

SHIP DETECTION ON HIGH RESOLUTION REMOTE SENSING IMAGE USING CNN

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

K RAJASHEKAR REDDY (Roll. No. 17SCS5080)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600 119**

MARCH - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **KAKANURU RAJASHEKAR REDDY (Roll. No. 17SCS5080)** who carried out the project entitled **"SHIP DETECTION ON HIGH RESOLUTION REMOTE SENSING IMAGE USING CNN"** under my supervision from **August 2020 to March 2021**.

Internal Guide

D.R. JOSHILA GRACE.,M.E.,Ph.d.,

Head of the Department

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

Dr. L. Lakshmanan, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **KAKANURU RAJASHEKAR REDDY** here by declare that the Project Report entitled “**SHIP DETECTION ON HIGH RESOLUTION REMOTE SENSING IMAGE USING CNN**” is done by me under the guidance of **DR. JOSHILA GRACE, M.E.,Ph.d.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D.,** and **Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **DR. Joshila Grace, M.E., Ph.d., Assistant Professor**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Deep convolutional neural network (DCNN) can achieve ship detection mission on the high-resolution remote sensing images. However, the false alarms caused by the onshore ship-like objects may decrease the accuracy and feasibility of these DCNN-based detection frameworks. In our work, an end-to-end method, named as Scene Mask R-CNN, is proposed to reduce the onshore false alarms. The scene mask extraction network (SMEN), as a network branch for scene segmentation, is innovatively introduced into the detection framework. The non-target area is marked out by an inferred scene mask which is used to assist the ship detection. Combining the feature map originated from feature extraction network (FEN) with the inferred scene mask by using the edge probability weighted (EPW) merging method, the false candidate targets in the non-target area are excluded. This novel mechanism of DCNN-based ship detection not only maintains the detection accuracy, but also effectively suppresses the false alarms in the non target area. Finally, the validity and accuracy of this method are verified on a ship dataset generated by the high-resolution optical remote sensing images.

TABLE OF CONTENTS

	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	viii
CHAPTER No.	TITLE	PAGE No.
1.	INTRODUCTION	
	1.1 About the project	1
2.	SYSTEM ANALYSIS	
	2.1 Existing system	2
	2.2 Proposed system	2
3.	REQUIREMENTS SPECIFICATION	
	3.1 Introduction	3
	3.2 Hardware and Software specification	4
	3.3 Technologies Used	4
4.	DESIGN AND IMPLEMENTATION CONSTRAINTS	
	4.1 Constraints in Analysis	27
	4.2 Architecture Diagram	28
	4.3 Sequence Diagram	30
	4.4 Use Case Diagram	31
	4.5 Activity Diagram	32
	4.6 Collaboration Diagram	33
5.	MODULES	
	5.1 Module Description	34
	5.2 Coding	35
	5.3 Test Procedure	37
	5.4 Test Data And Output	38
	SCREEN SHOTS	45
	SOURCE CODE	47
	REFERENCES	58

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
4.2	Architecture Diagram	29
4.3	Sequence Diagram	30
4.4	Use Case Diagram	31
4.5	Activity Diagram	32
4.6	Collaboration Diagram	33
B.1	Container Ship Detection	45
B.2	Cruise Ship Detection	45
B.3	Accommodation Ship Detection	46
B.4	War Ship Detection	46

LIST OF ABBREVIATIONS

DEX	Dalvik Executables
TCP	Transmission Control Protocol
IP	Internet Protocol
HTTP	Hyper Text Transfer Protocol

CHAPTER 1

INTRODUCTION

Aim:

To automate the detection of presence of ships and to classify the types of ships available in the given Image

1.1 Synopsis:

The detection of inshore and offshore ships is an essential task for a large variety of applications in both military and civilian fields. For example, in the civil field, ship detection plays a strong supervisory role in monitoring and managing marine traffics, transportation, fisheries dumping of pollutants and illegal smuggling. The dataset of sea ships images where trained and used to detect the presence of the ships in a given image.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Computer-aided ship detection methods greatly free up human resources and typically include two steps: extracting image features, and then using classifiers for classification and localization. These methods can produce stable results under calm sea conditions. However, when disturbances such as waves, clouds, rain, fog, and reflections happen, the extracted low-level features are not robust. Besides, manual selection of features is time-consuming and strongly dependent on the expertise and characteristics of the data itself.

Problem Statement:

- Getting images of ships.
- Machine learning, testing and training of images, and prediction

2.2 PROPOSED SYSTEM

A system is proposed to automate the detection of presence of ships in the given image using Machine Learning and Deep Learning Algorithms. We are proposing along with ship detection, a ship classification based on the type and category of the ships. The proposed system will not only detect a ship but also categorize as war ship, container ship etc.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

3.1 INTRODUCTION

The mission of ship detection, for searching a target of interest and obtaining its spatial location automatically, is one of the important applications of high-resolution remote sensing (RS) images. In recent researches, the ship detection method based on deep learning provides a feasible solution with high efficiency and intelligence. This innovative framework combines the ship detection method with deep convolutional neural network (DCNN) [1], which conducts a new detection paradigm based on supervised learning. Mass target samples are used to train DCNN and make it have a strong generalization ability. Considering the synchronization of classification and regression processes, DCNN-based target detection has two implementation schemes. One is the two-stage method represented by Faster R-CNN [2], and the other is the one-stage method such as SSD [3] and YOLO [4]. Recently, the two-stage DCNN-based ship detection methods [5][8] are widely concerned. They generally outperform the one-stage ship detection methods [9] in detection accuracy and algorithm scalability. Li *et al.* [10] proposes hierarchical selective altering (HSF) layers to map features with different scales into the same dimensional space, and these layers are applied on Faster R-CNN to detect multi-scale targets effectively. Zhang *et al.* [11] focuses on the ships with multi-direction and designs a rotated region proposal network (RPN), which obtains multi-oriented proposals with ship orientated angle and extracts the discriminative features from the inclined candidate regions to classify the target. Although the above two-stage methods of DCNN-based ship detection have a good performance, the suppression of on shore false alarms is neglected for most nearshore detection tasks. More specifically, some onshore ship-like objects, such as dock, roof, and road, are interpreted as targets of interest with high probability, even if the trained network has a high

detection accuracy. Generally, as a network training trick, the way of additional negative samples may be adopted to achieve the feature-based suppression of onshore false alarms [12], [13]. Providing sufficient negative samples, the network deliberately learns these representative scenes which are difficult to determine, and some frequent objects which are likely to be misjudged as

certain targets, which improves the robustness of the trained network. Wu *et al.* [14] detects the ship head and extracts the surroundings of ship head as region proposals, meanwhile utilizing plenty of hard negative samples, even the onshore false alarms. However, these methods rely on negative samples heavily. Considering the coverage of the RS imageries and diversity of objects, it is difficult to gather all unpredictable negative samples for training.

