Project Report
On

SATELLITE IMAGERY RECOGNITION

Project II Submitted in partial fulfilment of the requirements for the degree of

B.Tech. in CSE (Artificial Intelligence and Machine Learning)

by

VIVEK THAKUR (13030820026)

HARSH RANJAN (13030820031)

DIVYA KUMAR BAID (13030820030)

SUMEDHA SARKAR (13030820032)

RITAM KONAR (13000320019)

Under the guidance of

Poojarini Mitra

Assistant Professor

CSE - Data Science

TECHNO MAIN SALT LAKE

EM 4/1 SALT LAKE CITY, SECTOR V

KOLKATA – 700091

West Bengal, India

TECHNO MAIN SALT LAKE

[Affiliated by Maulana Abul Kalam Azad University of Technology (Formerly known as WBUT)]

FACULTY OF CSE (AI & ML) DEPARTMENT

Certificate of Recommendation

This is to certify that **Vivek Thakur, Harsh Ranjan, Divya Kumar Baid, Sumedha Sarkar** and **Ritam Konar** have completed their project report on: **Satellite Imagery Recognition**, under the direct supervision and guidance of **Poojarini Mitra**. We are satisfied with their work, which is being presented for the partial fulfilment of the degree of B. Tech in CSE (Artificial Intelligence and Machine Learning), Maulana Abul Kalam Azad University of Technology) (Formerly known as WBUT), Kolkata – 700064.

Poojarini Mitra

(Signature of Project Supervisor)

Date: ---

Dr. Sudipta Chakrabarty

(Signature of Project Coordinator)

Date: ---

Dr. Soumik Das

(Signature of the HOD)

Date: ---

Acknowledgement

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to the teaching fraternity of the Department of CSE (Artificial Intelligence and Machine Learning), for giving us this opportunity to undertake this project and also supporting us whole heartedly.

We also wish to express our gratitude to the HOD and our project mentor Poojarini Mitra ma'am for her kind hearted support, guidance and utmost endeavor to groom and develop our academic skills. At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during the effort in shaping this concept till now.

Signature of the Candidates

Vivek Thakur (13030820026)

Harsh Ranjan (13030820031)

Divya Kumar Baid (13030820030)

Sumedha Sarkar (13030820032)

Ritam Konar (13000320019)

Abstract

Recent advancements in AI have opened doors to various military applications like surveillance, reconnaissance, and threat assessment. However, identifying military vehicles in aerial images remains a challenge due to scale, illumination, and orientation variations. Additionally, the lack of well-annotated datasets hampers the efficacy of deep learning algorithms in this domain. Despite the potential benefits of AI in military contexts, several challenges persist. These include the need for transparency in AI systems, ensuring robustness against data manipulation, and overcoming data scarcity in military settings. In satellite imagery analytics, detecting small objects poses a significant challenge, especially given the limited resolution of overhead imagery. To address this, a project focused on exploring instance segmentation for military asset detection using transfer learning, employing techniques such as MLP & CNN with data augmentation and dropouts, as well as basic R-CNN models and ConvNets. The experiments aimed to analyze the effectiveness of these approaches in distinguishing various types of military vehicles across different contexts, including airborne and seaborne scenarios, demonstrating promising results in recognizing military vehicles through deep architectures trained on customized datasets.

Contents

Scope of Project

Our Work on this project coincided with ongoing conflicts like Russo-Ukraine, Armenia-Azerbaijan, Israel-Palestine and Attacks of merchant vessels near Red Sea and Bab-Al Mandeb strait, we observed the critical use of satellite imagery by journalists, human rights groups, and open-source analysts. High-resolution, real-time satellite imagery aids in tracking troop movements, verifying incidents in remote areas, assessing infrastructure damage, and documenting potential war crimes.

Given the escalating reliance on satellite imagery in modern conflicts, our goal was to explore using deep learning to identify and classify military vehicles of variety. The sheer volume of satellite imagery being produced necessitates an automated solution for quickly analyzing images, especially those containing objects of interest.

Recent Events

The Russo-Ukrainian War is an ongoing war between Russia and Ukraine, which began in February 2014. Starlink terminals, which provide high-speed communications, have been vital in giving Ukraine's military an edge over invading Russian troops. Ukrainian intelligence said it has confirmed that Russian forces are using satellite internet service Starlink on the battlefield in occupied areas in the east of the country. Various satellite images have surfaced which give us an estimate and idea on both military capacity as well as the aftermaths of the destruction caused by the war conflict.

On September 19, 2023, Azerbaijan launched a large-scale military offensive on Nagorno-Karabakh. The offensive violated the 2020 ceasefire agreement and resulted in heavy fighting between Azerbaijani and Armenian forces. On September 20, 2023, a ceasefire agreement was reached at the mediation of the Russian peacekeeping command in Nagorno-Karabakh. European Space Imaging (EUSI) partner, Maxar has collected satellite images (September 26th) of the Nagorno-Karabakh (NKAO) region that reveal a long traffic jam of vehicles along the Lachin corridor as thousands of ethnic Armenians leave Stepanakert.

An armed conflict between Israel and Hamas-led Palestinian militant groups has been taking place chiefly in and around the Gaza Strip since 7 October 2023. Satellite imagery shows that the fighting has resulted in heavy damage to almost every corner of Gaza City, far beyond the port area.

The Indian Navy said it had freed a hijacked Iranian fishing vessel from nine armed pirates in the Arabian Sea on 29th March 2024, rescuing its crew unharmed. The fishing vessel, Al-Kambar 786, was southwest of the Yemeni island of Socotra on March 28 when it was reported to have been boarded by pirates, according to a statement from the navy late on Friday. Indian Navy rescued 23 Pakistani nationals from Somali pirates in a 12-hour operation in the Arabian Sea.

These are some of the events that has occurred recently including many others, and we can see in each case, somehow or the other, data and information collected through satellite images have held undeniable significance, and has aided different forces to have advantages such as tracking troop movement, verifying enemy strength and many more. So, importance of satellite imagery recognition and detection have a significant role to play in future in the defense field as well as other fields.

Continuous Project Proposal

The main goal of our project is satellite imagery recognition. In the previous project our group initially worked with the terrain based military vehicles, we used datasets that captured data via

Figure 1: Satellite images show new deployment in Belarus, over 100 military vehicles seen

Figure 2: Russian War Report: Russian false-flag operation seeks to drag Belarus into Ukraine war

SAR (synthetic aperture radar). We initially developed a simple ANN-MLP model for initial exploration and then we developed 3 further models, initially a simple CNN model, then adding more layers in the second comparatively complex CNN model to enhance the performance of the model, and then for the final model we introduced refinements like data augmentation and dropout layers. The final model gives the most optimal result. We made the comparative study of our model ref-

Figure 3: Russian air-force

erencing 2 different pre-existing models, namely Xception and VGG19, and our model shows the most effective performance amongst all these models.

Now for our current project we are trying to extend our domain of the project and work with the sea borne as well as air borne military vehicles referencing our previous project. Here we are developing new separate models for the sea borne and air borne vehicle detection according to the necessities and differences between the two domains and our goal is to achieve similar high performance in these two domains as well. So, our goal here is to further expand our field of study from terrain borne to sea borne and air borne military vehicle detection using satellite Imagery recognition.

Figure 4: INS Viraat near Alang Ship Breaking Yard

Figure 5: a seized oil tanker Niovi off the coast of Bandar Abbas, Iran, May 6, 2023

