used in rectangular boundary to know the difference)

r is the radius of the circular boundary with label we can get the diameter of the circular boundary that is 1400 km and radius will be half the diameter so r=700 km---------- (5)

Substituting (5), (4), (3), (2) in (1),

$$V = 3.14 \times 700 \times 700 \times 45 \times 1440$$

$$V = 9.97012e10\ kilo\ gallon\ per\ hour$$

The volume of oil spread calculating in the circular boundary is greater than the volume of oil spread calculated in the rectangular boundary. Generally oil spill won't have a fixed shape and edges so circular boundary will be able to reach the oil spreading over the ocean than the rectangular boundary which has four sides with edges.

The volume of oil spread calculated using circular boundary is also approximate as you can see from the image that the some part of spill is not covered by the boundary. This case can be cleared by maximizing the circular boundary but that may can include some water as well so in order to spot the irregular shaped oil spill we can try free shaped boundary which only covers the oil spill and not along with the water.

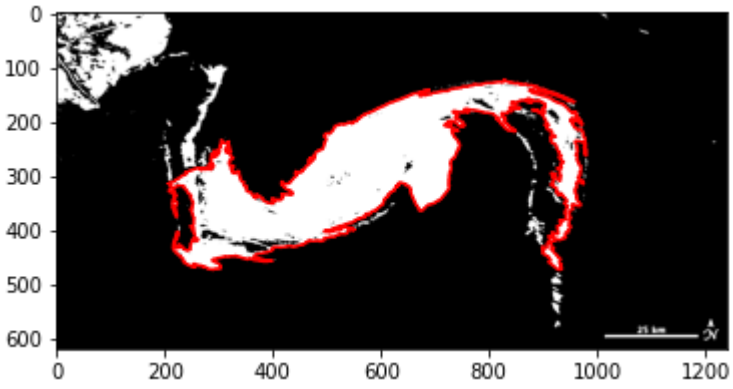


Fig 5.19 Free boundary

Fig 5.17 is the perfect example for the free boundary of the oil spill, which can be used to cover the overall irregular shaped oil spills but the problem is using the formula for the free boundary but in case of rectangular boundary we used rectangular area formula for the volume of the oil spread and in the case of circular boundary we

used circle are formula for the volume of the oil spread. While using free boundary we have to use integral calculus to find the area of the irregular shapes and it might be challenging to find the area.

is the perfect example for the free boundary of the oil spill, which can be used to cover the overall irregular shaped oil spills but the problem is using the formula for the free boundary but in case of rectangular boundary we used rectangular area formula for the volume of the oil spread and in the case of circular boundary we used circle are formula for the volume of the oil spread. While using free boundary we have to use integral calculus to find the area of the irregular shapes and it might be challenging to find the area.

CHAPTER 6

SAMPLE CODING

Following is the command for the accessing google drive from google colab

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

DC GAN

Importing Data from the dataset

```
import torch
import torchvision.transforms as transforms
import torchvision.datasets as dset
root = '/content/drive/MyDrive/Oil Spill Detection for gan'
def get_celeba(params):
    transform = transforms.Compose([
        transforms.Resize(params['imsize']),
        transforms.CenterCrop(params['imsize']),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5),
            (0.5, 0.5, 0.5))])
    dataset = dset.ImageFolder(root=root, transform=transform)
    dataloader = torch.utils.data.DataLoader(dataset,
        batch_size=params['bsize'],
        shuffle=True)
    return dataloader
```