3.2 HARDWARE AND SOFTWARE SPECIFICATION

3.2.1 HARDWARE REQUIREMENTS

- Hard disk : 500 GB and above.
- Processor : i3 and above.
- Ram : 4GB and above.
- Webcam

3.2.2 SOFTWARE REQUIREMENTS

- Operating System : Windows 7 and above (64-bit).
- Python : 3.6

3.3 TECHNOLOGIES USED

- Python
- Region Based Convolution Neural Network & CNN

Introduction to Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

- Python 2.0 was released in 2000, and the 2.x versions were the prevalent releases until December 2008. At that time, the development team made the decision to release version 3.0, which contained a few relatively small but significant changes that were not backward compatible with the 2.x versions. Python 2 and 3 are very similar, and some features of Python 3 have been backported to Python 2. But in general, they remain not quite compatible.
- Both Python 2 and 3 have continued to be maintained and developed, with periodic release updates for both. As of this writing, the most recent versions available are 2.7.15 and 3.6.5. However, an official End Of Life date of January 1, 2020 has been established for Python 2, after which time it will no longer be maintained.
- Python is still maintained by a core development team at the Institute, and Guido is still in charge, having been given the title of BDFL (Benevolent Dictator For Life) by the Python community. The name Python, by the way, derives not from the snake, but from the British comedy troupe Monty Python's Flying Circus, of which Guido was, and presumably still is, a fan. It is common to find references to Monty Python sketches and movies scattered throughout the Python documentation.
- It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Net beans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python is Interpreted

- Many languages are compiled, meaning the source code you create needs to be translated into machine code, the language of your computer's processor, before it can be run. Programs written in an interpreted language are passed straight to an interpreter that runs them directly.
- This makes for a quicker development cycle because you just type in your code and run it, without the intermediate compilation step.
- One potential downside to interpreted languages is execution speed. Programs that are compiled into the native language of the computer processor tend to run more quickly than interpreted programs. For some applications that are particularly computationally intensive, like graphics processing or intense number crunching, this can be limiting.
- In practice, however, for most programs, the difference in execution speed is measured in milliseconds, or seconds at most, and not appreciably noticeable to a human user. The expediency of coding in an interpreted language is typically worth it for most applications.
- For all its syntactical simplicity, Python supports most constructs that would be expected in a very high-level language, including complex dynamic data types, structured and functional programming, and object-oriented programming.
- Additionally, a very extensive library of classes and functions is available that provides capability well beyond what is built into the language, such as database manipulation or GUI programming.
- Python accomplishes what many programming languages don't: the language itself is simply designed, but it is very versatile in terms of what you can accomplish with it.

Regional CNN

In this chapter we will talk about the Region based methods for object detection. In particular, R-CNN (Regional CNN), the original application of CNNs to this problem, along with its descendants Fast R-CNN, and Faster R-CNN. In classification, there's generally an image with a single object as the focus and the task is to say what that image is. But when we look at the world

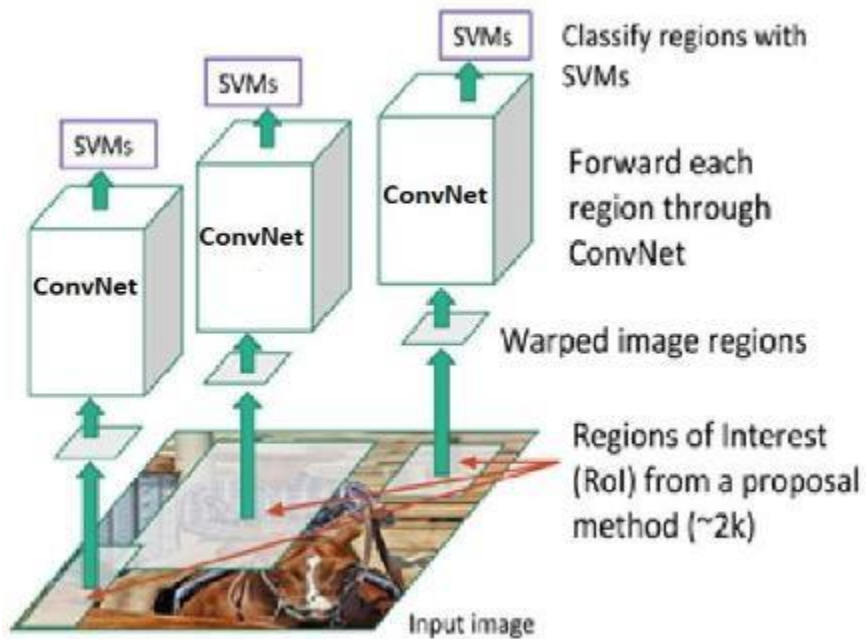
around us, we carry out far more complex tasks. We see complicated sights with multiple overlapping objects, and different backgrounds and we not only classify these different objects but also identify their boundaries, differences, and relations to one another. The goal of a Region based CNN is to take an image input to detect and localize the object in the image. All three above mentioned architectures were used for object detection and localization for a driving scenario to detect cars, pedestrians and traffic signs etc. and performed a time study to conclude which of the three architectures can be used as a detector for a real-world driving scenario for detection of objects present on road. We will also discuss their architecture and describe how they perform the task of object detection and localization given the input image.

R-CNN

R-CNN was first introduced by Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik in 2014. Region based Convolutional Network(R-CNN) combines two ideas: (1) one can apply high-capacity Convolutional Networks (CNN'S) to bottom-up region proposals in order to localize and segment objects and (2) When labelling data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific-fine-tuning, boosts performance significantly. They combined region proposals with CNNs and called the resulting model an R-CNN or Region-based Convolutional Network.

