Towards devising an effective and reliable means of fish detection and classification through the exploration of various deep learning algorithms.

by

Rafid Farhan
18101231
Ninad Abdur Rahman
18101223
Syeda Sara Ummy Ahsan
18101437

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
January 2022

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Rafid Farhan
18101231

Ninad Abdur Rahman
18101223

Syeda Sara Ummy Ahsan
18101437

Approval

The thesis/project titled "Towards devising an effective and reliable means of fish detection and classification through the exploration of various deep learning algorithms." submitted by

1. Rafid Farhan(18101231)
2. Ninad Abdur Rahman(18101223)
3. Syeda Sara Ummy Ahsan(18101437)

Of Fall, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on August 23, 2015.

Examining Committee:

Supervisor:
(Member)

Ahanaf Hassan Rodoshi
Lecturer
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

Arnisha Khondaker
Lecturer
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

Hereby, we declare that the material is our original work and truthfully reflects our own research and analysis. Furthermore, the paper appropriately credits co-researchers, and all sources are correctly cited. All authors are actively responsible for the work that was conducted.

Abstract

Due to a number of reasons, marine ecosystems change with certain species of fish disappearing while novel species of fishes become a new staple within a given ecosystem, e.g., a lake, river, etc. Monitoring these changes in ecosystems as different species dwindle and swell in number is crucial for marine researchers, fishery owners, and fish species preservation programs. These increase and decrease in numbers indicate changes in environmental conditions that either favours a certain species or does not. In order to study these changes in conditions, it is imperative to firstly detect the changes in the population of species which is where we come in. The challenges for an underwater project range from water pressure, lack of sunlight, different orientations of fish, the motion of aquatic plants, riverbed structures, and the sheer diversity of shapes in different species. Machine learning and image processing technologies can be of significant importance in identifying such underwater fish species. In our research, we decided to use Convolutional Neural Networks (CNN), namely YOLOv4, to detect fish in input image frames. To classify the fish species, we will use a CNN network. The fusion of these networks is proposed in order to achieve a high level of classification accuracy of fish species from small-sized samples. In order to demonstrate the effectiveness of the model, we propose two datasets, namely BDIndigeneousFish and A-Large-Scale-Fish-Dataset is used, which contain a vast range of image data of several species from different habitats. The image data is fed into the Darknet, which identifies and detects the fish pixels in the image frame. Furthermore, these input images are then passed on to CNN for classification.

Keywords: Fish Detection, CNN model for Classification, YOLOv4 for detection, Artificial Intelligence, VGG-16, DenseNet, Xception

Dedication

This dissertation is dedicated to our wonderful parents who have supported us throughout our journey. In addition we would like to dedicate this work to all of the incredible lecturers and professors, we met and learned from while pursuing our degree.

Acknowledgement

Firstly, all praise to Allah for whom our thesis have been completed without any major interruption.
Secondly, to our advisor Ahanaf Hassan Rodoshi and co-advisor Arnisha Khondaker for their support and advice in our work.
Thirdly, to our friends and peers who have inspired us. And finally, to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Recognition of fish species is important when it comes to underwater object detection because of the importance it holds in marine science[5][6]. Observing a variety of fish species in order to understand the behavior patterns is crucial, which gives insights into marine ecological systems. The health of these ecological systems can be understood, and predictions about positive or negative changes can be made using the distribution of species and how they are dispersed in certain regions. In addition, regarding evaluation and monitoring the environmental changes of our planet, information about these ecological systems becomes crucial as it helps in determining biomass levels and geological changes in the undersea world. With the purpose of achieving these, visual classification of fishes is important, which aids in tracking the movement patterns and tendencies in the activities, providing a deeper understanding of the species and the marine environments collectively. Due to these reasons, many computer vision methodologies have been proposed for the classification of fish species in different past studies.