Creation of Generator and Discriminator

```
import torch
import torch.nn as nn
import torch.nn.functional as F
def weights_init(w):
    classname = w.__class__.__name__
```

```
 if classname.find('conv') != -1:
        nn.init.normal_(w.weight.data, 0.0, 0.02)
    elif classname.find('bn') != -1:
        nn.init.normal_(w.weight.data, 1.0, 0.02)
        nn.init.constant_(w.bias.data, 0)
class Generator(nn.Module):
    def __init__(self, params):
        super().__init__()
        self.tconv1 = nn.ConvTranspose2d(params['nz'], params['ngf']*8,
            kernel_size=4, stride=1, padding=0, bias=False)
        self.bn1 = nn.BatchNorm2d(params['ngf']*8)
        self.tconv2 = nn.ConvTranspose2d(params['ngf']*8, params['ngf']*4,
            4, 2, 1, bias=False)
        self.bn2 = nn.BatchNorm2d(params['ngf']*4)
        self.tconv3 = nn.ConvTranspose2d(params['ngf']*4, params['ngf']*2,
            4, 2, 1, bias=False)
        self.bn3 = nn.BatchNorm2d(params['ngf']*2)
        self.tconv4 = nn.ConvTranspose2d(params['ngf']*2, params['ngf'],
            4, 2, 1, bias=False)
        self.bn4 = nn.BatchNorm2d(params['ngf'])
        self.tconv5 = nn.ConvTranspose2d(params['ngf'], params['nc'],
            4, 2, 1, bias=False)
    def forward(self, x):
        x = F.relu(self.bn1(self.tconv1(x)))
        x = F.relu(self.bn2(self.tconv2(x)))
        x = F.relu(self.bn3(self.tconv3(x)))
        x = F.relu(self.bn4(self.tconv4(x)))
        x = F.tanh(self.tconv5(x))
        return x
class Discriminator(nn.Module):
    def __init__(self, params):
        super().__init__()
        self.conv1 = nn.Conv2d(params['nc'], params['ndf'],
            4, 2, 1, bias=False)
```

```
        self.conv2 = nn.Conv2d(params['ndf'], params['ndf']*2,
            4, 2, 1, bias=False)
        self.bn2 = nn.BatchNorm2d(params['ndf']*2)
        self.conv3 = nn.Conv2d(params['ndf']*2, params['ndf']*4,
            4, 2, 1, bias=False)
        self.bn3 = nn.BatchNorm2d(params['ndf']*4)
        self.conv4 = nn.Conv2d(params['ndf']*4, params['ndf']*8,
            4, 2, 1, bias=False)
        self.bn4 = nn.BatchNorm2d(params['ndf']*8)
        self.conv5 = nn.Conv2d(params['ndf']*8, 1, 4, 1, 0, bias=False)
    def forward(self, x):
        x = F.leaky_relu(self.conv1(x), 0.2, True)
        x = F.leaky_relu(self.bn2(self.conv2(x)), 0.2, True)
        x = F.leaky_relu(self.bn3(self.conv3(x)), 0.2, True)
        x = F.leaky_relu(self.bn4(self.conv4(x)), 0.2, True)
        x = F.sigmoid(self.conv5(x))
        return x
```

Training the dataset and loss during training

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random
import os
seed = 369
random.seed(seed)
torch.manual_seed(seed)
print("Random Seed: ", seed)
params = {
    "bsize" : 128,
```

```
    'imsize' : 64,
    'nc' : 3,
    'nz' : 100,
    'ngf' : 64,
    'ndf' : 64,
    'nepochs' : 10,
    'lr' : 0.0002
    'beta1' : 0.5,
    'save_epoch' : 2}
device = torch.device("cuda:0" if(torch.cuda.is_available()) else "cpu")
print(device, " will be used.\n")
dataloader = get_celeba(params)
sample_batch = next(iter(dataloader))
plt.figure(figsize=(8, 8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(
    sample_batch[0].to(device)[ : 64], padding=2, normalize=True).cpu(), (1, 2, 0)))
plt.show()
netG = Generator(params).to(device)
netG.apply(weights_init)
print(netG)
netD = Discriminator(params).to(device)
netD.apply(weights_init)
print(netD)
criterion = nn.BCELoss()
fixed_noise = torch.randn(64, params['nz'], 1, 1, device=device)
real_label = 1
fake_label = 0
optimizerD = optim.Adam(netD.parameters(), lr=params['lr'], betas=(params['beta1'],
0.999))
optimizerG = optim.Adam(netG.parameters(), lr=params['lr'], betas=(params['beta1'],
0.999))
img_list = []
```