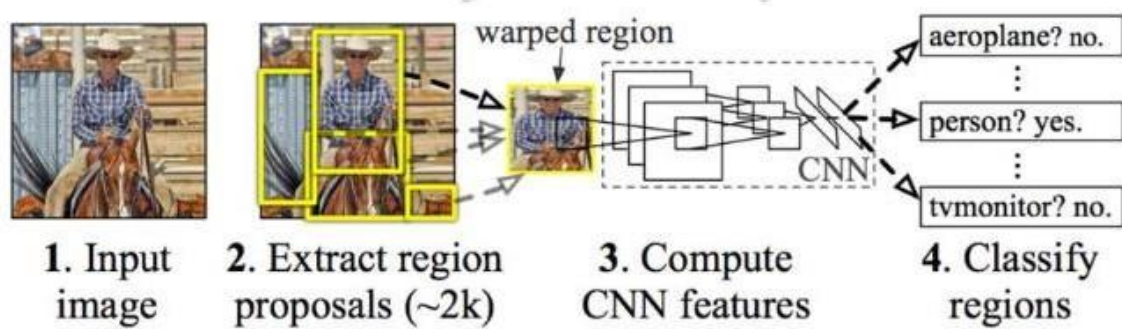
R-CNN Architecture:

In R-CNN we input an image and then using the basic feature detection techniques such as edge detection etc. we get Region Proposals Also referred as Regions of interest. This process is also known as selective search.



R-CNN Architecture

In the next step after these region proposals are extracted these wrapped image regions go through a trained CNN in our implementation Alex Net and then on the final layer the classification is done Using a Support Vector Machine(SVM) classifier which classifies whether this is an object and if so what. Pictorial representation of above process is shown in Figure.



Procedure for Object Detection and Localization in R-CNN

R-CNN works really well but is really quite slow for implementation in real time object detection.

The reasons being

1. It requires a forward pass of the CNN (Alex Net) for every single region proposal for every single image that's around 2000 forward passes per image.
2. It must train three different models separately -the CNN to generate image features, the classifier that predicts the class and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train

Convolutional Neural Networks (CNN)

Introduction

Convolutional neural networks (CNN) sound like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex

Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let's take a look at how to use these CNNs for image classification.

The Problem Space

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc.) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.



What We See

```

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 94 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 46 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 29 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 05 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

```

What Computers See

Inputs and Outputs

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describe the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc.).

What We Want the Computer to do

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat.

This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able to perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let's get into the specifics.

Biological Connection

But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 (Video) where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs.

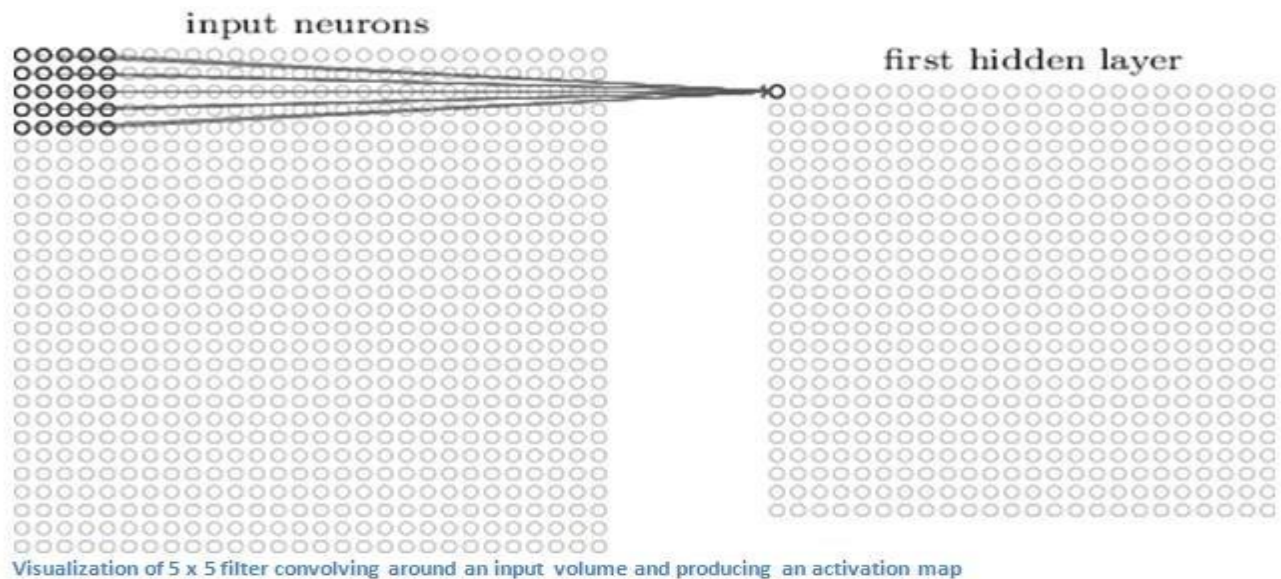
Structure

Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part understands what each of these layers do. So, let's get into the most important one.

First Layer – Math Part

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a $32 \times 32 \times 3$ array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5×5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter (or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimension of this filter is $5 \times 5 \times 3$. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would

be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a $28 \times 28 \times 1$ array of numbers, which we call an activation map or feature map. The reason you get a 28×28 array is that there are 784 different locations that a 5×5 filter can fit on a 32×32 input image. These 784 numbers are mapped to a 28×28 array.



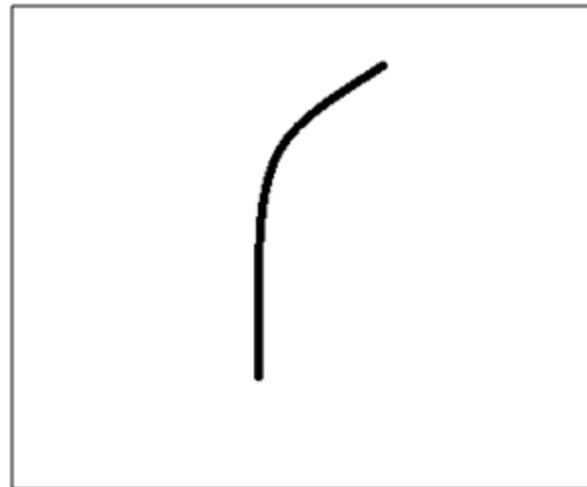
Let's say now we use two $5 \times 5 \times 3$ filters instead of one. Then our output volume would be $28 \times 28 \times 2$. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

First Layer – High Level Perspective

However, let's talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as **feature identifiers**. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is $7 \times 7 \times 3$ and is going to be a curve detector. (In this section, let's ignore the fact that the filter is 3 units deep and only consider the top depth slice of the filter and the image, for simplicity.) As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve (Remember, these filters that we're talking about as just numbers!).