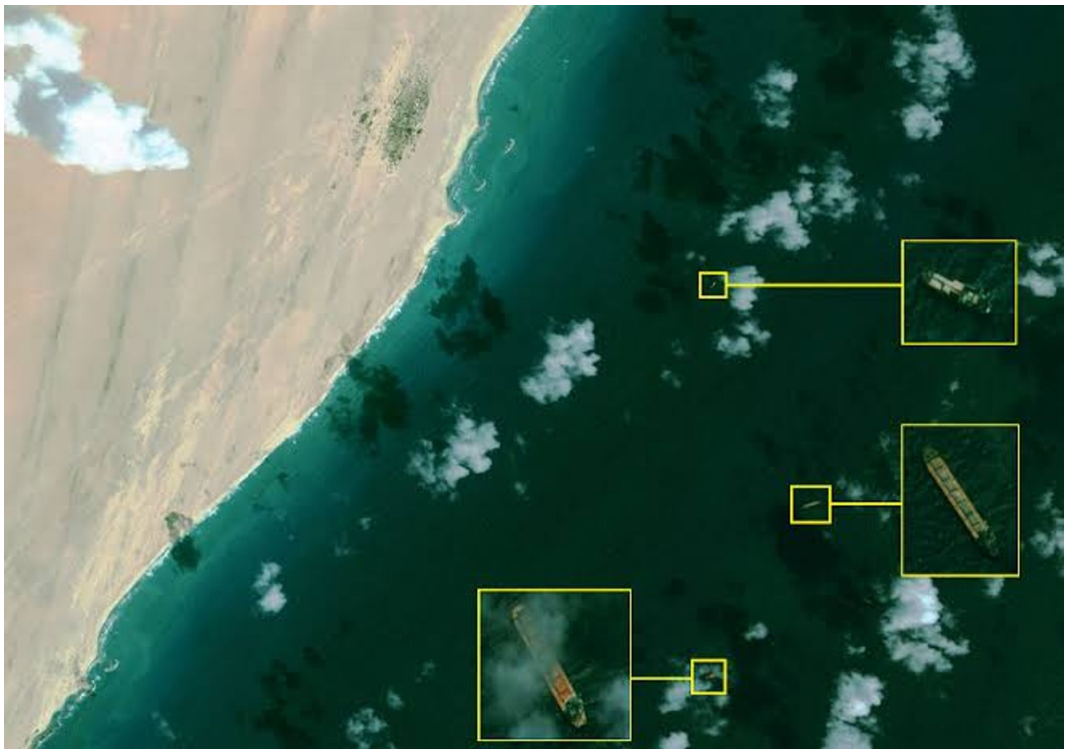


Figure 6: Warships near Iranian Shore

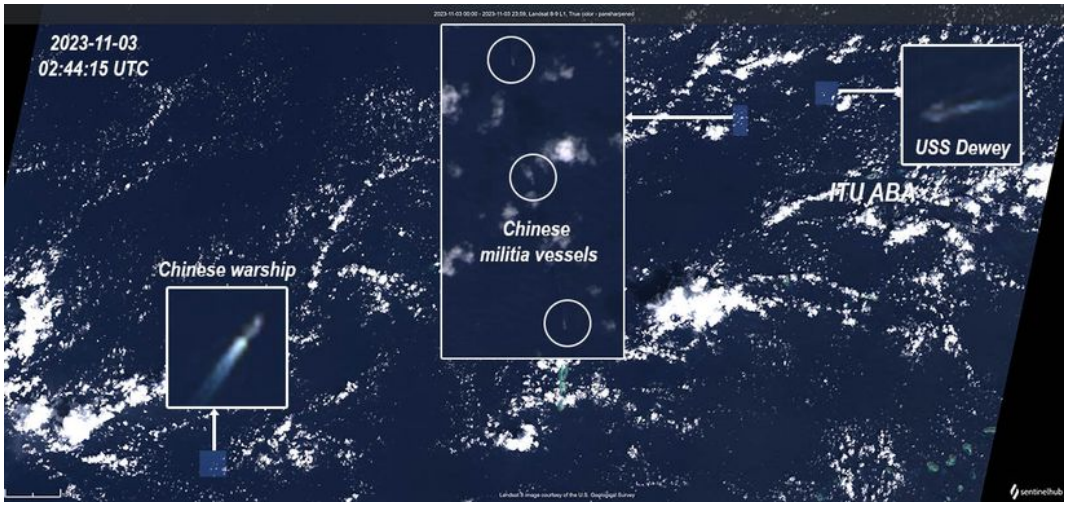


Figure 7: Image tracking Chinese warships and military vessels

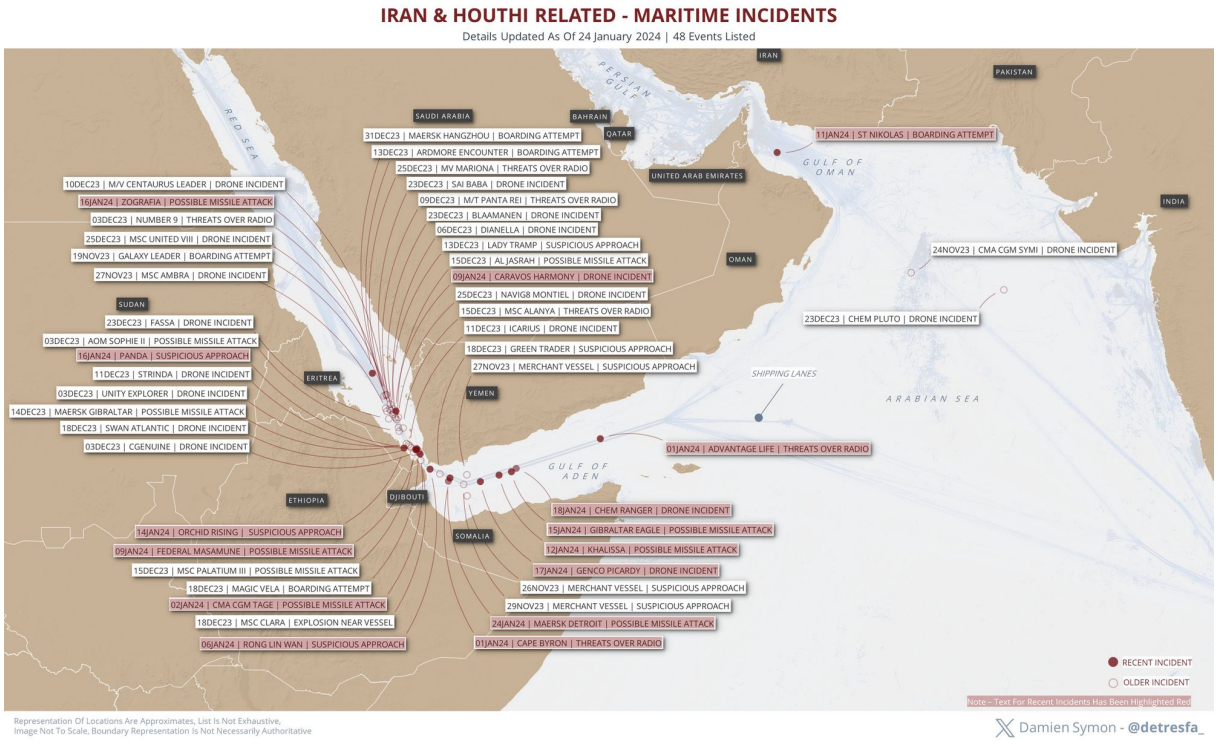


Figure 8: Iran and Houthi related Maritime Incidents

Chapter 1

Introduction

In recent times, deep neural network architectures, such as convolutional neural networks (CNNs), have demonstrated remarkable success in object recognition and detection. They have a potent discriminative feature extraction capabilities, enabling the replacement of manually crafted features with task-specific deep features. While there is limited literature on the detection of military vehicles in optical aerial imagery, some researches have introduced deep learning architectures for the detection of military vehicles in aerial imagery.

Detection methods based on R-CNN have exhibited strong performance in natural scene images . Substantial reductions in computational costs for training and testing can be achieved through the implementation of Fast R-CNN [1] and Faster R-CNN [2]. They achieved good results on common detection benchmark datasets. Notably, they employ a shared convolutional feature map for the entire image, as opposed to computing convolutional features separately.

Limited Spatial Extent: Satellite images often feature small, densely clustered objects of interest, in contrast to conventional **ImageNet data [1]**, which typically comprises larger subjects. Satellite image resolution, measured by the ground sample distance (GSD), representing the physical size of a single pixel in the image means even small objects like cars may span only about 15 pixels.

Complete rotation invariance: Objects viewed from overhead can have any orientation (e.g. ships can have any heading between 0 and 360 degrees, whereas trees in ImageNet data are reliably vertical).

Scarcity of Training Data: The availability of adequate training data is limited. While military organizations and satellite systems excel in data collection for debriefing, reconnaissance, or reconstruction purposes, the suitability of this data for machine learning is uncertain. This necessitates adapting data collection processes to fully harness modern AI techniques, as further discussed.

Furthermore, Computer vision techniques have made significant progress since the introduction of **CNNs [4]** in the **ImageNet [3]** competition. The presence of extensive, high quality labelled datasets such as **ImageNet [3]**, **PASCAL VOC [5]** and **MS COCO [6]** have facilitated remarkable advancements in rapid real-time object detection ; few of the best are: Faster **R-CNN**

[**2**], **SSD** [**7**], and **YOLO** [**8**].

1.1 Data Requirements for Developing AI Detectors — The Challenge of Geographic Data Diversity

To develop a deep learning algorithm specialized on geographic data, a large variability of data is required in terms of:

1. geographical zones : weather conditions (snow, clouds), urban or desertic areas, latitude and time of the day (shadows)
2. Sizes and shapes of the observables to detect different models of aircrafts or multi-terrain vehicles which can have different shapes, colors or positions but it will still belong to the same category and therefore, it will have to be classified in the very same class.

Figure 1.1: Complexity of geographical areas and observable with various forms — Source : 2022 Copyright Maxar Technologies

Without these tools and an enriched database, it would be very difficult, if not impossible, to source suitable images for algorithm development. To proceed effectively, we require: (1) The most precise geographic database available, (2) A diverse selection of readily accessible images: a minimum of 5,000 to 10,000, and (3) A detailed image description to be used as a train or test.

1.2 Problem Analysis

1.2.1 Sea-Borne

The dataset used for this part is derived from **Planet's Open California dataset**, which is openly licensed, used over Kaggle. This dataset is intended to facilitate the challenging task of identifying the presence and location of large ships in satellite images. Due to imbalance in dataset, up-sampling is done; for creating a model for classifying the ROIs into the 2 classes, *ship* and *no-ship*.

Creating an Ab-INITIO **R-CNN** model using TensorFlow; Initially, features are extracted from the images using a CNN model. For image detection in images using R-CNN the feature map produced by the convolutional layers is used, i.e., the model drops the layers after the final convolutional block and passes the generated feature map to a RPN.

1.2.2 Air-Borne

The dataset used for constructing this model is sourced from the **Fine-Grained Visual Classification of Aircraft (FGVC-Aircraft)**, a benchmark dataset for the fine grained visual categorization of aircraft, used over Kaggle. The task is to employ ConvNets with the Keras Tuner Hyperband tuner to create a sequential model for classifying military aircraft into twenty distinct types.

Aircraft model recognition is particularly fascinating due to the following aspects: (1) Spanning a century, it includes thousands of diverse models from various manufacturers and airlines. (2) Significant design variations are based on factors like size, purpose, propulsion, and technology, depending on private, civil, or military use. (3) Repurposing by different companies leads to appearance variations, which may be seen as noise or useful information. (4) Aircraft's rigidity simplifies modeling, focusing on core aspects of fine-grained recognition.

The Datasets, Models and their Evaluations suggest that a machine capable of programming itself has the potential to: (1) improve efficiency with respect the development costs of both software and hardware, (2) perform specific tasks at a superhuman level, (3) provide objective and fair decisions where humans are known for being subjective, biased, unfair, corrupt, etc.

Chapter 2

Literature Survey

Conventionally, the developed approaches aiming to solve the vehicle detection problem in aerial images focus on non-military vehicle types [3]. They typically rely on using a sliding window approach composed of hand-crafted feature extraction followed by a classifier or a cascade of classifiers. For instance, **Liu and Mattyus [9]** detected vehicles with two attributes (orientation and type) on aerial images using such a cascaded classifier. To localize the vehicles, it employs a fast binary detector in a soft-cascade structure whose output is fed as an input to a multiclass classifier for estimation of orientation and type of the vehicle.

Tuermer et al. [3] employed a series of processing steps to extract potential vehicular regions that are later classified using a histogram of oriented features. **Cheng et al. [4]** performed pixel-wise dynamic Bayesian network based classification to detect vehicles for an aerial surveillance application using color and edge features. **Shao et al. [5]** utilized an interactive bootstrapping approach with multiple image descriptors such as histogram of gradients, local binary pattern and opponent histogram to train an intersection kernel support vector machine. Non-maximum suppression is later used to eliminate false detected vehicles.

R-CNN, introduced by **Ross Girshick. [13]** 2014, addresses the challenge of efficient object localization in object detection. Unlike previous methods that rely on Exhaustive Search (sliding windows at various scales to propose region proposals), R-CNN employs the Selective Search algorithm, leveraging object segmentation and Exhaustive Search to efficiently propose regions.

In R-CNN, each region proposal is individually processed through the CNN architecture. However, this approach generates around 2000 region proposals per image, making training and testing computationally intensive. To mitigate this, **Ross Girshick. [1]**, proposed Fast R-CNN, which takes the entire image and region proposals as input in a single forward pass. This method combines ConvNets, RoI-pooling, and the classification layer into a unified architecture, streamlining the process.

Faster R-CNN was introduced in 2015 by **Ross Girshick. [2]**. After the Fast R-CNN, the bottleneck of the architecture is selective search. Since it needs to generate *2000* proposals per image. It constitutes a major part of the training time of the whole architecture. In Faster R-CNN, it was replaced by the region proposal network.

	R-CNN	Fast R-CNN	Faster R-CNN
region proposals method	Selective search	Selective search	Region proposal network
Prediction timing	40-50 sec	2 seconds	0.2 seconds
computation	High computation time	High computation time	Low computation time
The mAP on Pascal VOC 2007 test dataset(%)	58.5	66.9 (when trained with VOC 2007 only) 70.0 (when trained with VOC 2007 and 2012 both)	69.9(when trained with VOC 2007 only)
The mAP on Pascal VOC 2012 test dataset (%)	53.3	65.7 (when trained with VOC 2012 only) 68.4 (when trained with VOC 2007 and 2012 both)	67.0(when trained with VOC 2012 only) 70.4 (when trained with VOC 2007 and 2012 both) 75.9(when trained with VOC 2007 and 2012 and COCO)

Figure 2.1: The table is in reference to the workings of R-CNN, fast R-CNN & faster R-CNN

Fine-Grained Visual Classification of Aircraft, Subhransu Maji el at. [14] This article introduces the FGVC-Aircraft dataset, which includes 10,000 images of 100 distinct aircraft models, organized hierarchically. Despite subtle visual differences, they are distinguishable, making for a challenging yet manageable task in visual recognition. The paper outlines classification tasks, evaluation protocols, and baseline results, a benchmark for the dataset. The paper presents classification tasks, evaluation methods, and baseline results, creating a dataset benchmark. In contrast to typical fine-grained visual classification, like animals, aircraft exhibit rigidity but offer variations in purpose, size, designation, structure, historical style, and branding.

Li, Lisha, and Kevin Jamieson. 2018 "Hyperband: A Novel Bandit-Based Approach

to Hyperparameter Optimization." [15]. The Keras Tuner Hyperband tuner is an advanced optimization tool used for hyperparameter tuning in machine learning models. It efficiently explores the hyperparameter space by employing the Hyperband algorithm, balancing exploration and exploitation for optimal results. This tuner is particularly effective for tuning deep learning models built with the Keras framework, allowing researchers and practitioners to find the best hyperparameter configurations for their specific tasks.

S. Richard F. Sims el at. "Efficient pattern recognition and classification" [16], formulated from the training images of all target classes such that the size of the filter is the same as the expected targets. For real time applications, the input scene is first correlated with all maximum average correlation height (MACH) filters and the correlation outputs are combined. The regions of interest (ROI) containing the probable targets are selected from the input scene based on the ROIs with higher correlation peak values in the combined correlation output.

Zeng, H., J. Huang, and Y. Liang. el at 1999 "Combat Vehicle Classification Using Machine Learning." [17] This paper introduces two machine learning techniques-Algorithm Quasi-optimal (AQ) and Decision Tree (DT) as the classifiers for undertaking pattern recognition task. Both learn the 2D signal introduced from MSTAR SAR (Synthetic Aperture Radar) imagery database consisting of three classes of combat vehicles-BMP- 2, BTR-70, and T-72 tank. 67 images drawn from the database with similar aspect (+/- 15 degrees) are used for training the classifiers while unseen 47 images are used for testing.