```
G_losses = []
D_losses = []
iters = 0
print("Starting Training Loop...")
print("-"*25)
for epoch in range(params['nepochs']):
    for i, data in enumerate(dataloader, 0):
        real_data = data[0].to(device)
        b_size = real_data.size(0)
        netD.zero_grad()
        label = torch.full((b_size, ), real_label, device=device)
        output = netD(real_data).view(-1)
        errD_real = criterion(output.float(), label.float())
        errD_real.backward()
        D_x = output.mean().item()
        noise = torch.randn(b_size, params['nz'], 1, 1, device=device)
        fake_data = netG(noise)
        label.fill_(fake_label  )
        output = netD(fake_data.detach()).view(-1)
        errD_fake = criterion(output.float(), label.float())
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        errD = errD_real + errD_fake
        optimizerD.step()
        netG.zero_grad()
        label.fill_(real_label)
        output = netD(fake_data).view(-1)
        errG = criterion(output.float(), label.float())
        errG.backward()
        D_G_z2 = output.mean().item()
        optimizerG.step()
        if i%50 == 0:
            print(torch.cuda.is_available())
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f
```

```
%.4f'
                % (epoch, params['nepochs'], i, len(dataloader),
                  errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))
        G_losses.append(errG.item())
        D_losses.append(errD.item())
        if (iters % 100 == 0) or ((epoch == params['nepochs']-1) and (i == len(dataloader)-
1)):
            with torch.no_grad():
                fake_data = netG(fixed_noise).detach().cpu()
            img_list.append(vutils.make_grid(fake_data, padding=2, normalize=True))
        iters += 1
    os.makedirs('model', exist_ok=True)
    if epoch % params['save_epoch'] == 0:
        torch.save({
            'generator' : netG.state_dict(),
            'discriminator' : netD.state_dict(),
            'optimizerG' : optimizerG.state_dict(),
            'optimizerD' : optimizerD.state_dict(),
            'params' : params
            }, 'model/model_epoch_{}.pth'.format(epoch))
torch.save({
            'generator' : netG.state_dict(),
            'discriminator' : netD.state_dict(),
            'optimizerG' : optimizerG.state_dict(),
            'optimizerD' : optimizerD.state_dict(),
            'params' : params
            }, 'model/model_final.pth')
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
```

```
plt.show()

fig = plt.figure(figsize=(8,8))
plt.axis("off")
ims = [[plt.imshow(np.transpose(i,(1,2,0)), animated=True)] for i in img_list]
anim = animation.ArtistAnimation(fig, ims, interval=1000, repeat_delay=1000, blit=True)
plt.show()
anim.save('celeba.gif', dpi=80, writer='imagemagick')

Creating the model and generating the images
import torch
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random
device = torch.device("cuda:0" if(torch.cuda.is_available()) else "cpu")
load_path = '/content/model/model_final.pth'
num_output = 64
state_dict = torch.load(load_path)
params = state_dict['params']
netG = Generator(params).to(device)
netG.load_state_dict(state_dict['generator'])
print(netG)
print(num_output)
noise = torch.randn(int(num_output), params['nz'], 1, 1, device=device)
with torch.no_grad():
    generated_img = netG(noise).detach().cpu()
plt.axis("off")
plt.title("Generated Images")
plt.imshow(np.transpose(vutils.make_grid(generated_img, padding=2, normalize=True),
(1,2,0)))
plt.show()
```

Downloading the generated images

```
from google.colab import files
import os
import shutil
folder_name = "generated_images"
folder_path = os.path.join(os.getcwd(), folder_name)
if not os.path.exists(folder_path):
    os.makedirs(folder_path)
for i in range(fake_data.shape[0]):
    image_filename = os.path.join(folder_path, "image" + str(i) + ".jpg")
    while os.path.exists(image_filename):
        i += 1
        image_filename = os.path.join(folder_path, "image" + str(i) + ".jpg")
    vutils.save_image(vutils.make_grid(fake_data[i], padding=2, normalize=True),
image_filename)
    try:
      shutil.move(image_filename, folder_path)
    except shutil.Error:
      print(f"Error: Destination path '{image_filename}' already exists.")
shutil.make_archive(folder_name, 'zip', folder_path)
files.download(folder_name + '.zip')
```

ESRGAN

Importing and downloading the libraries needed