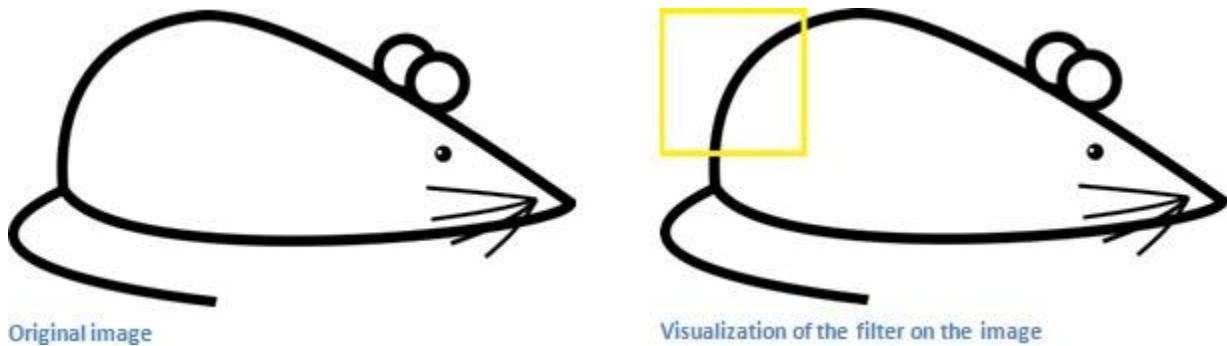
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner.



Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

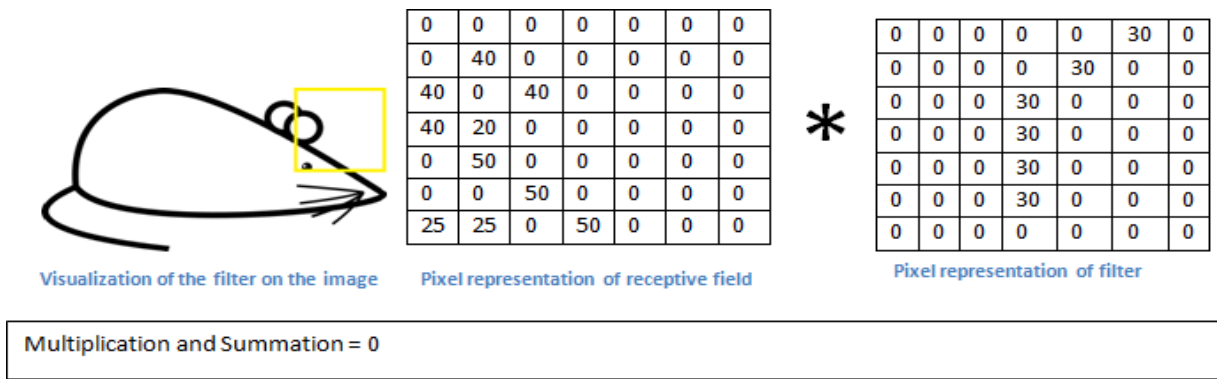
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$ (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.

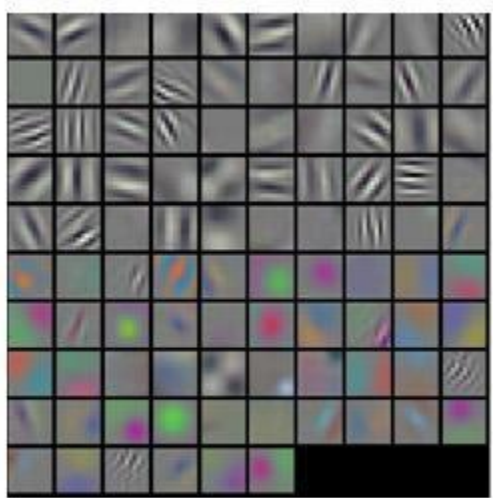


The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there are mostly likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in our activation map will be 0 because there wasn't anything in the input volume that caused the filter to activate (or more simply said, there wasn't a curve in that region of the original image). Remember, this is just for one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

Disclaimer:

The filter I described in this section was simplistic for the main purpose of describing the math that goes on during a convolution. In the picture below, you'll see some examples of actual

visualizations of the filters of the first conv layer of a trained network. Nonetheless, the main argument remains the same. The filters on the first layer convolve around the input image and “activate” (or compute high values) when the specific feature it is looking for is in the input volume.



Visualizations of filters

Going Deeper Through the Network

Now in traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I’d strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting. A classic CNN architecture would look like this.

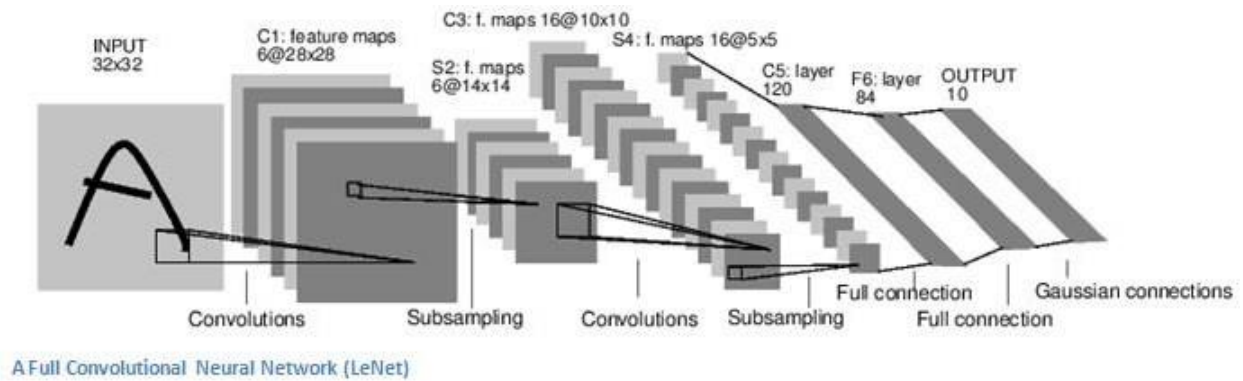
Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected

The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order predicting whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So, let's think about what the output of the network is after the first conv layer. It would be a $28 \times 28 \times 3$ volume (assuming we use three $5 \times 5 \times 3$ filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. If you want more information about visualizing filters in ConvNets, Matt Zeiler and Rob Fergus had an excellent [research paper](#) discussing the topic. Jason Yosinski also has a [video](#) on YouTube that provides a great visual representation. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger

area of the original input volume (another way of putting it is that they are more responsive to a larger region of pixel space).

Fully Connected Layer

Now that we can detect these high level features, the icing on the cake is attaching a fully connected layer to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the softmax approach). The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class. For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



Training (AKA: What Makes this Stuff Work)

Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at? How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called **Backpropagation**.

Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. The filters in the higher layers don't know to look for paws and beaks. As we grew older however, our parents and teachers showed us different pictures and images and gave us a corresponding label. This idea of being given an image and a label is the training process that CNNs go through. Before

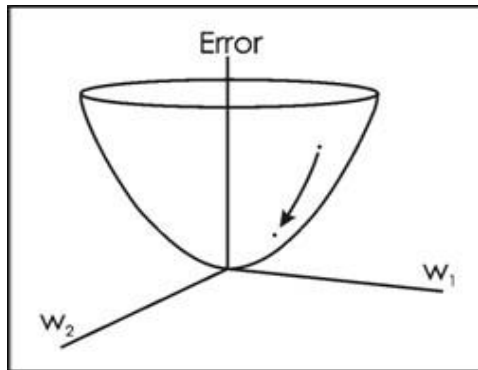
getting too into it, let's just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is. Back to backprop.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the forward pass, you take a training image which as we remember is a 32 x 32 x 3 array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1], basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the loss function part of backpropagation. Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is $\frac{1}{2}$ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right). In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in

calculus, we want to find out which inputs (weights in our case) most directly contributed to the loss (or error) of the network.



One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

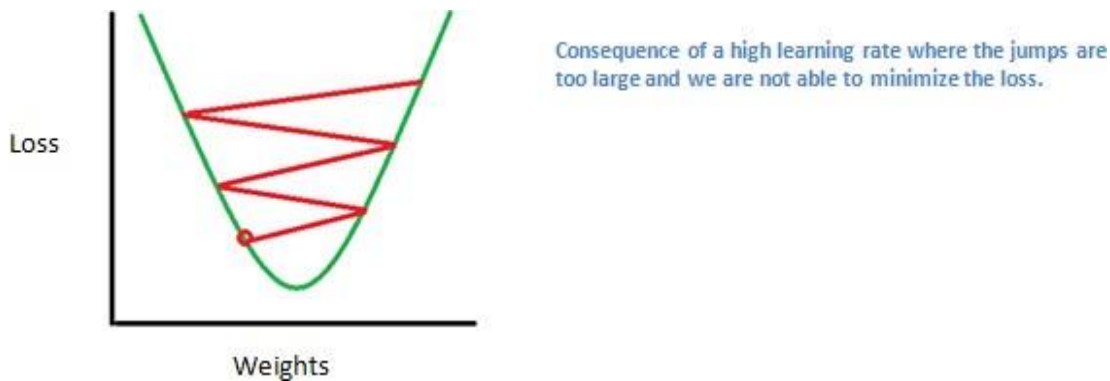
This is the mathematical equivalent of a dL/dW where W are the weights at a particular layer. Now, what we want to do is perform a backward pass through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the weight update. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
 w_i = Initial Weight
 η = Learning Rate

The learning rate is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the

model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

Testing

Finally, to see whether or not our CNN works, we have a different set of images and labels (can't double dip between training and test!) and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works!

How Companies Use CNNs

Data, data, data. The companies that have lots of this magic 4 letter word are the ones that have an inherent advantage over the rest of the competition. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.

Disclaimer

While this post should be a good start to understanding CNNs, it is by no means a comprehensive overview. Things not discussed in this post include the nonlinear and pooling layers as well as hyperparameters of the network such as filter sizes, stride, and padding. Topics like network architecture, batch normalization, vanishing gradients, dropout, initialization techniques, non-convex optimization, biases, and choices of loss functions, data augmentation, regularization methods, computational considerations, modifications of backpropagation, and more were also not discussed (yet).

CHAPTER 4

Design and Implementation Constraints

4.1 Constraints in Analysis

- ◆ Constraints as Informal Text
- ◆ Constraints as Operational Restrictions
- ◆ Constraints Integrated in Existing Model Concepts
- ◆ Constraints as a Separate Concept
- ◆ Constraints Implied by the Model Structure

Constraints in Design

- ◆ Determination of the Involved Classes
- ◆ Determination of the Involved Objects
- ◆ Determination of the Involved Actions
- ◆ Determination of the Require Clauses
- ◆ Global actions and Constraint Realization

Constraints in Implementation

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods.

Other Nonfunctional Requirements

4.1.1 Performance Requirements

The application at this side controls and communicates with the following three main general components.

- embedded browser in charge of the navigation and accessing to the web service;
- Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

4.1.2 Safety Requirements

1. The software may be safety-critical. If so, there are issues associated with its integrity level
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.
6. Systems with different requirements for safety levels must be separated.
7. Otherwise, the highest level of integrity required must be applied to all systems in the same environment.

4.2 Architecture Diagram:

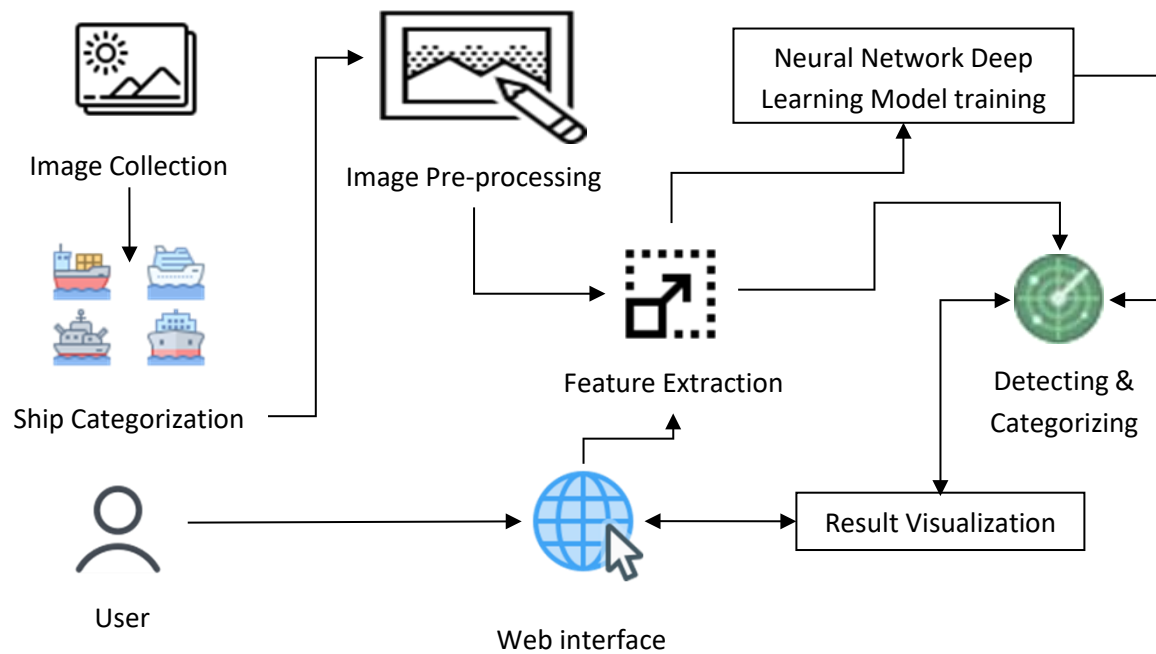
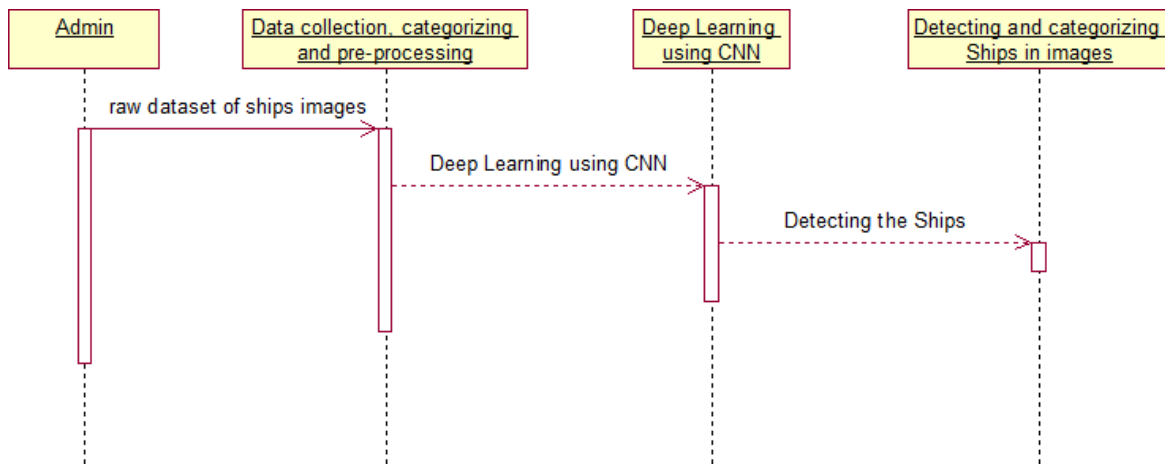