Chapter 3

Theoretical Background

3.1 Sea-Borne

3.1.1 Balanced and Imbalanced data

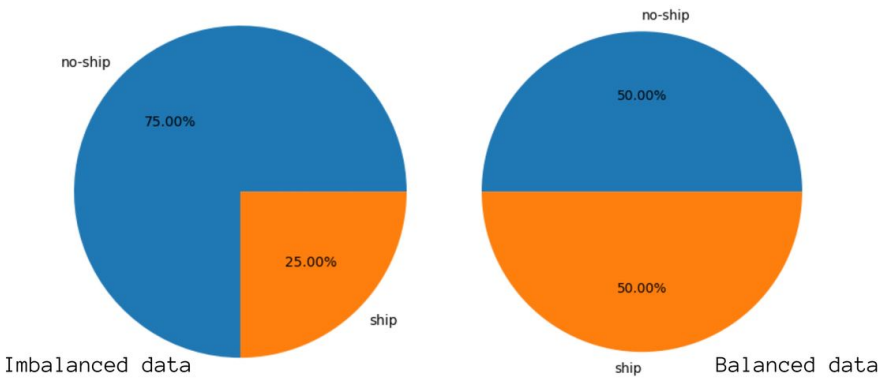


Figure 3.1: Balanced & Imbalanced data distribution

A balanced dataset is a dataset where each output class (or target class) is represented by the same number of input samples. For example, if we consider a two class problem , if the data set contains 50% of one class of problem and 50% of another class of problem then it is called balanced data.

Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e one class label has a very high number of observations and the other has a very low number of observations.

3.1.2 Data Augmentation

The first step in supervised learning is to test a variety of models against the training data and evaluate the models for predictive performance. After a model is validated and tuned with the validation data set, it is tested with the holdout data set to perform a final evaluation of its accuracy,

sensitivity, specificity, and consistency in predicting the right outcomes.

Holdout data: Holdout data refers to a portion of historical, labeled data that is held out of the data sets used for training and validating supervised machine learning models. It can also be called test data.

Holdout data is important in supervised machine learning to verify that the model that was trained and validated on historical data will produce similar performance when using new data while in operation. Holdout data should be kept separate from the training and validation data sets, and only used in the final assessment of the model's performance. This independence is important to prevent bias and to properly represent the behavior of the model with new data input going forward.

Image Augmentation

1. **Geometric transformations:** randomly flip, crop, rotate, stretch, and zoom images. You need to be careful about applying multiple transformations on the same images, as this can reduce model performance.
2. **Color space transformations:** randomly change RGB color channels, contrast, and brightness.
3. **Kernel filters:** randomly change the sharpness or blurring of the image.
4. **Random erasing:** delete some part of the initial image.
5. **Mixing images:** blending and mixing multiple images.

3.1.3 RGB channels, Pixel Intensity & Channel View

- **Pixel Intensity**: The image is divided into multiple equally sized units called pixels. **Each pixel in the image represents a discrete area in our sample and has an associated intensity value, a numerical representation denoting its brightness or darkness**. Since pixel intensity value is the primary information stored within pixels, it is the most popular and important feature used for classification. The intensity value for each pixel is a single value for a gray-level image, or three values for a color image.
- **Channel View**: Channel is a conventional term used to refer to a certain component of an image. Every image is made up of pixels and each pixel is made up of combinations of colours, to be more precise, primary colours. A channel is the grayscale image of a coloured image, which is made up of only one of the primary colours that form the coloured image.
- **RGB channels**: RGB channels are used in RGB images in which the color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. the red, green and blue components are stored separately in the R channel, G channel, and B channel respectively.

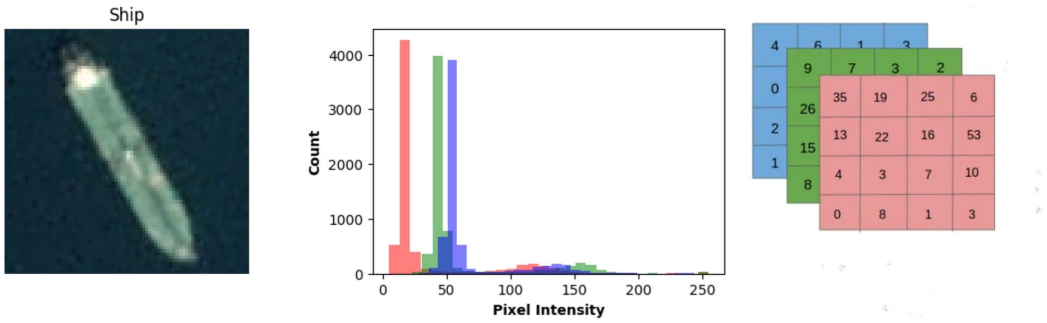


Figure 3.2: Satellite image; Pixel intensity; RGB channel view

3.1.4 ANN

Artificial neural networks (ANNs) are biologically inspired computer programs designed to simulate the way in which the human brain processes information. ANNs gather their knowledge by detecting the patterns and relationships in data and learn (or are trained) through experience, not from programming.

1. **Flatten layer:** The flatten layer lies between the CNN and the ANN, and its job is to convert the output of the CNN into an input that the ANN can process. The flatten layer is used to convert the feature map that it received from the max-pooling layer into a format that the dense layers can understand. A feature map is essentially a multi-dimensional array that contains pixel values; the dense layers require a one-dimensional array as input for processing. So the flatten layer is used to flatten the feature maps into a one-dimensional array for the dense layers.

2. **Dense layer:** Dense Layer is **simple layer of neurons in which each neuron receives input from all the neurons of previous layer**, thus called as dense. Dense Layer is used to classify image based on output from convolutional layers. This layer helps in changing the dimensionality of the output from the preceding layer so that the model can easily define the relationship between the values of the data in which the model is working.

3. **Weights and Biases: Weights** refer to connection managements between two basic units within a neural network. To train these units to move forward in the network, weights of unit signals must be increased or decreased. These connections will then be tested, reversed through the network to identify errors, and repeated to produce the optimal results.
   Biases in neural networks are additional crucial units in sending data to the correct end unit.

4. **Loss:** Loss functions are used to determine the error (aka "the loss") between the output of our algorithms and the given target value.
   Binary Cross Entropy is used for binary classification tasks with two classes, while Categorical Cross Entropy is used for multiclass classification tasks with more than two classes. The choice of loss function depends on the specific problem and the number of classes involved.
   Categorical cross entropy: Used as a loss function for multi-class classification model where there are two or more output labels The output label is assigned one-hot category encoding

value in form of 0s and 1. The output label, if present in integer form, is converted into categorical encoding using keras.

5. **ADAM Optimiser:** Adaptive Moment Estimation optimizer, is an optimization algorithm commonly used in deep learning. It is an extension of the stochastic gradient descent (SGD) algorithm and is designed to update the weights of a neural network during training.

3.1.5 CNN

1. **Convolutional layer:** The convolutional layer can be thought of as the feature extractor of this network, it learns to find spatial features in an input image. This layer is produced by applying a series of many different image filters, also known as convolutional kernels, to an input image. These filters are very small grids of values that slide over an image, pixel-by-pixel, and produce a filtered output image that will be about the same size as the input image. Multiple kernels will produce multiple filtered, output images.
   The idea is that each filter will extract a different feature from an input image and these features will eventually help to classify that image, for example, one filter might detect the edges of objects in that image and another might detect unique patterns in color. These filters, stacked together, are what make up a convolutional layer.

2. **Batch normalization:** Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.
   It is a two-step process. First, the input is normalized, and later rescaling and offsetting is performed.
   It serves to speed up training and use higher learning rates, making learning easier.

3. **Activation layer:** An activation layer in a CNN is a layer that serves as a non-linear transformation on the output of the convolutional layer. It is a primary component of the network, allowing it to learn complex relationships between the input and output data. The activation layer can be thought of as a function that takes the output of the convolutional layer and maps it to a different set of values. This enables the network to learn more complex patterns in the data and generalize better.

4. **Convolutional Layer (Convolutional Operation):** This process is main process for CNN. In this operation there is a feature detector or filter. This filter detects edges or specific shapes. Filter is placed top left of image and multiplied with value on same indices. After that all results are summed and this result is written to output matrix. Then filter slips to right to do this whole processes again and again. Usually filter slips one by one but it can be change according to your model and this slipping process is called 'stride'. Bigger stride means smaller output. Sometimes stride value is increased to decrease output size and time.

   (a) Conv2D - This is a 2 dimensional convolutional layer, the number of filters decide what the convolutional layer learns. Greater the number of filters, greater the amount of information obtained

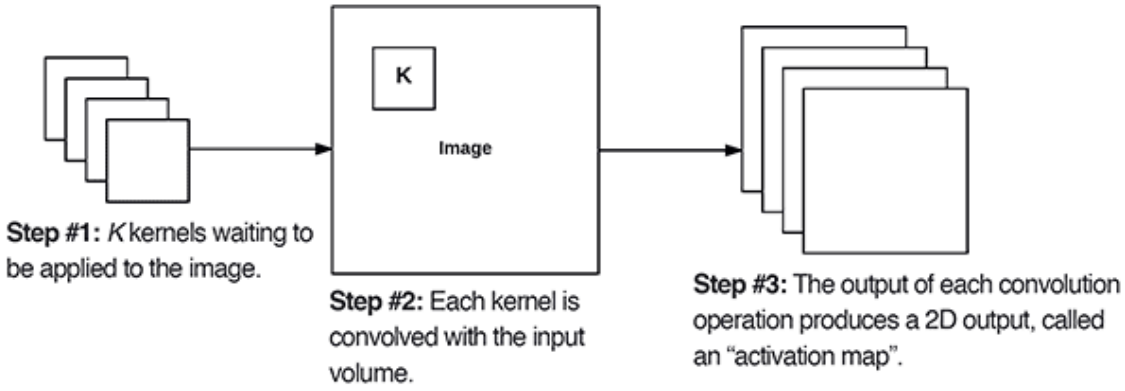


Figure 3.3: Steps within convolutional layer

(b) MaxPooling2D - This reduces the spatial dimensions of the feature map produced by the convolutional layer without losing any range information. This allows a model to become slightly more robust

(c) AveragePooling2D - A 2-D average pooling layer performs downsampling by dividing the input into rectangular pooling regions, then computing the average of each region. The window is shifted by strides along each dimension.

(d) Dropout - This removes a user-defined percentage of links between neurons of consecutive layers. This allows the model to be robust. It can be used in both fully convolutional layers and fully connected layers.

(e) Batch Normalization - This layer normalizes the values present in the hidden part of the neural network. This is similar to MinMax/Standard scaling applied in machine learning algorithms

(f) Padding- This pads the feature map/input image with zeros allowing border features to stay.

(g) Fully-connected - This layer has every output that's produced at the end of the last pooling layer is an input to each node in this fully-connected layer.

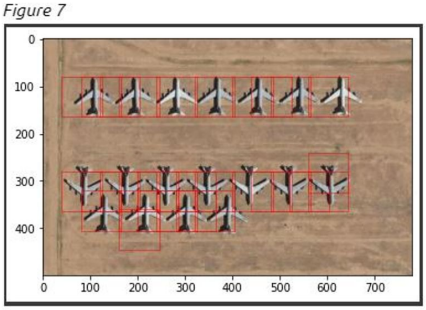
(h) Rescaling layer - A preprocessing layer which rescales input values to a new range. This layer rescales every value of an input (often an image) by multiplying by scale and adding offset.

3.2 Air-Borne

3.2.1 Instance annotation with Bounding Boxes

Horizontal bounding box: a rectangle drawn around the object's outermost edges that are parallel to the image's x and y axes. This type of bounding box is commonly used for object detection tasks.
Oriented bounding box: a rectangle drawn around the object's outermost edges that are oriented at an angle to the image's x and y axes. This type of bounding box is commonly used for tasks that involve objects with irregular shapes, such as aircraft.

Figure 7



3.2.2 CNN Hyper-parameter Tuning

There different parameters that can be tuned to improve the model:

1. Number of convolutional layers: more layers can potentially capture more complex features; however, too many convolutional layers can also lead to overfitting.