```
!git clone https://github.com/xinntao/Real-ESRGAN.git
%cd Real-ESRGAN
!pip install basicsr
!pip install facexlib
!pip install gfpgan
!pip install -r requirements.txt
!python setup.py develop
!git clone https://github.com/cszn/BSRGAN.git
```

```
!rm -r SwinIR
!git clone https://github.com/JingyunLiang/SwinIR.git
!pip install timm
!wget https://github.com/cszn/KAIR/releases/download/v1.0/BSRGAN.pth -P BSRGAN/model_zoo
!wget https://github.com/xinntao/Real-ESRGAN/releases/download/v0.1.0/RealESRGAN_x4plus.pth -P experiments/pretrained_models
!wget https://github.com/JingyunLiang/SwinIR/releases/download/v0.0/003_realSR_BSRGAN_DFO_s64w8_SwinIR-M_x4_GAN.pth -P experiments/pretrained_models
!wget https://github.com/JingyunLiang/SwinIR/releases/download/v0.0/003_realSR_BSRGAN_DFOWMFC_s64w8_SwinIR-L_x4_GAN.pth -P experiments/pretrained_models
```

Importing Data

```
import os
import glob
from google.colab import files
import shutil
print(' Note1: You can find an image on the web or download images from the RealSRSet (proposed in BSRGAN, ICCV2021) at https://github.com/JingyunLiang/SwinIR/releases/download/v0.0/RealSRSet+5images.zip .\n Note2: You may need Chrome to enable file uploading!\n Note3: If out-of-memory, set test_patch_wise = True.\n')
test_patch_wise = False
!rm -r BSRGAN/testsets/RealSRSet
upload_folder = 'BSRGAN/testsets/RealSRSet'
result_folder = 'results'
if os.path.isdir(upload_folder):
    shutil.rmtree(upload_folder)
if os.path.isdir(result_folder):
    shutil.rmtree(result_folder)
os.mkdir(upload_folder)
os.mkdir(result_folder)
```

```
uploaded = files.upload()
for filename in uploaded.keys():
  dst_path = os.path.join(upload_folder, filename)
  print(f'move {filename} to {dst_path}')
  shutil.move(filename, dst_path)



Model training
!rm -r results
if not test_patch_wise:
  %cd BSRGAN
  !python main_test_bsrgan.py
  %cd ..
  shutil.move('BSRGAN/testsets/RealSRSet_results_x4', 'results/BSRGAN')
if test_patch_wise:
  !python inference_realesrgan.py -n RealESRGAN_x4plus --input
BSRGAN/testsets/RealSRSet -s 4 --output results/realESRGAN --tile 800
else:
  !python inference_realesrgan.py -n RealESRGAN_x4plus --input
BSRGAN/testsets/RealSRSet -s 4 --output results/realESRGAN


if test_patch_wise:
  !python SwinIR/main_test_swinir.py --task real_sr --model_path
experiments/pretrained_models/003_realSR_BSRGAN_DFO_s64w8_SwinIR-
M_x4_GAN.pth --folder_lq BSRGAN/testsets/RealSRSet --scale 4 --tile 800
else:
  !python SwinIR/main_test_swinir.py --task real_sr --model_path
experiments/pretrained_models/003_realSR_BSRGAN_DFO_s64w8_SwinIR-
M_x4_GAN.pth --folder_lq BSRGAN/testsets/RealSRSet --scale 4
shutil.move('results/swinir_real_sr_x4', 'results/SwinIR')


if test_patch_wise:
  !python SwinIR/main_test_swinir.py --task real_sr --model_path
experiments/pretrained_models/003_realSR_BSRGAN_DFOWMFC_s64w8_SwinIR-
L_x4_GAN.pth --folder_lq BSRGAN/testsets/RealSRSet --scale 4 --large_model --tile
640
```