Fig: 4.2

4.3 Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.



Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

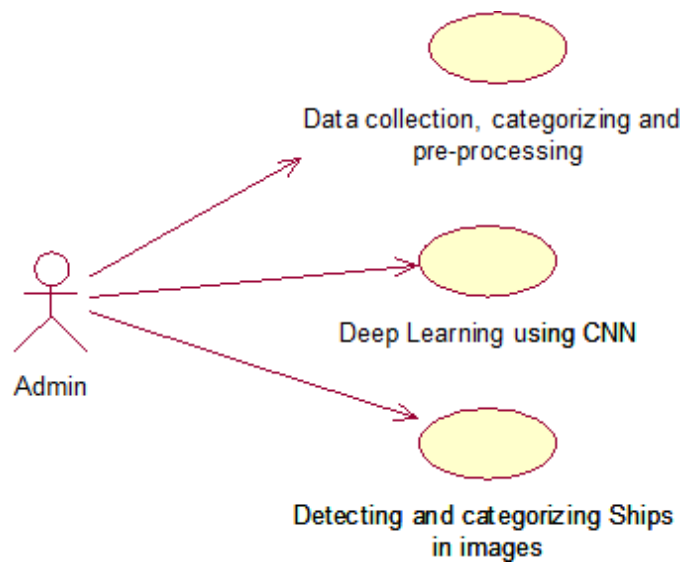
4.4 USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.

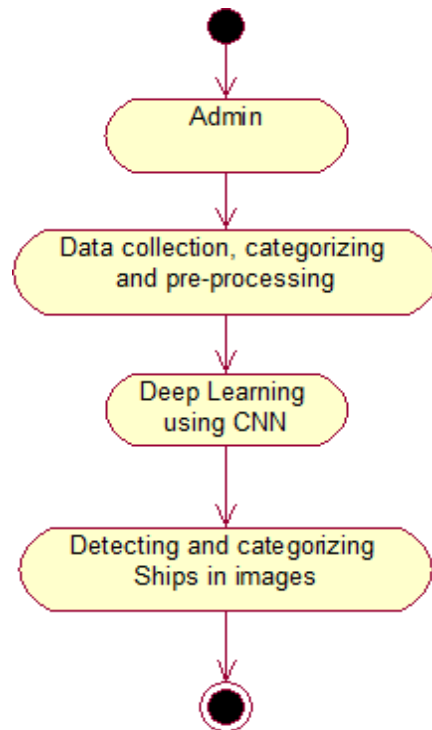


4.5 Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

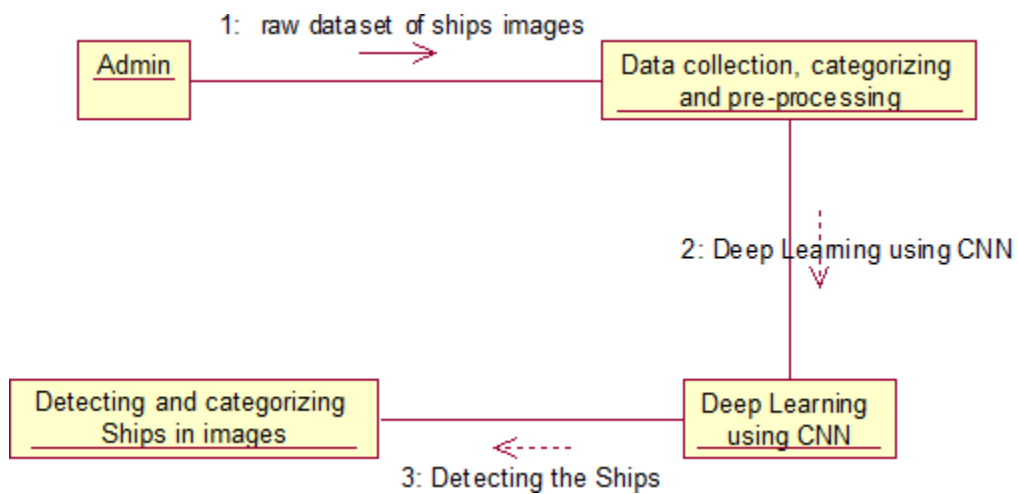
The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.



4.6 Collaboration Diagram:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.



CHAPTER 5

MODULES

- ✓ **Data collection, categorizing and pre-processing**
- ✓ **Deep Learning using CNN**
- ✓ **Detecting and categorizing Ships in images.**

5.1 Module Description:

Data collection, categorizing and pre-processing

A raw dataset was collected with type of ships and images for each type of ships. Since we did not have more number of ships images classified based on the categories, we use python to web crawl images from internet. The web scraped images were stored in different folders. For this project we were using four categories. We have downloaded ship images for the following categories, Accommodation, Container, War and Cruise. Python with vast text processing and networking libraries is the best tool to write one-off web scrapper.