2. Number of filters: increasing this number can make the model learn more intricate patterns existing in the images; however, it increases computational cost.

3. Filter size: this parameter helps capture different levels of details and patterns. Increasing it might help exploit more details.

4. Pooling size: pooling is a technique of reducing spatial dimensions of an image; smaller pooling filter sizes retain more spatial information; the catch is it increases the computational cost.

5. Activation functions: the ability to learn and generalize is handled by activation functions. For example, if I change ReLU, which I used in the code above, to LeakyReLU or ELU, it might mitigate the "dying ReLU" problem.

6. Learning Rate: this is the step size taken to reach convergence during gradient descent optimization. If it's too high, may cause unstable training.

7. Batch size: batch size is the number of training samples processed before updating the model's weights. Although larger batch sizes may result in faster training, they require more memory. Smaller batch sizes on the other hand, can essentially lead to better generalization but slower convergence.

8. Number of epochs: this is the number of times the model iterates over the entire training set. Too few epochs can result in underfitting, while too many of them may result in overfitting.

9. Regularization: the dropout seen in my code is one of the regularization techniques; L1/L2 regularization and batch normalization are other examples. These techniques can help prevent overfitting.

10. Optimizer: the choice of optimizer affects the speed and quality of convergence.

There are different approaches to determining the most optimal hyper-parameters for a CNN model:

1. Grid Search: a grid of possible parameter values for each parameter to be tuned is defined. The grid search algorithm exhaustively evaluates all possible combinations of parameters and selects the one that results in the best performance.

2. Random Search: It involves sampling parameter values from predefined ranges randomly. It doesn't guarantee the best results due to random sampling.

3. Automated Hyper-parameter Tuning: we can use libraries such as Keras Tuner and employ their built-in functions that implement search techniques to handle the search process. Since automated, they can be quite efficient and find the most optimal configuration for the model.

We have utilized the last approach for this particular task.

3.2.3 Overfitting & Underfitting:

Overfitting and Underfitting are the two main problems that occur in machine learning and degrade the performance of the machine learning models. Before understanding the overfitting and underfitting, let's understand some basic term that will help to understand this topic well.

1. Bias: Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.

2. Variance: If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

Overfitting

This happens when a machine learning model excessively covers or captures more data points than necessary. This leads to the model memorizing noise and inaccuracies in the dataset, decreasing its efficiency and accuracy. Overfit models exhibit low bias and high variance. The likelihood of overfitting increases with prolonged training, so more training can result in overfitting. Overfitting is a common issue in supervised learning.

Underfitting

This occurs when a machine learning model fails to grasp the underlying data trend. To prevent overfitting, training data can be cut short, preventing the model from learning effectively. Consequently, it struggles to identify the dominant data trend, lowering accuracy and generating unreliable predictions. Underfit models possess high bias and low variance.

Chapter 4

Device & Software Requirements

4.1 Hardware Requirements

Deep Learning is resource-intensive. Trying to train relatively simple models on a laptop can take hours or even days. Because of that, it is advisable to use GPUs for Deep Learning tasks.

The readily available way to access GPUs for Deep Learning that we are using is via Kaggle(a cloud-hosted service for running data science projects).Very similar in style to Jupyter notebooks.

A GPU Kernel will give you Tesla P100 16gb VRAM or NVIDIA T4 GPU as GPU, with 13gb RAM + 2-core of Intel Xeon as CPU. No-GPU option will give you 4-cores + 16gb RAM, hence more CPU power.

The NVIDIA T4 GPU is based on the Turing architecture and is optimized for inference workloads that require high throughput and low power consumption, the Tesla P100 GPU is based on the Pascal architecture and is optimized for both inference and training workloads.

The P100 GPU offers higher performance than the T4 GPU, especially for training workloads that require high performance and memory capacity. This is due to its larger number of CUDA cores and higher clock speeds.

We can also check if GPU support is enabled for TensorFlow with the following:

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

Output

```
Num GPUs Available: 1
```

4.2 Software Requirements

TensorFlow.data.AUTOTUNE

'tf.data' builds a performance model of the input pipeline and runs an optimization algorithm to find a good allocation of its CPU budget across all parameters specified as 'AUTOTUNE'. While the input pipeline is running, 'tf.data' tracks the time spent in each operation, so that these times can be fed into the optimization algorithm.

As the last step in our pre-processing, we used TensorFlow's autotune capabilities to maximize the data's performance according to our hardware, and minimize the amount of time it takes models to train:

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds =train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds =val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds =test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

TensorFlow $\geq$ 2.8; Keras $\geq$ 2.8; Scikit-learn $\geq$ 1.0; TensorFlow Addons $\geq$ 0.14; Keras Applications $\geq$ 1.4.0; Scikit-Image $\geq$ 0.19.0; OpenCV $\geq$ 4.8.0

These versions of the software libraries are recommended because they are the latest stable versions and include all of the features needed for training models, data manipulation (EDA), and evaluation on image processing and classification tasks.

Chapter 5

Benchmarking & Assumptions

5.1 Sea-Borne

5.1.1 Benchmarking: Training, Validation and Testing

Instead of using *train_test_split* the images and labels arrays are randomly shuffled using the same seed value set at *42*. This allows the images and their corresponding labels to remain linked even after shuffling.

```
np.random.seed(42)
np.random.shuffle(images)

np.random.seed(42)
np.random.shuffle(labels)
```

This method allows the user to make all 3 datasets. The training and validation dataset is used for training the model while the testing dataset is used for testing the model on unseen data. Unseen data is used for simulating real-world prediction, as the model has not seen this data before. It allows the developers to see how robust the model is.

5.1.2 Assumptions: Splitting of data

70% - Training; 20% - Validation; 10% - Testing

5.2 Air-Borne

5.2.1 Benchmarking

We need the images to be of the same size. Since we are going to extract objects from each image, they will not be of the same size as the annotations do not cover the same area, i.e., some form a smaller rectangle with dimensions (24, 24), and some bigger ones like (111, 140).

5.2.2 Assumptions

Images need to be transformed in a way that they are of equal dimensions. Having said that, we also need to make sure the ratio of images don't change. The easiest way to do so, is to create a

squared empty image and paste the object onto the center of it, of which the function below takes care.

Chapter 6

Approaches

6.1 Sea-Borne

6.1.1 R-CNNS

R-CNNs (Region-based Convolutional Neural Networks) are a family of machine learning models used in computer vision and image processing. Specially designed for object detection, the original goal of any R-CNN is to detect objects in any input image defining boundaries around them.

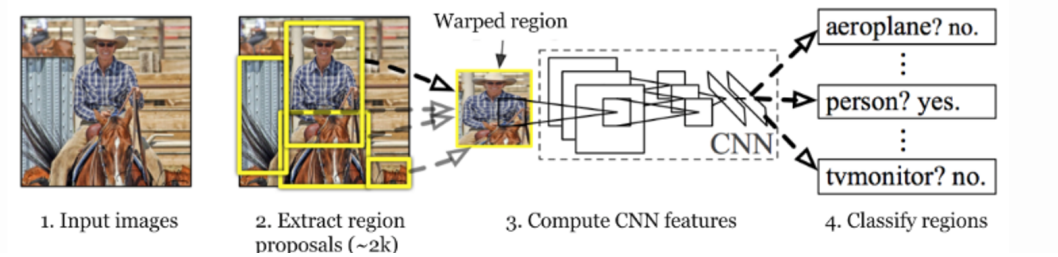


Figure 6.1: Steps followed within an R-CNN model

In the R-CNN model, input images undergo selective search to identify regions of interest, resulting in rectangular boundaries. These regions may number over 2000, and they pass through a CNN for feature extraction. The extracted features are then classified using an SVM.

6.1.2 Further Possibilities of Improvements

For image detection in images using Faster R-CNN the feature map produced by the convolutional layers is used, i.e., the model drops the layers after the final convolutional block and passes the generated feature map to a Regional Proposal Network, which can either use a vanilla CNN model containing fully connected layers or a Logistic Regression, Support Vector machines or Random forests. Advised to use vanilla CNN as it uses less CPU and memory, plus slightly faster and capable of giving multiple outputs if required.

6.2 Air-Borne

Our dataset comprises a diverse collection of airplane images captured under various conditions. To enhance our understanding and the capabilities of AI in aviation-related image analysis, we're looking to leverage the power of Graph Convolutional Networks.

Early neural networks worked with regular or Euclidean data, but real-world data often have non-Euclidean graph structures. This mismatch has driven advances in Graph Neural Networks (GNNs). Variants of GNNs, including Graph Convolutional Networks (GCN), have emerged in recent years, pioneered by **Thomas Kipf and Max Welling.** [**20**].

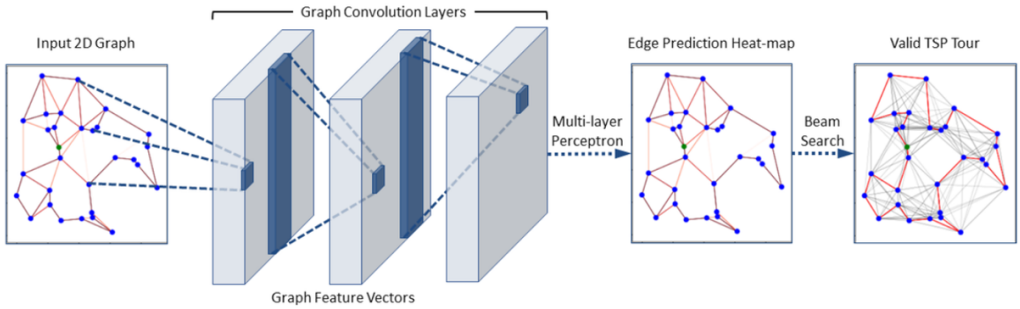


Figure 6.2: Steps followed within a GNN model

Graph Convolutional Networks are a cutting-edge architecture that excels in processing structured data, like graphs. In our context, each image can be treated as a graph, with pixels or regions representing nodes and their spatial relationships forming the edges. GCNs have demonstrated exceptional performance in tasks such as image classification and object detection by capturing both local and global context.

As for traditional CNNs, a GCN consists of several convolutional and pooling layers for feature extraction, followed by the final fully-connected layers.

6.2.1 Keras Tuner Hyperband Tuner

Hyperband quickly identifies poor hyperparameter results and trains multiple models briefly. the implementation of Hyperband trains multiple models for a small number of epochs; The top-performing models are then extended for further training in iterative cycles, ultimately yielding the best model.

This algorithm is one of the tuners available in the keras-tuner library. We must define a function that gets the parameters as the argument and returns a compiled model. Later, we will pass that function to the tuner.

```
# Create a KerasTuner Hyperband tuner
tuner = kt.Hyperband(
```

```
    build_model,
    objective='val_accuracy',
    max_epochs=100,
    directory='my_dir',
    project_name='my_project')

# Perform the hyperparameter search
tuner.search(X_train, y_train, epochs=60, validation_data=(X_test, y_test))

# Print the best hyperparameters found
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"Best Hyperparameters: {best_hps}")

# Build and train the model with the best hyperparameters
model = tuner.hypermodel.build(best_hps)
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test))
```

Chapter 7

Design and Methodology

7.1 Data Design

7.1.1 Sea-Borne

The dataset consists of images extracted from **Planet satellite** [**22**] imagery collected over the San Francisco Bay and San Pedro Bay areas of California. It includes 4000 80x80 RGB images. Images were derived from Planet Scope full-frame visual scene products, which are orthorectified to a 3-meter pixel size.
label: Valued 1 or 0, representing the "ship" class and "no-ship" class, respectively.

Figure 7.1: Example of "ship" images

Class Labels

The "ship" class includes 1000 images. Images in this class are centered on the body of a single ship. Ships of different sizes, orientations, and atmospheric collection conditions are included. Example images from this class are shown below.

The "no-ship" class comprises 3000 images, evenly divided into three categories: random land cover features (excluding ships), partial ships, and mislabeled images (usually due to bright pixels or linear features).

Figure 7.2: Example of "no-ship" images

Two columns in this project - data and labels. the pixel value data for each 80x80 RGB image is stored as a list of 19200 integers within the data list. The first 6400 entries contain the red channel values, the next 6400 the green, and the final 6400 the blue.