```
else:
  !python SwinIR/main_test_swinir.py --task real_sr --model_path
experiments/pretrained_models/003_realSR_BSRGAN_DFOWMFC_s64w8_SwinIR-
L_x4_GAN.pth --folder_lq BSRGAN/testsets/RealSRSet --scale 4 --large_model
shutil.move('results/swinir_real_sr_x4_large', 'results/SwinIR_large')
for path in sorted(glob.glob(os.path.join('results/SwinIR_large', '*.png'))):
  os.rename(path, path.replace('SwinIR.png', 'SwinIR_large.png'))
Generating the output
import cv2
import matplotlib.pyplot as plt
def display(img1, img2):
  total_figs = 5
  fig = plt.figure(figsize=(total_figs*12, 14))
  ax1 = fig.add_subplot(1, total_figs, 1)
  plt.title('Input image', fontsize=30)
  ax1.axis('off')
  ax2 = fig.add_subplot(1, total_figs, 2)
  plt.title('BSRGAN (ICCV2021) output', fontsize=30)
  ax2.axis('off')
  ax3 = fig.add_subplot(1, total_figs, 3)
  plt.title('Real-ESRGAN output', fontsize=30)
  ax3.axis('off')
  ax4 = fig.add_subplot(1, total_figs, 4)
  plt.title('SwinIR (ours) output', fontsize=30)
  ax4.axis('off')
  ax5 = fig.add_subplot(1, total_figs, 5)
  plt.title('SwinIR-Large (ours) output', fontsize=30)
  ax5.axis('off')
  ax1.imshow(img1)
  ax2.imshow(img2['BSRGAN'])
  ax3.imshow(img2['realESRGAN'])
  ax4.imshow(img2['SwinIR'])
  ax5.imshow(img2['SwinIR-L'])
```

```
def imread(img_path):
  img = cv2.imread(img_path)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  return img


print('Note: BSRGAN may be better at face restoration, but worse at building restoration because it uses different datasets in training.')
if test_patch_wise:
  print('BSRGAN does not support "test_patch_wise" mode for now. Set test_patch_wise = False to see its results.\n')
else:
  print('\n')
input_folder = upload_folder
result_folder = 'results/SwinIR'
input_list = sorted(glob.glob(os.path.join(input_folder, '*')))
output_list = sorted(glob.glob(os.path.join(result_folder, '*')))
for input_path, output_path in zip(input_list, output_list):
  img_input = imread(input_path)
  img_output = {}
  img_output['SwinIR'] = imread(output_path)
  img_output['SwinIR-L'] = imread(output_path.replace('SwinIR/', 'SwinIR_large/').replace('SwinIR.png', 'SwinIR_large.png'))
  if test_patch_wise:
    img_output['BSRGAN'] = img_output['SwinIR']*0+255
  else:
    img_output['BSRGAN'] = imread(output_path.replace('SwinIR', 'BSRGAN'))
  path = output_path.replace('/SwinIR/', '/realESRGAN/').replace('_SwinIR.png','_out{}'.format(os.path.splitext(input_path)[1]))
  if os.path.exists(path):
    shutil.move(path, path.replace('_out.', '_realESRGAN.'))
  img_output['realESRGAN'] = imread(path.replace('_out.', '_realESRGAN.'))
  display(img_input, img_output)
```

Downloading the output

```
zip_filename = 'SwinIR_result.zip'
if os.path.exists(zip_filename):
  os.remove(zip_filename)
os.system(f"zip -r -j {zip_filename} results/*")
files.download(zip_filename)
```

CNN

Importing and creating a dataset for the model

```
import warnings
warnings.filterwarnings("ignore")
import os
import shutil
import glob