The critical need to manage and safeguard the coastal and open ocean habitats is becoming increasingly apparent. Even though the seas are enormous, their living forms are nonetheless sensitive to the consequences of human activities, even when no harm is intended or anticipated.We require a good biological and ecological understanding of their populations and communities in order to make appropriate use of the living resources available.The necessity to limit fishing to produce the greatest sustainable yield is widely recognized in principle, but translating available data about fish populations into real fishing quotas is challenging in practice. The way that species interact in the water is poorly known, and the conditions are exceedingly changeable. More information on migratory patterns, breeding behavior, and other topics is still needed. Marine scientists are investigating these issues, with the goal of creating computer models of marine ecology in some cases.
The World's Forgotten Fishes [2], a report by 16 worldwide conservation organizations, stated that global freshwater fish populations were in freefall. Pollution, overfishing and damaging fishing methods, the introduction of invasive non-native species, climate change, and the modification of river ecologies are just a few of the causes. Since 1970, migratory freshwater fish populations have declined by 76 percent [2], and huge fish weighing more than 30 kilograms have mostly disappeared

from most rivers. Last year, 16 freshwater fish species were declared extinct, bringing the worldwide population of megafish down by 94%. Freshwater environments were losing biodiversity at double the rate of oceans and forests. Freshwater fish numbers around 18,000 species, with more being identified all the time.[2]

Researchers predicted the oceans will be empty of fish by 2048 analyzing several kinds of data. Research says the maritime ecosystem will be unable to support our way of life if biodiversity continues to dwindle. In fact, it may not be capable of supporting our life. Already, the number of edible fish and seafood species has dropped by 90% [3]. Toxins in the water are filtered out by ocean creatures. They guard the coastline. They also lower the likelihood of algal outbreaks like red tide. The extinction of these species is a threat to life itself. Continually checking fish species and identifying unexpected species in a given ecosystem that might turn out to be an invasive species capable of wreaking havoc on a habitat and its inhabitants might be of help in reducing extinction of our resources.

Detection of an unknown species can be challenging due to the environment. The purpose of this paper is to provide a new model using which unknown species of an environment can be detected effectively. New species are likely to cause changes in the environment, which might affect the ecological system as well as interactions within the existing population. In light of these circumstances, we developed a highly accurate and dependable fish detection and classification system that detects and classifies different fish species known to a habitat while also predicting novel species.

In this paper, we will be taking a different approach. We would use the version of YOLOv4, which is pre-trained on ImageNet for detection and after that explore CNN algorithms, for classification .Once a subframe containing fish is detected, the frame would be passed on to a classification component. The training process has been divided into two steps which are widely known as transfer learning [13]. In the pre-training stage, we would train the network on 2 different datasets and then use the learned weights to train the newly accumulated dataset containing images of fishes common to that specific environment. The process is called post-training. In this process, fish detection training would be completely independent of the classification training. The objects require resizing into appropriate input size which has been done using OpenCV in our paper. As far as we know, the approach has not been applied in the previous studies.

1.2 Research Objective

One of the essential precursors of the management of fisheries and conservation of fish species in specific aquatic habitats is the possession of detailed knowledge of the diversity and distributions of species in addition to the habitat requirements of the species. However, due to a number of factors that contribute to the changes in environmental conditions, certain fish species may dwindle in number or disappear entirely from previously flourishing habitats. Environmental changes can range from the change of pH levels of the water, pollution or the introduction of a new species of fish or plant, etc. In order to monitor and study these changes, ichthyologists can

make use of a fish detection and classification system that can accurately classify known species and predict an uncommon/novel species to that specific habitat as unknown, providing which is the primary objective of our research.

Thus our main objective is to develop a fish detection and classification system using state-of-art models in order to achieve a high level of accuracy in the aforementioned tasks. Our primary task revolves around training our model to detect and classify fish species that are present and common within a specific habitat. Furthermore, after training, our model is expected to predict species that are foreign, novel, and/or uncommon within the selected habitat as unknown. In terms of the model, our initial target is to implement multiple CNN algorithms to accomplish the same task of classification. The reasoning behind this aspect of our project is to compare and contrast the effectiveness that we will be able to achieve through our implementation of these various algorithms. In summary, our research objective revolves around the following points:

- Provide ichthyologists, fishery owners, and fish species preservation organizations with an accurate and reliable means of detecting and classifying fish species in order to detect changes in numbers of said species and further study the changes in the environment that, in turn are bringing about these changes in the habitats.
- Detect and classify common fish species within a particular environment/habitat in addition to classifying any novel/uncommon species as unknown, which will be set aside for manual inspection.
- Make use of multiple state-of-art deep learning algorithms such as VGG-16, DenseNet-121, etc. in our classification task, conduct an in-depth study of the architectures and provide an in-depth analysis of these algorithms.
- Perform a comparative analysis between the algorithms used and provide accurate reasoning for which algorithms work best in our context.
- Acquire a novel dataset containing a large number of fish images of common species from a selected lake/river in Bangladesh. The formation of a quality dataset for future researchers is one of our primary objectives, as quality datasets in a Bangladeshi context are hard to come by.
- Evaluate the outcome of our research with the added perspective of an online acquired dataset collected from outside Bangladesh in addition to the dataset we collected ourselves.
- Provide a short evaluation of our results in comparison to pre-existing results from previously conducted research that are similar in nature.

1.3 Thesis Outline

As discussed, our main objective is to construct an effective system for classifying different fish species in a given environment and aim for the highest possible

accuracy through the comparison of different CNN algorithms used for the same classification task.

Thus in the first chapter (Chapter 1), we start with an overview detailing the importance of the research and the motivation behind pursuing the research. Moreover, we emphasize the Bangladeshi context/perspective for our research as we feel that research projects on these areas are less common in our country.

Chapter 2 deals with the related literature and works previously done by other researchers on similar topics. Detailed reviews with the indication of significant results that were achieved in these research works are discussed, along with certain areas of improvement.

The third chapter (Chapter 3) details the neural networks that are in use with a comprehensive explanation of the architectures, the layers, and associated components, along with a background analysis on CNNs in general.

Chapter 4 details specifics about the dataset that is in use along with the acquisition process. Moreover, this chapter describes the preprocessing techniques that were implemented in order to make the data more suited for use in the project. In addition, the splitting of the data is also detailed in this section.

In Chapter 5, a comprehensive implementation process of the different CNN algorithms that are in use is discussed with the input setup and the training parameters used in the process of model creation. In addition, graphical representations of accuracy and associated data are also detailed in this chapter.

The sixth chapter (Chapter 6) is associated with the results of the research project with comparisons of epochs and overall results acquired. Graphical representations include confusion matrices and bar charts of different models that were used.

Finally, the last chapter (Chapter 7) concludes with limitations and challenges that we as researchers had to deal with, along with prospects for any further research venture on top of the existing work.

Chapter 2

Related Work

Literature Review

This section is dedicated to critically review any previously conducted research-work in fish detection and classification through the use of Deep learning and image processing. Here we analyze different techniques, CNN architectures and datasets that have been used in order to achieve the results that previous researchers were able to achieve.

Various studies have been conducted by different researchers related to the classification and identification of fish species using several deep learning techniques in previous years. Some of these are delineated in this section. Dhruv Rathi et al. [10], using the Fish4Knowledge dataset, implemented Otsu's Thresholding and some other classifiers in order to remove noise and identify a sure foreground. Identification of fish species is a multi-class classification problem that is an interesting machine learning and computer vision research subject. cutting -edge algorithms that accomplish classification mostly by extracting and matching shape and texture features from individual input pictures. Convolutional Neural Networks are used in their suggested technique, which makes the process easier and more resilient even when working with big datasets. They conducted the classification by utilizing Gaussian Blurring, Morphological Operations, Otsu's Thresholding, and Pyramid Mean Shifting to pre-process the pictures. The system's first step was to filter out the noise from the dataset. Prior to the training stage, image processing is used to eliminate underwater obstructions, mud, and non-fish bodies from the pictures. For the categorization of fish species, the second phase employs a Deep Learning technique, including the construction of Convolutional Neural Networks (CNN). Thresholding (Otsu) works with a threshold value and creates a grey level histogram from a grayscale picture. When a grayscale pixel exceeds the threshold value , it is regarded as white, and vice versa. This ensures the foreground fish is in focus. Subsequently, using Erosion and Dilation and implementing Morphological operations, after the noise reduction process, the image's split sections are reunited. In order to train the CNN in a reliable and error minimizing manner, Dhruv Rathi et al. [10] used activation functions such as Rectified Linear Unit (ReLU), Softmax and tanh. The CNN that Dhruv Rathi et al. [10] used takes 100x100x3 original RGB coloured image stacked with 100x100x1 image which came from the preprocessing step. So the final input that the input layer takes in is 100x100x4. A mask of 5x5x4