Data Preprocessing

Due to imbalance in dataset, upsampling is done on the minority class, by randomly duplicating images until the 2 classes have comparable distribution in the dataset. After this is done, the dataset will be split into the training, testing and validation sets by randomly shuffling them and then splitting.

Another way is to introduce class weights for each specific class. Each class is penalised with the specific class weight. Higher the class weight, greater the penalty. Classes with lower percentage have a higher penalty. This allows for the model to penalise itself heavily if class detected is incorrect.

EDA of dataset

- bar-plot: Bar plot is made to find the count of images per class
- pie-plot: Pie plot is drawn to find the percentage of class distribution in the dataset

Augmenting Minority Class Images

We augment images in the *ship* class to balance class representation. The current class ratio is 1:3 (1 *ship* image for every 3 *no-ship* images). To balance it, we generate 2 augmented images per original *ship* image, achieving a balanced dataset.

If augmentation is needed, set AUGMENTATION to *True*. This balances the dataset by augmenting minority classes. To train using class weights, set AUGMENTATION to *False*.

With AUGMENTATION set to *True*, class balance is achieved. If AUGMENTATION is set to *False*, calculate class weights and adjust the Keras API's fit function accordingly during training.

7.1.2 Air-Borne

Fine-Grained Visual Classification of Aircraft FGVC-Aircraft dataset

The original dataset comprises 10,200 aircraft images, with 100 images per each of 102 aircraft model variants, primarily airplanes. Our Kaggle version contains 3,842 images, spanning 20 types, with 22,341 instances. Each image annotates the main aircraft with a tight bounding box and a hierarchical airplane model label.

In the dataset, the folders "Horizontal Bounding Boxes" and "Oriented Bounding Boxes" contain 'xml' files with annotations, detailing aircraft names and rectangle coordinates encapsulating the objects.

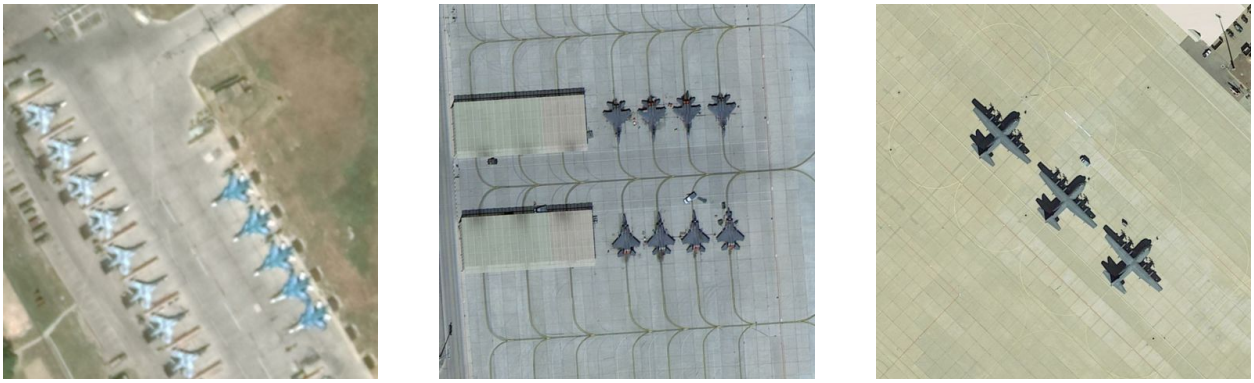


Figure 7.3: A1 & A19, A13, and A2 Aircraft Type as Label respectively.

Images must be standardized to a consistent size to facilitate object extraction. Annotations, which cover varying areas, result in non-uniform object sizes, ranging from small rectangles (e.g., 24x24) to larger ones (e.g., 111x140). To ensure equal dimensions while maintaining aspect ratios, a straightforward approach involves creating squared blank images and placing objects at their centers, a process managed by the following function.

- **def pad_img(img):** This function transforms an image to the desired size. If squared, the image remains the same in ratio; if rectangular, it'll be padded evenly to become a squared one.
- **def extract_obj(img, annotations):** This function extracts objects from an image based on provided annotations.
- **def preprocessing(data):** This function prepares and organizes image data and corresponding labels, making it ready for training machine learning models. This preparation includes tasks such as data extraction, padding, labeling, encoding, normalization, and conversion into appropriate data structures.

7.2 Architectural Design

7.2.1 Sea-Borne

In the initial phase of this study, we embarked on the development of a straightforward Artificial Neural Network (ANN) model to address the specific task at hand.

The architectural design of this preliminary ANN model adhered to the classic Sequential structure, comprising layers:

```
Model: "sequential"
_________________________________________________
 Layer (type)        Output Shape      Param #
=================================================
 flatten (Flatten)   (None, 19200)     0

 dense (Dense)       (None, 200)       3840200

 dense_1 (Dense)     (None, 150)       30150

 dense_2 (Dense)     (None, 2)         302

=================================================
Total params: 3,870,652
Trainable params: 3,870,652
Non-trainable params: 0
```

The model boasts a considerable total of 3,870,652 parameters, a substantial portion being trainable. This extensive parameterization, while enabling effective learning from the training data, also elevates the risk of overfitting, as observed in initial results.

To address overfitting and enhance predictive capabilities, this study progressed to explore advanced deep learning models.

Convolution Model Creation

- **conv_block:** This function iteratively applies convolutional layers, batch normalization, and activation layers. Developer-defined parameters include the number of filters, kernel size, and strides
- **basic_model** - This function creates the model using the **conv_block**, max pooling layers and dropouts, introducing flatten layer and dense layers.

For image detection using Faster R-CNN, the model utilizes the feature map from convolutional layers. It either employs a vanilla CNN model with fully connected layers or alternative methods such as Logistic Regression, Support Vector Machines, or Random Forests. **Block 1:** The model begins with an input layer defined using the Keras Input layer, specifying the input shape. ZeroPadding is applied to the input image, so that boundary features are not lost.

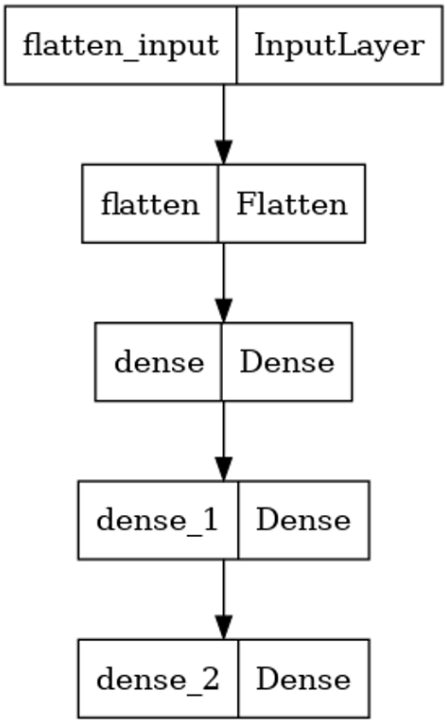


Figure 7.4: The architectural design of the ANN model-classic Sequential structure

Figure 7.5: Bar representation of layers in ANN

Block 2: In this block, First Convolutional Layer starts with 16 filters and kernel size with (3,3) and strides (2,2). Padding is maintained same, so the image does not changes spatially, until the next block in which MaxPooling occurs.

Block 3-4: In both these blocks Similar structure with a convolutional layer followed by a Max-Pooling and Dropout layers were used.

Output Block: The feature map produced by the previous convolutional layers is converted into a single column using Flatten Layer and the classified using a Dense layer(output layer) with the number of classes present in the dataset, and sigmoid as activation function.

7.2.2 Air-Borne

We utilize a model-building function, **build_model(hp)**, which receives a **HyperParameters** object (hp) to search for optimal hyperparameters. The Keras Tuner, specifically the Hyperband algorithm, efficiently explores the hyperparameter space within a predefined number of training epochs, maximizing validation accuracy while preventing overfitting using early stopping.

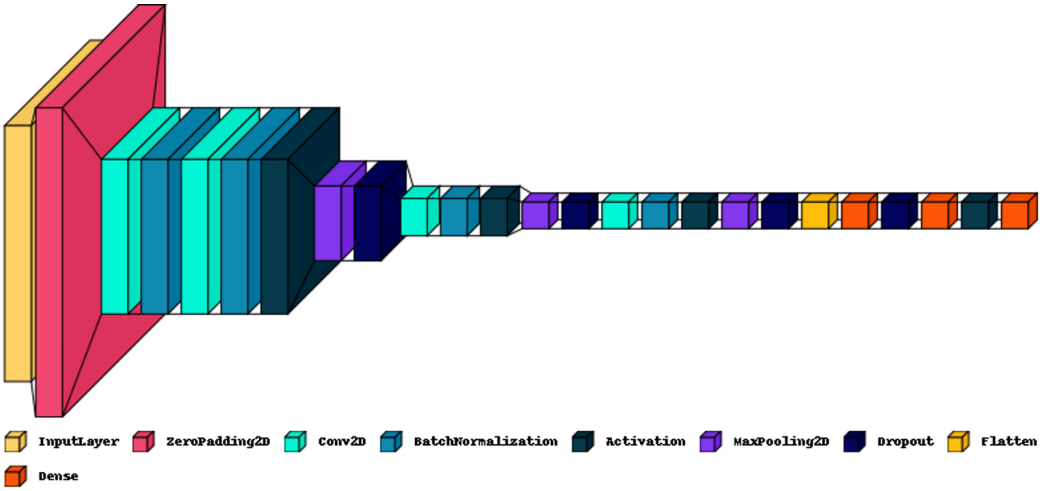


Figure 7.6: Layers of convolution model for Sea Borne

The best hyperparameters, determined through tuning, are then applied to construct the final model. This model undergoes extensive training on the training dataset to maximize performance and is subsequently evaluated on a separate test dataset to assess accuracy and generalization.

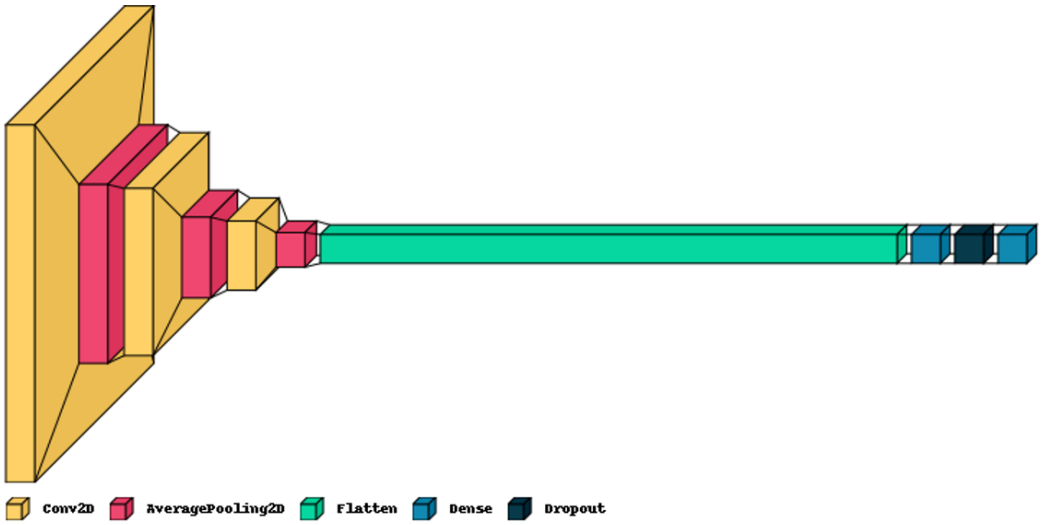


Figure 7.7: Layers of convolution model for Air Borne

Here are the key components of the **build_model** function:

- Input Layer:
  - Convolutional Layer with 32 filters, 3x3 kernel, and ReLU activation.
  - Input shape of (64, 64, 3) for images.
- Convolutional Layers:

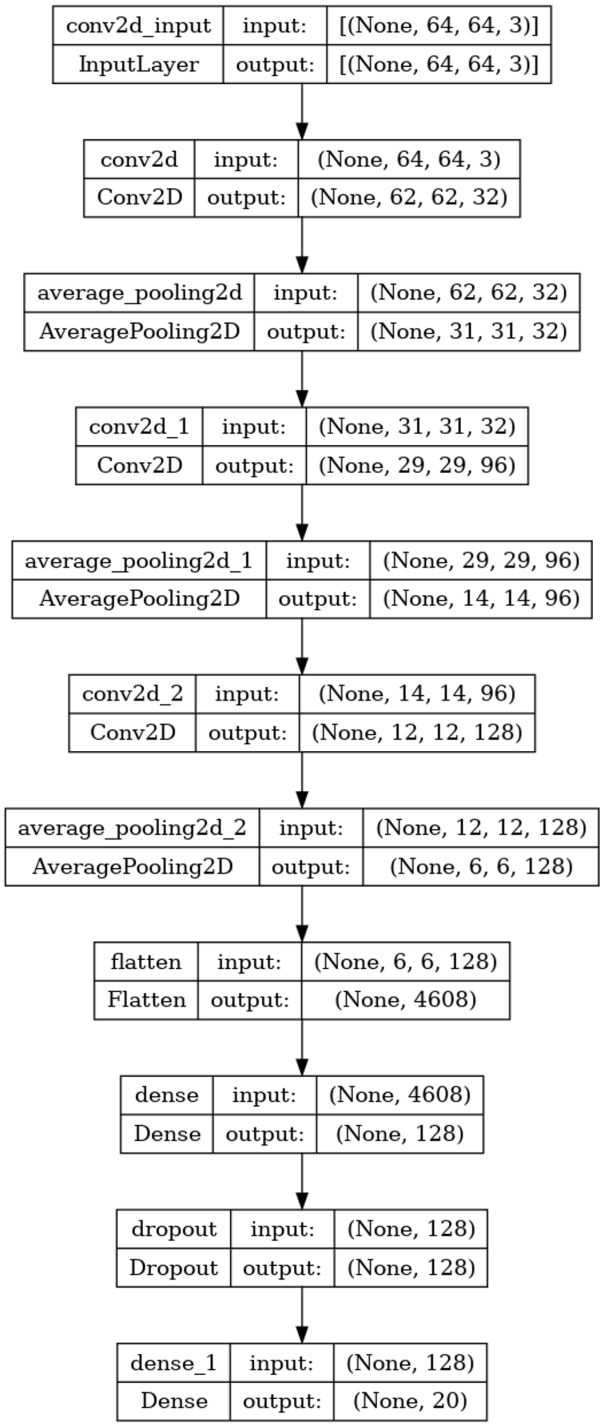


Figure 7.8: Best Model Structure

– The number of convolutional layers is tunable within the range of 1 to 3..
– Each convolutional layer includes.

    - A tunable number of filters (between 32 and 128 with a step of 32).
    - A 3x3 kernel and ReLU activation.
    - Average Pooling Layer with a 2x2 pool size.

- Flattening Layer: Converts the output from convolutional layers into a flat vector.
- Dense Layer:
  - The number of units in the dense layer is tunable within the range of 64 to 256 with a step of 32.
  - ReLU activation.
- Dropout Layer:
  - A tunable dropout rate between 0.3 and 0.6 with a step of 0.1.
- Output Layer:
  - Dense layer with the appropriate number of output units (n_labels) and softmax activation.
- Compilation:
  - The optimizer's learning rate is tunable and can be chosen from [1e-3, 1e-4].
  - Adam optimizer is used for optimization.
  - Categorical cross-entropy loss is employed for classification tasks.
  - Accuracy is used as the evaluation metric.

7.3 Procedural Design

7.3.1 Sea-Borne

ANN Implementation

This procedural design outlines the steps to create, configure, train, and evaluate the ship-detection model. It also incorporates an early stopping mechanism to prevent overfitting.
After model initialization, we have used the following procedure:

1. Input Layer
   - Add an input layer to the model using **model.add(Flatten(input_shape=[80, 80, 3]))**.
   - This layer flattens the 80x80x3 input image into a 19200-dimensional vector.
2. Hidden Layers
   - Add two dense hidden layers to the model:
     - First hidden layer with 200 units and ReLU activation: **'model.add(Dense(200, activation='relu'))'**.

- Second hidden layer with 150 units and ReLU activation: **model.add(Dense(150, activation='relu'))**.

3. Output Layer

   - Add an output layer to the model with two units and a sigmoid activation function:
     - **model.add(Dense(2, activation='sigmoid')**.
   - This is suitable for binary classification (ship or non-ship).

4. Model Compilation

   - Compile the model by specifying:
     - - Loss function: 'categorical_crossentropy' (appropriate for multi-class classification).
     - Optimizer: 'Adam' (a popular optimization algorithm).
     - Metrics for evaluation: ['accuracy'].
   - **model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])**.

5. Early Stopping Callback

   - Define an early stopping callback to monitor validation loss and prevent overfitting.
     - earlystopping = callbacks.EarlyStopping(monitor="val_loss", mode="min", patience=10, restore_best_weights=True).

6. Model Training

   - Train the model using the following parameters:
     - Training data: x_train and corresponding labels y_train.
     - Number of training epochs: 100.
     - Validation data: x_val and corresponding labels y_val.
     - Early stopping callback to prevent overfitting.
   - history = model.fit(x_train, y_train, epochs=100, validation_data=(x_val, y_val), callbacks=[earlystopping]).

```
from keras import callbacksmodel = Sequential()
model.add(Flatten(input_shape=[80, 80, 3]))
model.add(Dense(200, activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

model.compile(loss='categorical_crossentropy', optimizer='Adam',
metrics=['accuracy'])
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
                                        mode ="min", patience = 10,
                                        restore_best_weights = True)
```

```
history = model.fit(x_train, y_train, epochs = 100,
validation_data=(x_val, y_val), callbacks = [earlystopping])


Epoch 1/100
75/75 [==============] - 6s 10ms/step - loss:
0.7728 - accuracy: 0.7887 - val_loss:
0.3624 - val_accuracy: 0.8700
Epoch 2/100
75/75 [==============] - 0s 5ms/step - loss:
0.3580 - accuracy: 0.8567 - val_loss:
0.3019 - val_accuracy: 0.8675
.......
.......
Epoch 27/100
75/75 [==============] - 0s 5ms/step - loss:
0.1349 - accuracy: 0.9463 - val_loss:
0.1620 - val_accuracy: 0.9425
.......
.......
Epoch 37/100
75/75 [==============] - 0s 5ms/step - loss:
0.1494 - accuracy: 0.9429 - val_loss:
0.2114 - val_accuracy: 0.9225
!!! Early Stopping by monitoring "validation loss" with patience = 10 !!!
```

7.3.2 ANN - MLP

As a starting point, We built a simple multi-layer perceptron (MLP) model, which is just a fancy way of saying it's a model with fully-connected layers. We set each of our three layers to have 100 nodes, and used a Rectified Linear Unit (ReLU) activation function:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 16384)             0

dense (Dense)                (None, 100)               1638500

dense_1 (Dense)              (None, 100)               10100

dense_2 (Dense)              (None, 100)               10100

dense_3 (Dense)              (None, 9)                 909
```

```
==========================================================
Total params: 1,659,609
Trainable params: 1,659,609
Non-trainable params: 0
----------------------------------------------------------
```

CNN Implementation

1. Convolutional Block (**conv_block** Function)
   - Create a function conv_block to define a convolutional block with the following parameters:
     - **X**: Input tensor
     - **k**: Kernel size for convolution
     - **filters**: Number of filters in the convolutional layer
     - **stage**: Stage of the block (integer)
     - **block**: Block identifier (e.g., 'a', 'b', 'c')
     - **s**: Stride for convolution
2. Inside the conv_block function:
   - Define convolution and batch normalization layers using provided parameters.
   - Apply the ReLU activation function.
   - Return the updated tensor X.
3. Model Construction
   - - Build the complete ship-detection model by stacking multiple 'conv_block' layers along with additional layers, such as pooling and fully connected layers.
4. Model Compilation
   - - Compile the model with appropriate loss function, optimizer, and evaluation metrics.
   - - Define callbacks for monitoring and controlling the training process (e.g., early stopping).
5. Training
   - Train the model using the prepared dataset.
   - Specify the number of epochs and batch size for training.
   - Utilize the defined callbacks to control training, such as early stopping if needed.

```
def conv_block(X,k,filters,stage,block,s=2):

    conv_base_name = 'conv_' +str(stage)+block+'_branch'
    bn_base_name = 'bn_'+str(stage)+block+"_branch"
```

```
F1 =filters

X =Conv2D(filters=F1, kernel_size=(k,k), strides=(s,s),
      padding='same',name=conv_base_name+'2a')(X)
X =BatchNormalization(name=bn_base_name+'2a')(X)
X =Activation('relu')(X)

return X
```

7.3.3 Air-Borne

1. Initialization
   - Import the required libraries and dependencies, including TensorFlow and Keras.
2. Model Creation
   - Initialize a Keras Sequential model using keras.Sequential().
3. Input Layer
   - Add the input layer with the following configurations:
     - Convolutional Layer with 32 filters, a 3x3 kernel, and ReLU activation.
     - Input shape of (64, 64, 3) for images.
4. Convolutional Layers
   - Determine the number of convolutional layers to be added:
     - The number of convolutional layers is tunable within the range of 1 to 3.
   - For each convolutional layer:
     - Add a convolutional layer with the following parameters:
       * A tunable number of filters (between 32 and 128 with a step of 32).
       * A 3x3 kernel and ReLU activation.
     - Add an Average Pooling Layer with a 2x2 pool size.
5. Flattening Layer
   - Add a flattening layer to convert the output from the convolutional layers into a flat vector.
6. Dense Layer
   - Configure the dense layer as follows:
     - The number of units in the dense layer is tunable within the range of 64 to 256 with a step of 32.
     - Apply ReLU activation.
7. Dropout Layer

- Introduce a dropout layer with a tunable dropout rate between 0.3 and 0.6 with a step of 0.1.

8. Output Layer
   - Add the output layer with the following settings:
     - A Dense layer with the appropriate number of output units (n_labels) and softmax activation.
9. Compilation
   - Configure the model's compilation with the following settings:
     - Determine the learning rate for the optimizer as a choice from [1e-3, 1e-4].
     - Use the Adam optimizer for optimization.
     - Apply categorical cross-entropy loss for classification tasks.
     - Monitor and report accuracy as the evaluation metric.
10. Return Model
    - Return the configured model.
11. Hyperparameter Tuning (not part of the function)
    - Use the Keras Tuner library (e.g., Hyperband tuner) to search for optimal hyperparameters by calling the 'build_model' function repeatedly with various hyperparameter combinations.
12. Final Model Training (not part of the function)
    - Use the best hyperparameters obtained from hyperparameter tuning to build the final model.
    - Train the model on the training dataset for a more extended period.

This procedural design provides a structured approach to creating and configuring machine learning models with the flexibility to adapt to different tasks and datasets. The focus is on the configurability of hyperparameters and efficient hyperparameter tuning to enhance model performance.