TRAIN_DIR = "./DATASET"
ORG_DIR = "/content/drive/MyDrive/OilSpill Detection"
CLASS = ['Oil','Water']

for C in CLASS:
  DEST = os.path.join(TRAIN_DIR,C)
  if not os.path.exists(DEST):
    os.makedirs(DEST)
  for img_path in glob.glob(os.path.join(ORG_DIR, C)+"*"):
    SRC = img_path
    shutil.copy(SRC, DEST)
```

Model Building

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.layers import Dense, Flatten
from keras.models import Model
from keras.applications.inception_v3 import InceptionV3, preprocess_input
```

```
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
import keras
base_model = InceptionV3(input_shape=(256,256,3), include_top = False)
for layer in base_model.layers:
  layer.trainable = False

X = Flatten()(base_model.output)
X = Dense(units=2, activation='sigmoid')(X)
model = Model(base_model.input, X)
model.compile(optimizer='adam',loss = keras.losses.binary_crossentropy,
metrics=['accuracy'])
model.summary()
train_datagen = ImageDataGenerator(featurewise_center = True,
                                   rotation_range = 0.4,
                                   width_shift_range=0.3,
                                   horizontal_flip=True,
                                   preprocessing_function = preprocess_input,
                                   zoom_range=0.4,
                                   shear_range = 0.4)
train_data = train_datagen.flow_from_directory(directory="/content/Real-
ESRGAN/DATASET",
                                               target_size = (256,256),
                                               batch_size = 50)
train_data.class_indices



Visualizing the data
t_img, label = train_data.next()
t_img.shape
def plotImages(img_arr, label):
  for idx,img in enumerate(img_arr):
    if idx <= 10:
```

```
    plt.figure(figsize=(5,5))
    plt.imshow(img)
    plt.title(img.shape)
    plt.axis = False
    plt.show()  plotImages(t_img, label)
```

Model Checkpoint

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
mc = ModelCheckpoint(filepath="./best_model.h5",monitor = "accuracy",
                                    verbose = 1,save_best_only=True)
es = EarlyStopping(monitor="accuracy",min_delta=0.01,patience=5,verbose=1)
cb = [mc,es]

his = model.fit_generator(train_data, steps_per_epoch=10, epochs=30,callbacks=cb)
```

Loading the model and showing the accuracy graph

```
from keras.models import load_model
model = load_model("/content/Real-ESRGAN/best_model.h5")
h= his.history
h.keys()
plt.plot(h['loss'],'go--')
plt.plot(h['accuracy'],'go--', c= "red")
plt.title("Loss vs Acc")
plt.show()
```

Testing the Model

```
from google.colab import files
uploaded = files.upload()
path = next(iter(uploaded))
img = cv2.imread(path)
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
img = load_img(path, target_size=(256,256))
i = img_to_array(img)
i = preprocess_input(i)
input_arr = np.array([i])
input_arr.shape
pred = np.argmax(model.predict(input_arr))
if pred == 0:
  print("The image is oil")
else:
  print("The image is water")
plt.imshow(rgb_img)
plt.title("input image")
plt.axis = False
plt.show()
```

CONFUSION MATRIX

```
from sklearn import metrics
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

expected = ['oil','oil','water','oil','water','water','oil','water','water','water']
predicted = ['oil','oil','oil','water','water','water','oil','oil','oil','oil']
confusion_matrix = metrics.confusion_matrix(expected, predicted)
cf =metrics.confusion_matrix(expected,predicted)
cf

exp_series = pd.Series(expected)
pred_series = pd.Series(predicted)
pd.crosstab(exp_series, pred_series, rownames=['Actual'],
colnames=['Predicted'],margins=True)
plt.matshow(cf)
plt.title('Confusion Matrix Plot')
plt.colorbar()
```

```
plt.xlabel('Precited')
plt.ylabel('Actual')
plt.show();
metrics.confusion_matrix(predicted,expected)
plt.imshow(cf,cmap=plt.cm.Blues,interpolation='nearest')
plt.colorbar()
plt.title('Confusion Matrix without Normalization')
plt.xlabel('Predicted')
plt.ylabel('Actual')
tick_marks = np.arange(len(set(expected))) # length of classes
class_labels = ['0','1']
tick_marks
plt.xticks(tick_marks,class_labels)
plt.yticks(tick_marks,class_labels)
# plotting text value inside cells
thresh = cf.max() / 2.
for i in range(cf.shape[0]):
  for j in range(cf.shape[1]):
    plt.text(j,i,format(cf[i,j],'d'),horizontalalignment='center',color='white' if cf[i,j] >thresh
else 'black')
plt.show();
```

Accuracy