mask performs convolution over this 100x100x4 input feature map. When 32 of these 5x5x4 filters are applied, the result is a 96x96x32 matrix. The map is then shrunk to the same size as the convolutional layer, in terms of number of planes. In total, Dhruv Rathi et al. [10] used 27,142 images to train the CNN and achieved an accuracy of 96.29% and a computational time of 0.00183s per frame using the ReLU activation function; in the case of Softmax and tanh, the accuracy percentages are 61.91 and 72.62 percent respectively.

Xiu Li et al. [4] used a Region Proposal Network (RPN) to generate proposals as part of detection rather than in pre-processing. A pre-trained Zeiler and Fergus (ZF) model with five convolutional layers and three fully-connected layers was used on a dataset Fish Task from ImageCLEF. Training and modification on this network were conducted subsequently. A multi-task loss function was used for jointly training softmax classification and bounding-box regression for particular region proposals. In terms of the outcome, the final fully connected layer was divided into a couple of layers which had outputs of softmax probabilities of 12 fish classes, while the other layer had given the output of the bounding box values. With the aim of speeding up the detection and recognition process, Xiu Li et al. [4] shared convolutional layers between proposal generation and the detection network. Xiu Li et al. [4] was able to achieve an average precision of 82.7% with an average detection time of 0.102s per image.

B Vikram Deep et al. [12] suggested a hybrid convolutional neural network architecture that uses CNN for feature extraction as well as SVM and K-Nearest Neighbor for classification. The Fish4Knowledge dataset, which comprises a sample of 23 different species, was used to train both frameworks. Laplacian kernelwas used to extract sharp edges of an object. Because the laplacian kernel produces a binary image, a modified kernel is employed to produce a coloured image, which is then sent onto a framework that consists of three convolution layers, each followed by a max-pooling layer. These layers are furthermore connected by two fully connected layers. The ability to detect edges is the key reason for adopting max pooling. 32 3x3 kernel size filters make up the first convolution layer. A total of 64 3x3 filters make up the second convolution layer, and finally, the third convolution layer has 128 filters with a kernel size of 3x3. And the following pooling layer has a kernel size of 2x2. In order to avoid overfitting, before a fully connected layer, an extra dropout layer is used. The final fully connected layer is in charge of defining a species. The suggested framework was tested in Python with the Keras framework and the Tensorflow backend. In a convolutional neural network, there is no established rule for calculating the number of filters and kernels. Depending on the task, they are discovered by trial and error. The suggested framework has discovered that 32, 64, and 128 filters produce satisfactory results for the configuration. In this framework, Rectified Linear Unit(ReLU) works as an activation function; on top of that, an additional activation function, namely Softmax in the last fully connected layer, is used. B Vikram Deep et al. [12] proposed a framework using a Hybrid convolutional neural network, in which feature extraction is done using The DeepCNN architecture and to classify the species SVM or k-NN classifier is used. Among the proposed frameworks, DeepCNN-KNN by far produces the best result with an accuracy of 98.79%.