Deep Learning using CNN

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. Deep learning is a class of machine learning algorithms that use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. We use convolutional Neural Network in our project for training and detecting the ships in the given image. We use multilayer neural network and each layer output is given as input to the next layer.

Detecting and categorizing Ships in images

The CNN trained model is used to detect the availability and the category of the ship in the given image. The output of the training module is saved as an object file using python. When the user inputs the images to for detection, the prediction method in the trained model is used to detect the presence of the ship, and also the category of the ship. A new image file is created with the category written on top of the actual image.

5.2 CODING

Once the design aspect of the system is finalized the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screwed into the system.

CODING STANDARDS

Coding standards are guidelines to programming that focuses on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standard needed to achieve the above-mentioned objectives are as follows:

Program should be simple, clear and easy to understand.

Naming conventions

Value conventions

Script and comment procedure

Message box format

Exception and error handling

5.2.1 NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be self-descriptive. One should even get the meaning and scope of the variable by its name. The conventions are adopted for easy understanding of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

5.2.2 VALUE CONVENTIONS

Value conventions ensure values for variable at any point of time. This involves the following:

- Proper default values for the variables.
- Proper validation of values in the field.
- Proper documentation of flag values.

5.2.3 SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

5.2.4 MESSAGE BOX FORMAT

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

- X – User has performed illegal operation.
- ! – Information to the user.

5.3 TEST PROCEDURE

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

5.4 TEST DATA AND OUTPUT

5.4.1 UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

5.4.2 FUNCTIONAL TESTS

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

5.4.3 PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

STRESS TEST

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

5.4.4 STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

- Exercise all logical decisions on their true or false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.
- Handling end of file condition, I/O errors, buffer problems and textual errors in output information

5.4.5 INTEGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

TESTING TECHNIQUES / TESTING STRATEGIES

TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered

error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclometric complexity
- Deriving test cases
- Graph matrices Control

BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods

- Equivalence partitioning
- Boundary value analysis
- Comparison testing

SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software and an independent test group conducts testing.
- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of individual who desires to penetrate system.

VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the

customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

- Input screen design.
- Output screen design.

SCREEN SHOTS



Fig: B.1 Container Ship Detection



Fig: B.2 Cruise Ship Detection



Fig: B.3 Accommodation Ship Detection



Fig: B.4 War Ship Detection

SOURCE CODE

Import Libraries

```
import random

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from PIL import Image

import json

#from mpl_toolkits.basemap import Basemap

import os

import itertools

from PIL import Image

import sklearn as sklearn

from sklearn.metrics import confusion_matrix


from keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

from sklearn import model_selection

from sklearn.model_selection import train_test_split, learning_curve


import warnings

warnings.filterwarnings('ignore')


from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier
```

```

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.svm import SVC, LinearSVC

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neural_network import MLPClassifier as MLPC

from sklearn.ensemble import GradientBoostingClassifier

import xgboost as xgb

from xgboost import XGBClassifier

from sklearn import model_selection


import tensorflow as tf

import keras

from keras import backend as K

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D

from keras import initializers, layers, models

from keras.utils import to_categorical

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

from keras import callbacks

from keras.utils.vis_utils import plot_model

%matplotlib inline

with open(r'D:\SHIP_DETECTION_SATELITE\shipsnet.json') as data_file:

    dataset = json.load(data_file)

Shipsnet= pd.DataFrame(dataset)

```

```

print(Shipsnet.head())

print("")

x = np.array(dataset['data']).astype('uint8')

y = np.array(dataset['labels']).astype('uint8')

def describeData(a,b):

    print('Total number of images: {}'.format(len(a)))

    print('Number of NoShip Images: {}'.format(np.sum(b==0)))

    print('Number of Ship Images: {}'.format(np.sum(b==1)))

    print('Percentage of positive images: {:.2f}%'.format(100*np.mean(b)))

    print('Image shape (Width, Height, Channels): {}'.format(a[0].shape))

describeData(x,y)

```

```

xReshaped = x.reshape([-1, 3, 80, 80]).transpose([0,2,3,1])
yReshaped = to_categorical(y, num_classes=2)
print("Data Shape",x.shape)
print('Labels Shape',y.shape)
print('Reshaped Data Shape',xReshaped.shape)
print('Reshaped Labels Shape',yReshaped.shape)

```

```

def describeDataset(features,labels):
    print("\n'X' shape: %s."%(features.shape,))
    print("\n'y' shape: %s."%(labels.shape,))
    print("\nUnique elements in y: %s"%(np.unique(y)))
describeDataset(xReshaped,yReshaped)

imgs0 = xReshaped[y==0]
imgs1 = xReshaped[y==1]

def plotOne(a,b):
    """
    Plot one numpy array
    """
    plt.subplot(1,2,1)
    plt.title('Not A Ship')
    plt.imshow(a[100])
    plt.subplot(1,2,2)
    plt.title('Ship')
    plt.imshow(b[100])
plotOne(imgs0, imgs1)

xReshaped = xReshaped/255
plotHistogram(xReshaped[100])

#from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2 ,random_state = 123)
# Reduce Sample Size for DeBugging
x_train = x_train[0:300000]
y_train = y_train[0:300000]
x_test = x_test[0:300000]