```
import numpy
import sklearn.metrics
y_true = ['oil','oil','water','oil','water','water','oil','water','water','water']
y_pred = ['oil','oil','oil','water','water','water','oil','oil','oil','oil']
r = sklearn.metrics.confusion_matrix(y_true, y_pred)
r = numpy.flip(r)
acc = (r[0][0] + r[-1][-1]) / numpy.sum(r)
print(acc)
acc = sklearn.metrics.accuracy_score(y_true, y_pred)
```

Precision

```
import sklearn.metrics
y_true = ['oil','oil','water','oil','water','water','oil','water','water','water']
y_pred = ['oil','oil','oil','water','water','water','oil','oil','oil','oil']
precision = sklearn.metrics.precision_score(y_true, y_pred, pos_label="oil")
print(precision)
```

Recall

```
import sklearn.metrics
y_true = ['oil','oil','water','oil','water','water','oil','water','water','water']
y_pred = ['oil','oil','oil','water','water','water','oil','oil','oil','oil']
recall = sklearn.metrics.recall_score(y_true, y_pred, pos_label="oil")
print(recall)
```

CHAPTER 7

CONCLUSION

In this project deep convolutional generative adversarial network (DCGAN) is used to generate the images in order to increase the datasets. The images generated by DCGAN is of low resolution which is not up to the expected level. The DCGAN generated images are mixed up with noise, so it has to be fine-tuned in such a way that the generated images are in good quality. To improve the resolution, SRGAN (Super Resolution Generative Adversarial Networks) can be integrated with DCGAN which may generate good resolution images. The main challenge here is the minimal amount of datasets available, to balance this , it is necessary to generate datasets on our own using GAN. if more than 10000 images are needed for training it will be difficult and time-consuming task to collect. Computational power plays a major role in training any GAN model which requires either GPU or TPU's to run it successfully.

In this paper 2D-CNN concentrates on categorization of two classes, they are oil and water. It is possible to add several classes in order to increase the accuracy of the model. For example classes like iceberg, dead plants, animals, wrecked ship can be added. By adding more classes, oil spill can be identified with the higher accuracy rate and the model won't get confused with the any other images and the models learn from these images whether it is oil spill or any other classes. We can derive the formula for oil spread along the ocean when it is surrounded by the free boundary Fig 5.19 using integral calculus or any other mathematical methods or experimental methods that include reviewing and calculating the oil spread across the oceans or water bodies.

REFERENCES

1) Krestenitis, M.; Orfanidis, G.; Ioannidis, K.; Avgerinakis, K.; Vrochidis, S.; Kompatsiaris, I. Oil Spill Identification from Satellite Images Using Deep Neural Networks. Remote Sens. 2019, 11, 1762. https://doi.org/10.3390/rs11151762
2) Tysiąc, P.; Strelets, T.; Tuszyńska, W. The Application of Satellite Image Analysis in Oil Spill Detection. Appl. Sci. 2022, 12, 4016. https://doi.org/10.3390/app12084016
3) Zeng, K.; Wang, Y. A Deep Convolutional Neural Network for Oil Spill Detection from Spaceborne SAR Images. Remote Sens. 2020, 12, 1015. https://doi.org/10.3390/rs12061015
4) Liu, J.; Yang, Z.; Liu, Y.; Mu, C. Hyperspectral Remote Sensing Images Deep Feature Extraction Based on Mixed Feature and Convolutional Neural Networks. Remote Sens. 2021, 13, 2599. https://doi.org/10.3390/rs13132599