Xin Sun et al. [5] proposes a solution to a common problem in image processing, i.e. improving the quality of sample images. Observing deep sea levels often is deemed to be difficult due to the environment and the low-quality images we acquire from deep-sea levels. A method that can work with improving the quality of these pictures using deep learning techniques and super-resolution methods is highly appreciated. Using deep learning techniques and super-resolution, this study provides a system for explicitly learning discriminative features from very low-resolution pictures. Because interesting spots are difficult to discover in low-contrast and low-resolution pictures, cutting-edge deep learning algorithms such as PCANet and Network In Network (NIN) enabled them to generate abstract discriminative features from the image. Furthermore, to address the low-resolution issue of underwater pictures, they used a single-image super-resolution technique to generate high-resolution images from low-resolution ones. High-quality, high-resolution pictures are created from low-resolution photographs during the super-resolution step. In this paper, they presented a self-training technique for training the super-resolution method. Two deep models are used to extract features in the recognition model learning stage, and the linear SVM is used as the classification model. They used a quick, direct super-resolution to solve the low-resolution problem of underwater pictures (FDSR). For each subspace, enhance predictability functions are learned using the FDSR method, using training picture patches features. Since PCANet requires that all input photos be the constant dimensions, we must convert the images to a fixed size before training. First, they use ImageNet to train the NIN, then fine-tune the FishCLEF 2015 dataset and using single-image super-resolution, they convert it to high-resolution pictures. Finally, to predict the labels of fish, a linear SVM classifier is trained. The process has yielded 77.27% and 69.84% accuracy for PCANet and NIN.

Katy Blanc et al. [1] have created a process chain to automate fish identification systems. They use the dataset LifeCLEF2014 and organize it into 4 subtasks. They consider 3 key points that describe how the bounding boxes are scaled. If the bounding boxes are not according to scale, they can be further readjusted. Firstly, motion is detected and differentiated from the background. A linear SVM classifier is trained for every species to identify species from every frame using bounding box key points. Katy Blanc et al. [1] used an adaptive background mixture model, which consists of the assumption that each pixel in a scene is modelled by a mixture of many Gaussian distributions. Background subtraction is performed, which forms a mask in the places where motion is detected. The masked output is firstly eroded and then dilated to define the detected moving objects. The SVM classifier is trained with filtered keypoint descriptors. The descriptors for the species are fed as positive samples, while the background descriptors are fed as negative examples. The proportion of negative to positive descriptors is similar. The results of this are certainly good but can be improved a lot more by improving their process chain and finding a better tracer for fishes.

Kristian Muri Knausgard et al. [18] proposed a method to detect and classify objects captured on film without pre-processing or filtering the image beforehand. Detecting and classifying temperate fishes due to noise and variations in light levels, as

well as the surrounding environment, might be difficult. In this paper, a method has been proposed in which there would be 2 modules in order to detect and classify species. For the detection, Kristian Muri Knausgard et al. [18] used YOLO and for classification; they combined CNN and SE-Block without pre-filtering. The training samples for temperate fish being limited, the YOLO model had been trained on ImageNet and the fish classifier on Fish4Knowledge. They obtained a new dataset of temperate fish species for this study's training samples in the classification phase because the Fish4Knowledge dataset is limited to tropical fish species. The network was pre-trained to learn general features of fish using the Fish4Knowledge dataset, which was the method they used for classification. Following that, the learnt weights were utilized as a foundation for post-training on a freshly obtained dataset. Images of temperate fish species were included. This system requires no pre-processing of data and in this system, object detection takes a (live) video feed as input and produces fish items that haven't been classified. Six hundred nineteen pictures with a total of 1943 fish that have been meticulously annotated have been used in this detection method as a training dataset. They gathered video data in a variety of sites with depths ranging from 1 to 40 meters also in different seasons. Two datasets have been used for classification: the Fish4knowledge and another temperate fish dataset. Fish4knowledge contained 27320 images with 23 separate species. The temperate dataset, on the other hand, was a compilation of pictures from a variety of sources. Video footage captured with GoPro cameras (HERO4-7+Black) were received from three distinct sources. The cameras were installed at Austevoll, western Norway, at a depth of 2-5 meters near tiny reef areas utilized as breeding grounds for a variety of wrasse fishes, labelled according to their sex. The training is performed using a batch size of 64 and 8 subdivisions. The model archived a precision of 99.27% with the pre-training. Both the image classification and the detection are updated after training to incorporate temperate fish of concern. Pre-training weights are automatically added to post-training. The post-training model receives an accuracy of 83.68% with data augmentation and 87.74%. Therefore we can conclude by saying that the model works on a larger dataset.