```

```
y_test = y_test[0:300000]
```

```
x_trainReshaped,x_testReshaped,y_trainReshaped,y_testReshaped =  
train_test_split(xReshaped,yReshaped,test_size = 0.2 ,random_state = 123)  
# Reduce Sample Size for DeBugging  
x_trainReshaped = x_trainReshaped[0:300000]  
y_trainReshaped = y_trainReshaped[0:300000]  
x_testReshaped = x_testReshaped[0:300000]  
y_testReshaped = y_testReshaped[0:300000]
```

```
def compareABunchOfDifferentModelsAccuracy(a,b,c,d):  
    print('\nCompare Multiple Classifiers:')  
    print('\nK-Fold Cross-Validation Accuracy:\n')  
    models = []  
    models.append(('LR', LogisticRegression()))  
    models.append(('RF', RandomForestClassifier()))  
    models.append(('KNN', KNeighborsClassifier()))  
    #models.append(('SVM', SVC()))  
    #models.append(('LSVM', LinearSVC()))  
    #models.append(('GNB', GaussianNB()))  
    #models.append(('DTC', DecisionTreeClassifier()))  
    #models.append(('XGB', XGBClassifier()))  
    # models.append(('GBC', GradientBoostingClassifier()))  
    # models.append(('LDA', LinearDiscriminantAnalysis()))  
    resultsAccuracy = []  
    names = []  
    for name, model in models:  
        model.fit(a,b)  
        kfold = model_selection.KFold(n_splits=10, random_state=7)  
        accuracy_results = model_selection.cross_val_score(model, a,b, cv=kfold, scoring='accuracy')
```

```

    resultsAccuracy.append(accuracy_results)
    names.append(name)
    accuracyMessage = "%s: %f (%f)" % (name, accuracy_results.mean(), accuracy_results.std())
    print(accuracyMessage)

# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison: Accuracy')
ax = fig.add_subplot(111)
plt.boxplot(resultsAccuracy)
ax.set_xticklabels(names)
ax.set_ylabel('Cross-Validation: Accuracy Score')
plt.show()
return

compareABunchOfDifferentModelsAccuracy(x_train,y_train,x_test,y_test)

def defineModels():
    print('LR = LogisticRegression')
    print('RF = RandomForestClassifier')
    print('KNN = KNeighborsClassifier')
    print('SVM = Support Vector Machine SVC')
    print('LSVM = LinearSVC')
    print('GNB = GaussianNB')
    print('DTC = DecisionTreeClassifier')
    print('XGB = XGBClassifier')
    # print('GBC = GradientBoostingClassifier')
    # print('LDA = LinearDiscriminantAnalysis')
    return

defineModels()# Plot confusion matrix
def plot_confusion_matrix(cm, classes,

```

```

        normalize=False,
        title='Confusion matrix',
        cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel("True label")
    plt.xlabel('Predicted label')

# Special callback to see learning curves
class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()

```

```

        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)
    def plotKerasLearningCurve():
        plt.figure(figsize=(10,5))
        metrics = np.load('logs.npy')[()]
        filt = ['acc'] # try to add 'loss' to see the loss learning curve
        for k in filter(lambda x : np.any([kk in x for kk in filt]), metrics.keys()):
            l = np.array(metrics[k])
            plt.plot(l, c='r' if 'val' not in k else 'b', label='val' if 'val' in k else 'train')
            x = np.argmin(l) if 'loss' in k else np.argmax(l)
            y = l[x]
            plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not in k else 'b')
            plt.text(x, y, '{ } = {:.4f}'.format(x,y), size='15', color= 'r' if 'val' not in k else 'b')
        plt.legend(loc=4)
        plt.axis([0, None, None, None]);
        plt.grid()
        plt.xlabel('Number of epochs')
    def plot_learning_curve(history):
        plt.figure(figsize=(8,8))
        plt.subplot(1,2,1)
        plt.plot(history.history['acc'])
        plt.plot(history.history['val_acc'])
        plt.title('model accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')

```

```

plt.savefig('./accuracy_curve.png')
#plt.clf()
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('./loss_curve.png')

dict_characters = {0: 'No Ship', 1: 'Ship'}

def runKerasCNNAugment(a,b,c,d):
    batch_size = 128
    num_classes = 2
    epochs = 12
    #img_rows, img_cols = X_train.shape[1],X_train.shape[2]
    #input_shape = (img_rows, img_cols, 3)
    input_shape = (80,80,3)
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                    activation='relu',
                    input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))

```

```

model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # apply ZCA whitening
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
    horizontal_flip=True, # randomly flip images
    vertical_flip=False) # randomly flip images
history = model.fit_generator(datagen.flow(a,b, batch_size=32),
                             steps_per_epoch=len(a) / 32, epochs=epochs, validation_data = [c, d],callbacks =
[MetricsCheckpoint('logs')])
score = model.evaluate(c,d, verbose=0)
print("\nKeras CNN #1C - accuracy:', score[1],'\n')
y_pred = model.predict(c)
map_characters = {0: 'No Ship', 1: 'Ship'}
print("\n', sklearn.metrics.classification_report(np.where(d > 0)[1], np.argmax(y_pred, axis=1),
target_names=list(map_characters.values())), sep='')
score = model.evaluate(c,d, verbose=0)
Y_pred_classes = np.argmax(y_pred,axis = 1)
Y_true = np.argmax(d,axis = 1)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

```

```
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

runKerasCNNAugment(x_trainReshaped, y_trainReshaped, x_testReshaped, y_testReshaped)
```

REFERENCES

- [1] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527_1554, Jul. 2006.
- [2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real- time object detection with region proposal networks," *IEEE Trans. Pattern . Mach. Intell.*, vol. 39, no. 6, pp. 1137_1149, Jun. 2017.
- [3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, in Lecture Notes in Computer Science, vol. 9905. Amsterdam, The Netherlands: Springer, Oct. 2016, pp. 21_37.
- [4] Y.-L. Chang, A. Anagaw, L. Chang, Y. C. Wang, C.-Y. Hsiao, and W.-H. Lee, "Ship detection based on YOLOv2 for SAR imagery," *Remote Sens.*, vol. 11, no. 7, p. 789, Apr. 2019.
- [5] Z. Lin, K. Ji, X. Leng, and G. Kuang, "Squeeze and excitation rank faster R-CNN for ship detection in SAR images," *IEEE Geosci. Remote Sens. Lett.*, vol. 16, no. 5, pp. 751_755, May 2019.
- [6] C. Dong, J. Liu, and F. Xu, "Ship detection in optical remote sensing images based on saliency and a rotation-invariant descriptor," *Remote Sens.*, vol. 10, no. 3, p. 400, Mar. 2018.
- [7] T. Zhang and X. Zhang, "High-speed ship detection in SAR images based on a grid convolutional neural network," *Remote Sens.*, vol. 11, no. 10, p. 1206, May 2019.
- [8] X. Yang, J. Yang, J. Yan, Y. Zhang, T. Zhang, Z. Guo, S. Xian, and K. Fu, "SCRDet: Towards more robust detection for small, cluttered and rotated objects," Aug. 2019, *arXiv:1811.07126*. [Online]. Available: <https://arxiv.org/abs/1811.07126v4>

[9] L. Qu, S. Wang, N. Yang, L. Chen, L. Liu, X. Zhang, F. Gao, and J. Dong, "Improving object detection accuracy with region and regression based deep CNNs," in *Proc. Int. Conf. Secur., Pattern Anal., Cybern. (SPAC)*, Dec. 2017, pp. 318_323.

[10] Q. Li, L. Mou, Q. Liu, Y. Wang, and X. X. Zhu, "HSF-Net: Multiscale deep feature embedding for ship detection in optical remote sensing imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 12, pp. 7147_7161, Dec. 2018.