```
# Define a model-building function
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=
    (64, 64, 3)))
    model.add(layers.AveragePooling2D((2, 2)))
# Tune the number of convolutional layersfor iin range(hp.Int('conv_layers',
    1, 3)): model.add(layers.Conv2D(hp.Int
    (f'filters_{i}', 32, 128, step=32),(3, 3), activation='relu'))
    model.add(layers.AveragePooling2D((2, 2)))
    model.add(layers.Flatten())
```

```
  # Tune the number of units in the dense layer
model.add(layers.Dense(hp.Int('units', 64, 256, step=32),activation='relu'))
  # Add a dropout layer
model.add(layers.Dropout(hp.Float('dropout', 0.3, 0.6, step=0.1)))
  # Tune the learning rate for the optimizer
hp_learning_rate = hp.Choice('learning_rate', values=[1e-3, 1e-4])
      optimizer = keras.optimizers.Adam(learning_rate=hp_learning_rate)
      model.add(layers.Dense(n_labels, activation='softmax'))
      model.compile(optimizer=optimizer,loss='categorical_crossentropy',
      metrics=['accuracy'])
      return model


  # Perform the hyperparameter search using KerasTuner Hyperband tuner
  Trial 254 Complete [00h 01m 57s]
  val_accuracy: 0.878588080406189

  Best val_accuracy So Far: 0.9301784038543701
  Total elapsed time: 01h 06m 06s

  # Get the optimal hyperparameters
  Number of conv blocks: 2
  filters_0: 96
  filters_1: 128
  filters_0: 96
  filters_1: 128
  units: 128
  dropout: 0.4
  learning_rate: 0.001
```

Chapter 8

Results & Evaluation

8.1 Sea-Borne

8.1.1 ANN

Artificial Neural Network (ANN) model: This simple model achieved high training and evaluation accuracy, which could signal overfitting, where the model performs well on training data but may struggle with unseen data.

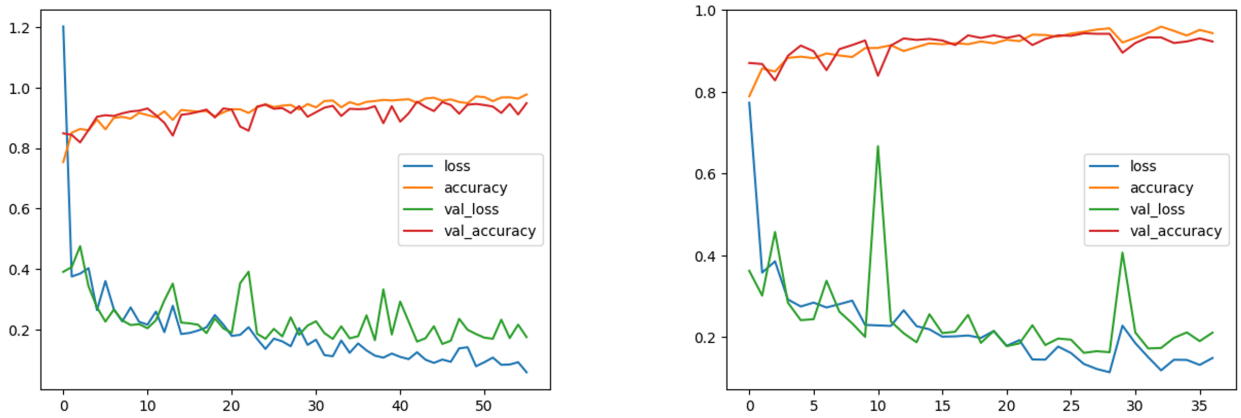


Figure 8.1: Iteration run 1 & 2 respectively.

```
model.evaluate(x_test, y_test)
25/25 [==============] - 0s 3ms/step - loss: 0.2083 - accuracy: 0.9112
[0.2083025723695755, 0.9112499952316284]
```

8.1.2 CNN

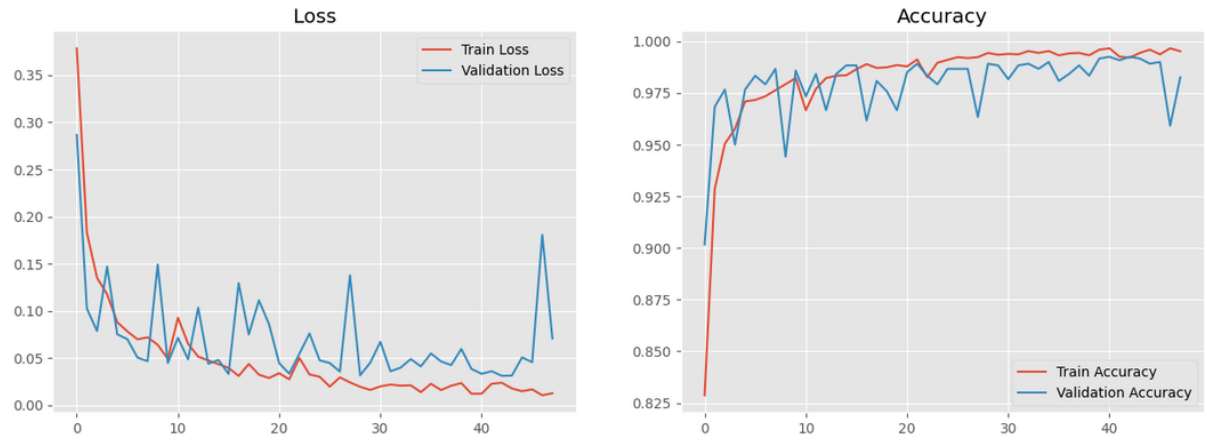


Figure 8.2: The loss and accuracy for the training and validation datasets is plotted using the show_final_history function

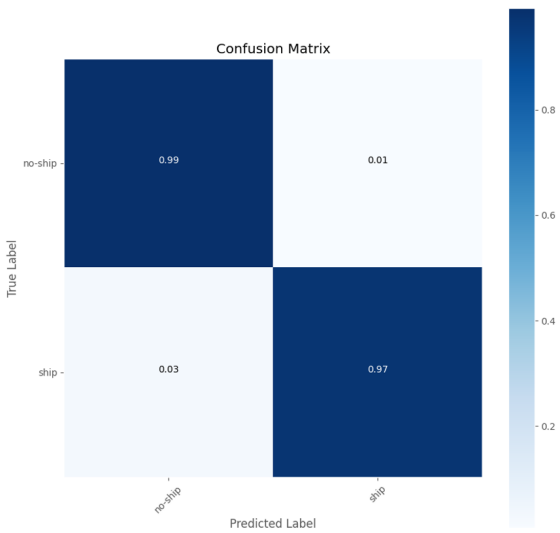


Figure 8.3: Confusion Matrix for Validation dataset

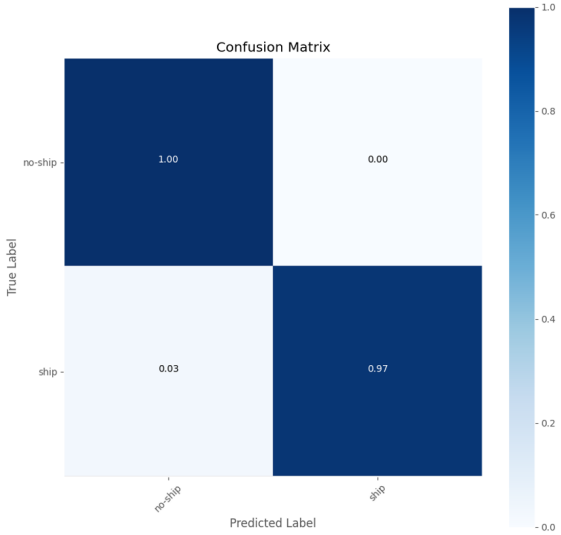


Figure 8.4: Confusion Matrix for Validation dataset

With Augmentation:
- Precision: no-ship: 0.942; ship: 0.989
- Recall: no-ship: 0.99; ship: 0.94

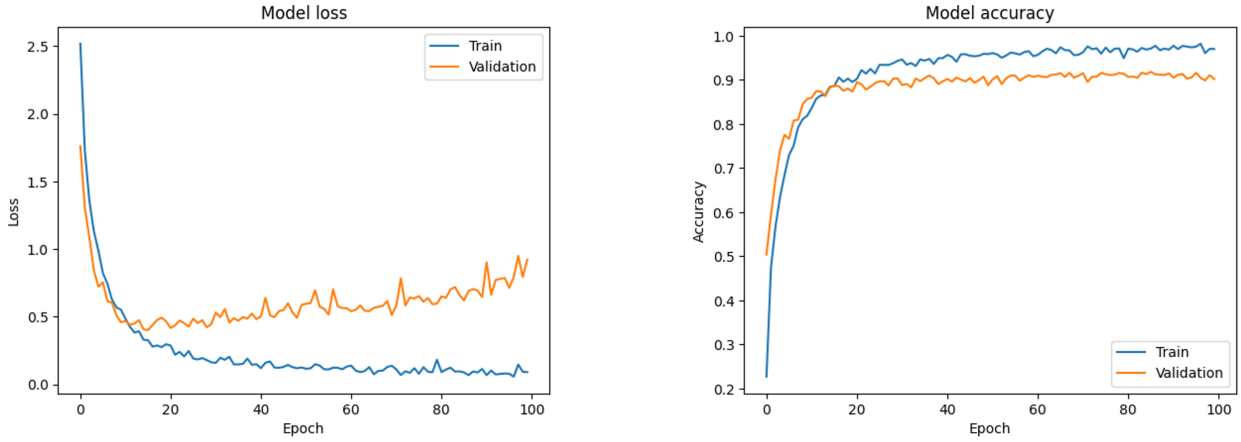
With class weights:
- Precision: no-ship: 1.00; ship: 0.99
- Recall: no-ship: 0.99; ship: 1.00

8.2 Air-Borne

```
# Build and train the model with the best hyperparameters
Epoch 1/100
164/164 [==========] - 3s 9ms/step - loss: 2.5173 - accuracy: 0.2268
- val_loss: 1.7586 - val_accuracy: 0.5039
```