Yanling Han et al. [16] conducted a study to classify sea ice combining 3 classification modules which proved to generate a better outcome compared to other classification modules. Since most traditional methods focus on either spectral or spatial information in this study, they tried focusing on both. Yanling Han et al. [16] uses 3D-CNN in order to exploit special spectrum features and combined SE-block so as to distinguish contributions of spectra. Moreover, to achieve higher accuracy on all kinds of samples SVM classifier is used. The experiment is conducted on three different data Baffin Bay, Bohai Bay, and Liaodong Bay. Even though studies show remote sensing technology is able to provide data fast and efficiently, it consists of tens of thousands of bands that need to be differentiated. Furthermore, difficult environments make it difficult to acquire labelled data in remote sensing technology. As a solution to classifying these remote sensing images, researchers applied machine learning algorithms such as CNN and SVM. Also The central notion of SE block is that the network learns feature weights using a loss function, providing better classification results. In this paper, it is shown that the value of each neuron (x,y,z) of the 3D-CNN to be,

$$v_{ij}^{xyz} = g\left(\sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)} + b_i j\right), \tag{2.1}$$

On top of that, to reach an optimum value, the RelU function is used. This paper uses CNN for model training, and the sea ice classification is done by the SVM, a shallow learning algorithm. Unfortunately, being a shallow learning algorithm, it's hazardous for the SVM algorithm to extract and classify features with its limited computational unit. Because deep learning can readily extract hidden features, it's a good idea to use it instead of the shallow learning algorithms.

Chapter 3

Background Analysis

This section is dedicated to the exploration and analysis of different models from an architectural and theoretical perspective. Consists of sections dedicated towards several networks along with their important features.

3.1 CNN (Convolutional Neural networks)

With the emergence of the Artificial Neural Network (ANN), the discipline of machine learning has taken a radical turn in recent years [6]. ANNs are regarded as one of the most powerful technologies since they can manage a large quantity of data. The need for deeper hidden layers has lately begun to outperform traditional approaches in several disciplines, particularly pattern recognition. The Convolutional Neural Network (CNN) is a class of ANN generally employed for solving computer vision tasks.
CNNs are designed with the assumption that the input will be pictures. This concentrates the architecture such that it is best suited to dealing with the specific type of data. One of the significant differences is that the layers inside the CNN are made up of neurons that are organized in three dimensions: the input's spatial dimensionality (height and breadth) and depth. The depth of an activation volume is the third dimension, not the total number of layers within the ANN. Each layer's neurons will only bind to a small portion of the layer before it. As previously stated, CNN is a deep learning model for processing data with a grid pattern, such as photographs, aimed to learn spatial hierarchies of information, from low to high-level patterns, automatically and adaptively. Convolution, pooling, and fully connected layers are the three types of layers that constitute a CNN [11]. A construct of the network comprises numerous convolution layers and a pooling layer, repeated several times, followed by one or more fully connected layers. The first two layers, convolution and pooling, extract information, while the third, a fully connected layer, transfers those features into the final output. A convolution layer is an essential component of CNN, which is made up of a series of mathematical operations, including convolution and a specific sort of linear operation. Because a feature can appear anywhere in a digital picture, pixel values are stored in a two-dimensional (2D) grid, and a tiny grid of parameters termed kernel, an optimizable feature extractor, is applied at each image point; CNNs are particularly efficient for image processing. Extracted features can evolve hierarchically and progressively more complicated as one layer feeds its out-

put into the next layer. Training is the process of adjusting parameters like kernels in order to reduce the discrepancy between outputs and ground truth labels using optimization algorithms like backpropagation and gradient descent, among others.