5) Chen, Q.; Fan, C.; Jin, W.; Zou, L.; Li, F.; Li, X.; Jiang, H.; Wu, M.; Liu, Y. EPGNet: Enhanced Point Cloud Generation for 3D Object Detection. Sensors 2020, 20, 6927. https://doi.org/10.3390/s20236927
6) Ruizhongtai Qi, Charles & Su, Hao & Niessner, Matthias & Dai, Angela & Yan, Mengyuan & Guibas, Leonidas. (2016). Volumetric and Multi-View CNNs for Object Classification on 3D Data.
7) Li, Y.; Lyu, X.; Frery, A.C.; Ren, P. Oil Spill Detection with Multiscale Conditional Adversarial Networks with Small-Data Training. Remote Sens. 2021, 13, 2378. https://doi.org/10.3390/rs13122378
8) Tahir, A.; Munawar, H.S.; Akram, J.; Adil, M.; Ali, S.; Kouzani, A.Z.; Mahmud, M.A.P. Automatic Target Detection from Satellite Imagery Using Machine Learning. Sensors 2022, 22, 1147. https://doi.org/10.3390/s22031147
9) Huang X, Zhang B, Perrie W, Lu Y, Wang C. A novel deep learning method for marine oil spill detection from satellite synthetic aperture radar imagery. Mar Pollut Bull. 2022 Jun;179:113666. doi: 10.1016/j.marpolbul.2022.113666. Epub 2022 Apr 29. PMID: 35500373.
10) De Kerf, T.; Gladines, J.; Sels, S.; Vanlanduit, S. Oil Spill Detection Using Machine Learning and Infrared Images. Remote Sens. 2020, 12, 4090. https://doi.org/10.3390/rs12244090

11) Shaban, M.; Salim, R.; Abu Khalifeh, H.; Khelifi, A.; Shalaby, A.; El-Mashad, S.; Mahmoud, A.; Ghazal, M.; El-Baz, A. A Deep-Learning Framework for the Detection of Oil Spills from SAR Data. Sensors 2021, 21, 2351. https://doi.org/10.3390/s21072351
12) Yang, Yi-Jie & Singha, Suman & Mayerle, Roberto. (2022). A deep learning based oil spill detector using Sentinel-1 SAR imagery. International Journal of Remote Sensing. 43. 4287-4314. 10.1080/01431161.2022.2109445.
13) A. A. Huby, R. Sagban and R. Alubady, "Oil Spill Detection based on Machine Learning and Deep Learning: A Review," 2022 5th International Conference on Engineering Technology and its Applications (IICETA), Al-Najaf, Iraq, 2022, pp. 85-90, doi: 10.1109/IICETA54559.2022.9888651.
14) Basit, A.; Siddique, M.A.; Bhatti, M.K.; Sarfraz, M.S. Comparison of CNNs and Vision Transformers-Based Hybrid Models Using Gradient Profile Loss for Classification of Oil Spills in SAR Images. Remote Sens. 2022, 14, 2085. https://doi.org/10.3390/rs14092085
15) Konik, Marta & Bradtke, Katarzyna. (2016). Object-oriented approach to oil spill detection using ENVISAT ASAR images. ISPRS Journal of Photogrammetry and Remote Sensing. 118. 37-52. 10.1016/j.isprsjprs.2016.04.006.
16) Y. -J. Yang, S. Singha and R. Mayerle, "Fully Automated Sar Based Oil Spill Detection Using Yolov4," 2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS, Brussels, Belgium, 2021, pp. 5303-5306, doi: 10.1109/IGARSS47720.2021.9553030.
17) Al-Ruzouq, R.; Gibril, M.B.A.; Shanableh, A.; Kais, A.; Hamed, O.; Al-Mansoori, S.; Khalil, M.A. Sensors, Features, and Machine Learning for Oil Spill Detection and Monitoring: A Review. Remote Sens. 2020, 12, 3338. https://doi.org/10.3390/rs12203338
18) Y. Li, J. Yang, Z. Yuan and Y. Zhang, "Marine Oil Spills Detection and Classification from Polsar Images Based on Complex-Valued Convolutional Neural Network," IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, Kuala Lumpur, Malaysia, 2022, pp. 7085-7088, doi: 10.1109/IGARSS46834.2022.9883991.