```
.......
Epoch 100/100
164/164 [==========] - 1s 7ms/step - loss: 0.0898 - accuracy: 0.9700
- val_loss: 0.9213 - val_accuracy: 0.9022
# Test the model
81/81 [==========] - 0s 3ms/step - loss: 0.9213 - accuracy: 0.9022
Model: "sequential_1"
Test Loss: 0.9212860465049744
Test Accuracy: 0.902249813079834
```

8.2.1 Plot the performance of the model



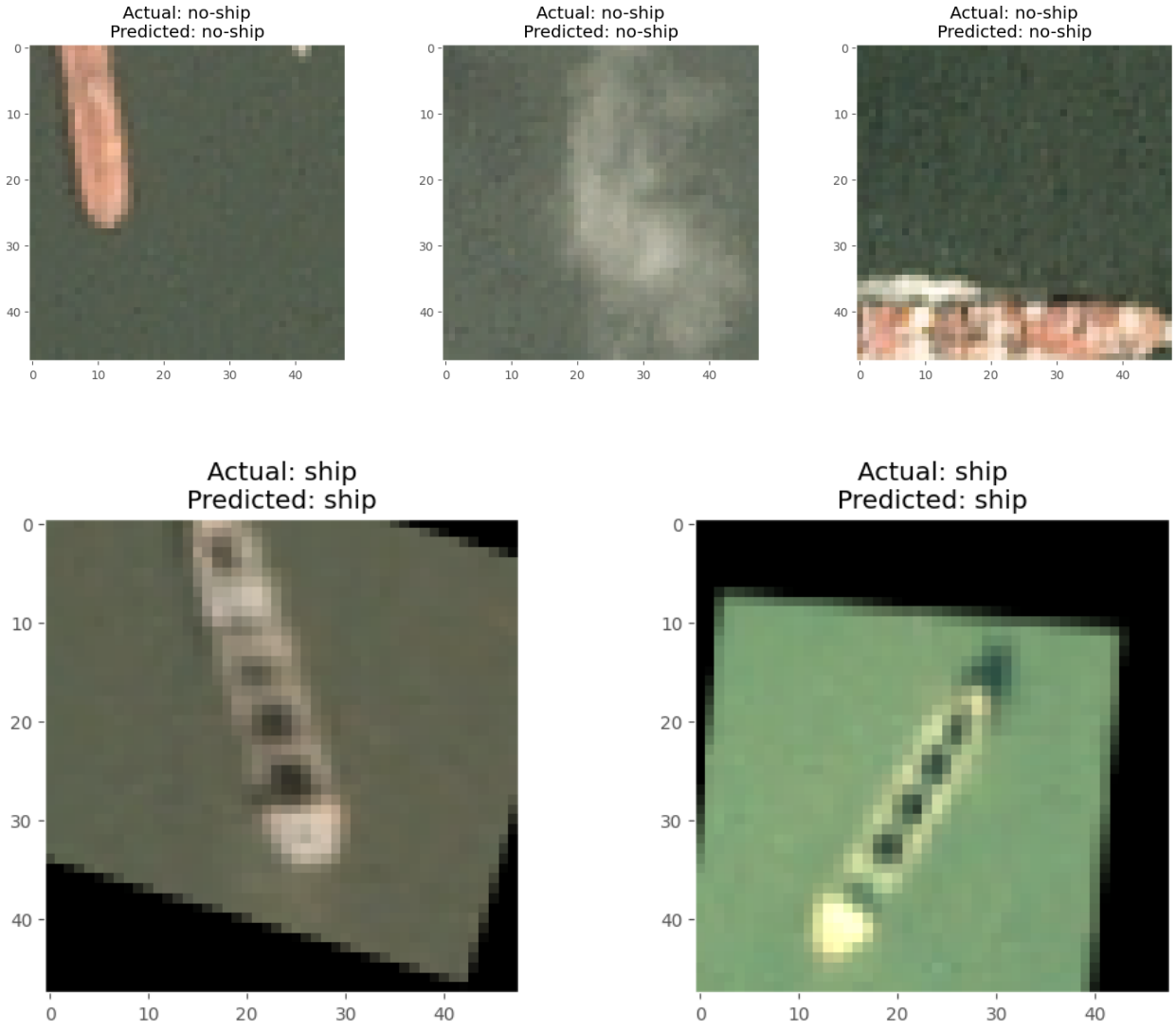
Interpretation:

1. Initially, the model quickly generalizes well to the validation data, indicated by higher validation accuracy, suggesting it captures meaningful data patterns without overfitting.

2. In the transition stage (around epoch 20), the model exhibits signs of overfitting. Overfitting occurs when the model specializes in training data but struggles to generalize to unseen data. Fluctuations in both accuracies show the model is fitting the training data more precisely, but this doesn't necessarily improve validation performance. The increasing gap between training and validation accuracies suggests slower improvement on the validation set.

8.2.2 Prediction and displaying a subset of results

Sea-Borne



Air-Borne

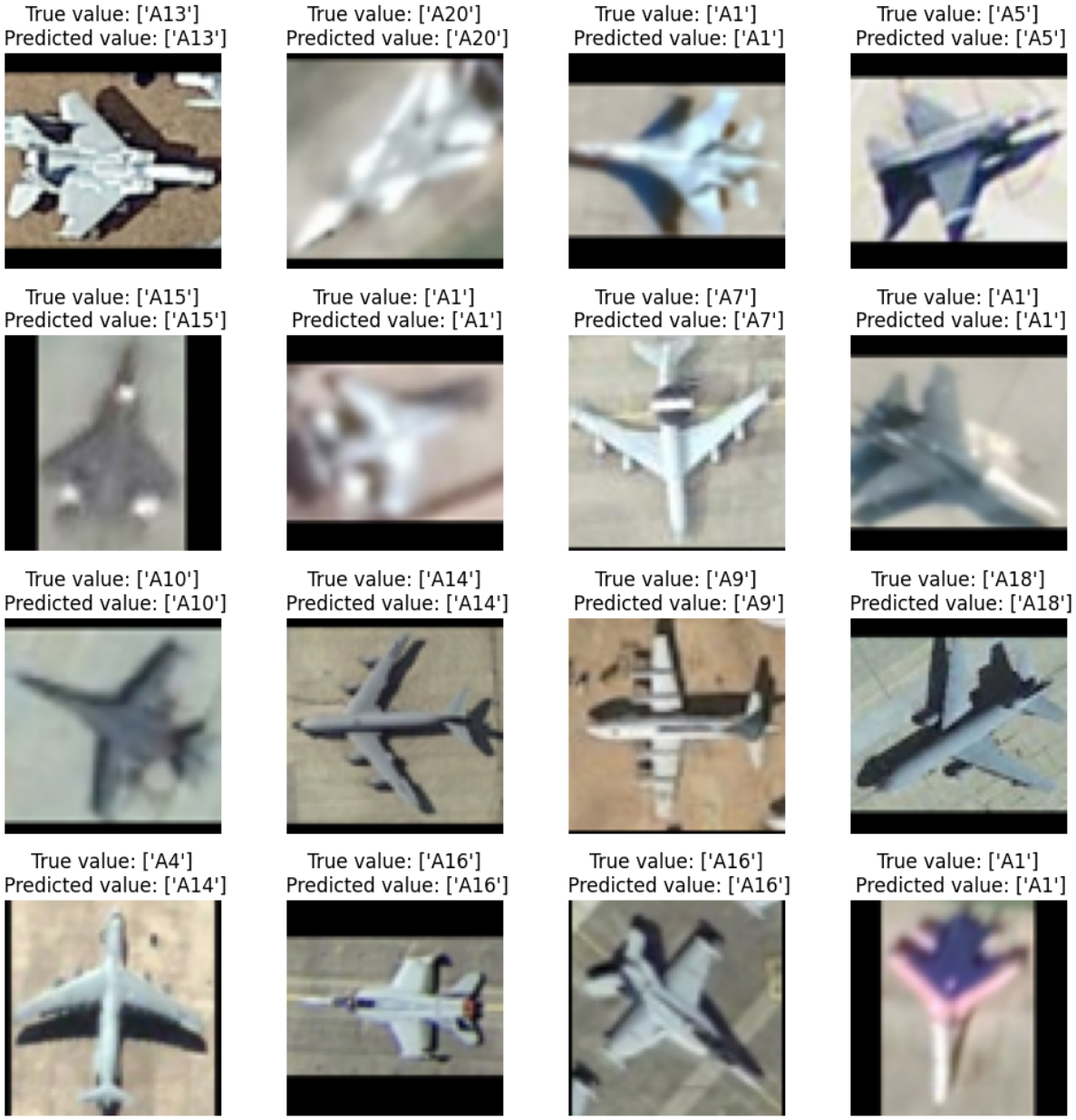


Figure 8.5: Predicted results with True and Predicted values

Chapter 9

Conclusion

In summary, these models demonstrated commendable performance and efficiency through quantitative evaluations. Qualitative assessments of the test dataset further confirmed their robust generalization over specific data type such as SAR, indicating promising real-world applications. The models effectively detected the absence of military vehicles and accurately classified them, even in cases where human assessment might falter. These findings highlight the models' potential in situations where the identification of military vehicles poses challenges to human observers. They emphasize the use of machine learning and computer vision for military vehicle detection in satellite and aerial imagery, paving the way for future research and advancements in this field.

9.1 Ethics & Social Responsibilities

In our project development, we were mindful of the potential ethical implications and unintended uses of the models we created.
While our models have valid applications, such as documentation and accountability, we acknowledge the ethical concerns that can arise, especially in conflict scenarios, where similar models might be used for potentially harmful purposes.
As responsible data scientists and machine learning practitioners, we maintained a strong ethical focus throughout the project's lifecycle. Our ethical considerations included:

1. Evaluating the potential for unintended or harmful model uses.
2. Assessing and addressing biases in our training data and identifying knowledge gaps.
3. Understanding the implications of model inaccuracies and defining its intended users.
4. Evaluating the risks associated with open access to the model and ensuring our data collection complies with legal and ethical standards while respecting privacy.

These ethical considerations were central to our work, and we recognized the importance of a collaborative, multidisciplinary approach to thoroughly assess ethical implications.// After a comprehensive ethical assessment, we made an informed decision to proceed with the project. However, we acknowledge that our final model has limitations that restrict its applicability to contemporary imagery. It primarily serves as a proof of concept for educational and research purposes. It's important to emphasize that if our model had the potential for unintended and harmful applications,

we would have refrained from making it publicly available, prioritizing ethical principles. There is a fine line that sometimes exists between beneficial solutions and potentially harmful solutions.

Chapter 10

Future Scope

There remains ample opportunity for further improvement and exploration in this field. Some potential areas for future work include:

1. Diverse Dataset Enhancement: Expanding the dataset's diversity by introducing additional environmental factors and occlusions to enhance the model's generalization.
2. Synthetic Data Generation and Domain Adaptation: Investigating the use of synthetic data generation and domain adaptation techniques to increase the volume of training samples.
3. Alternative Architectures: Exploring other deep learning architectures like U-Net and DeepLab and comparing their performance with R-CNN.
4. Cross-Platform Testing: Testing the model on various satellite and aerial imagery platforms and sensor modalities to enhance its robustness.
5. Data Variability: Incorporating noisy images, clutter, and obfuscated vehicles in the dataset to better simulate real-world conflict environments.
6. Environmental Factors: Investigating the impact of factors such as the satellite's angle relative to the target and the influence of vehicle shadows based on the time of day.

At a tactical level, AI can enhance semi-autonomous control of unmanned systems, allowing human operators to efficiently manage such systems, ultimately increasing their impact on the battlefield.

Deception and Adaptability: It's essential to acknowledge the susceptibility of DL-based models to deception through signal manipulation. For example, advanced object detection in unmanned aerial vehicles (UAVs) may be misled by carefully designed camouflage patterns on the ground. As a result, military organizations may need to adapt their data collection processes to fully harness modern AI techniques such as deep learning and computer vision.

The continuous development of these technologies holds significant promise for enhancing military applications, but it's imperative to address and adapt to the associated challenges.

References

[**1**] Girshick, Ross. "Fast R-CNN." Proceedings of the IEEE international conference on computer vision. 2015.

[**2**] Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.

[**3**] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. "Imagenet: A large-scale hierarchical image database." 2009 IEEE conference on computer vision and pattern recognition. 2009.

[**4**] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature. 2015.

[**5**] Everingham, Mark, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The PASCAL visual object classes challenge 2010 (VOC2010) development kit." 2010.

[**6**] Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Piotr Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. "Microsoft COCO: Common objects in context." European conference on computer vision. 2014.

[**7**] Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single shot multibox detector." European conference on computer vision. 2016.

[**8**] Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[**9**] Liu, Wei, and Gábor Mattyus. "Detecting military vehicles in satellite imagery using deep learning." IEEE Geoscience and Remote Sensing Letters. 2018.

[**10**] Tuermer, Kevin, Kai Blaschke, and Sören Tiede. "Object detection in satellite imagery using deep learning: A review." arXiv preprint arXiv:1903.07110. 2019.
[**11**] Cheng, Gaojun, Junwei Han, Xiaodan Li, Xinggang Wang, Qi Tian, Wenyu Liu, and Hongwei Zhang. "Military vehicle detection in satellite imagery using deep learning with multiple-scale feature fusion." IEEE Transactions on Geoscience and Remote Sensing. 2020.

[**12**] Shao, Jiaming, Yuanyuan Liu, Weiwei Sun, Jingjing Zhu, and Qi Tian. "Military vehicle detection in satellite imagery using a deep learning model with a multi-scale attention mechanism." IEEE Access. 2021.

[**13**] Girshick, Ross. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[**14**] Maji, Subhransu, Esa Rahtu, Juho Kannala, and Matti Pietikäinen. "Fine-grained visual classification of aircraft." Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013.

[**15**] Li, Lisha, and Kevin Jamieson. "Hyperband: A novel bandit-based approach to hyperparameter optimization." International Conference on Machine Learning. 2016.

[**16**] Sims, S. Richard F., and M. J. Smith. "Efficient pattern recognition and classification." International Journal of Pattern Recognition and Artificial Intelligence. 1992.

[**17**] Zeng, H., J. Huang, and Y. Liang. "Combat vehicle classification using machine learning." Proceedings of the 1999 IEEE International Conference on Neural Networks, 1999. IJCNN '99. International Joint Conference on Neural Networks. Vol. 1. IEEE, 1999.

[**18**] Chollet, François. "Xception: A lightweight, depthwise separable convolution network for image classification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[**19**] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556. 2014.

[**20**] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:16.
[**21**] Chollet, François. "Xception: A lightweight, depthwise separable convolution network for image classification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
[**22**] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556. 2014.

Bibliology

1. CNBC. "Satellite Images Show Large Russian Convoy Regrouping Near Ukraine's Capital Kyiv." 11 Mar. 2022. Web. 3 Nov. 2023.

2. Geospatial World. "GEOINT/OSINT Comes Off Age in the Ukraine Conflict." 14 Mar. 2022. Web. 3 Nov. 2023.

3. Amnesty International. "A Guide to How Amnesty Verifies Military Attacks in Ukraine." 10 Mar. 2022. Web. 3 Nov. 2023.

Keywords

Deep Learning, Convolutional Neural Networks (CNNs), Object Detection, Military Vehicles, Satellite Imagery, Ships, Maritime Vessels, Aircrafts, Fighter Planes.

License