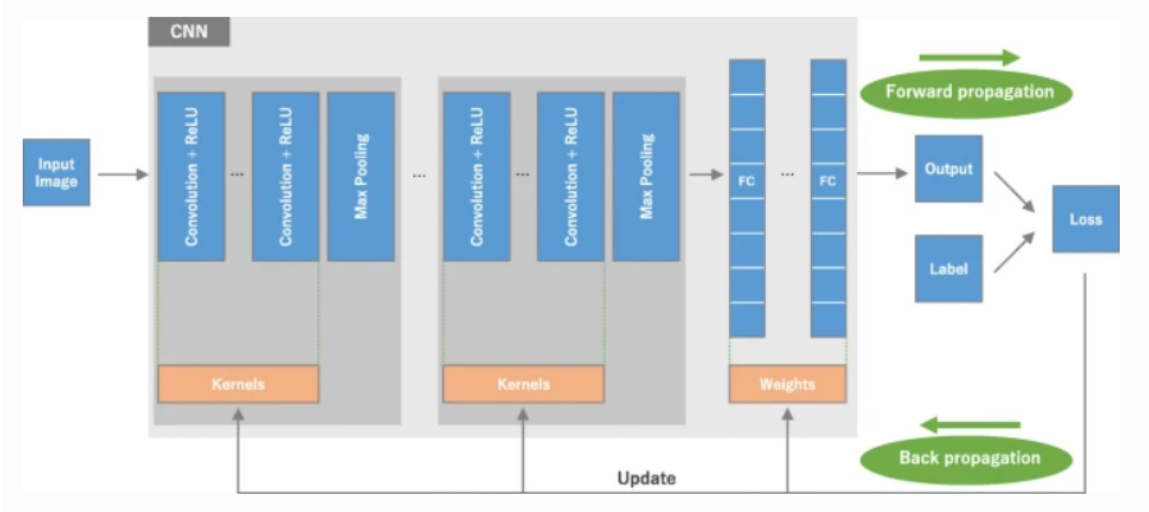


Figure 3.1: CNN

The learnable kernels are the focus of the convolution layer's parameters. The spatial dimensionality of these kernels is generally low, yet they extend throughout the whole depth of the input. When data passes through a convolutional layer, it convolves each filter across the input's spatial dimensions to form a 2D activation map [3]. The scalar product is computed for each value in that kernel as we progress through the input. Every kernel will have its own activation map, which will be layered along the depth dimension to generate the convolutional layer's whole output volume. Convolutional layers can also greatly reduce the model's complexity by optimizing its output. The depth, stride, and establishing zero-padding are three hyperparameters that are used. The depth of the output volume and stride can both be set ahead of time. Setting the stride to a higher value reduces overlapping and generates a smaller output. To ensure uniformity for simplicity of calculation zero-padding is utilized to regulate the size of the output. A pooling layer performs a standard downsizing operation on the feature maps, lowering their in-plane dimensionality and reducing the number of learnable parameters. It is worth noting that none of the pooling layers has learnable parameters, although filter size, stride, and padding are hyperparameters in pooling operations, much like convolution operations. The pooling layer adjusts the dimensionality of each activation map in the input using the MAX function. These are typically max-pooling layers with dimensionality of 2 by 2 kernels. Because of the destructive nature of pooling, having a kernel size greater than 3 will typically result in a significant reduction in model performance. Beyond max-pooling, CNN designs may also include general pooling. A set of fully-connected layers map the features retrieved by the convolution layers and downsized by the pooling layers to the network's final outputs. Instead of focusing on the entire issue domain, Convolutional Neural Networks use information about a single type of input. This enables a far simpler network architecture to be built up, which has shown incredible results in a variety of fields in previous studies.

3.2 VGG 16

The VGG-16 architecture was developed and thus named after the Visual Geometry Group, which is based in Oxford, and the results of the VGG-16 architecture on the ImageNet dataset are very impressive [2].VGG architecture uses 3x3 kernel-sized filters instead of larger variations. VGG architecture employs either 16 or 19 layers hence the two variations being VGG-16 and VGG-19 [2]. We decided to choose VGG-16 architecture due to the fact that VGG-19 is deeper, thus requiring higher computational resources.

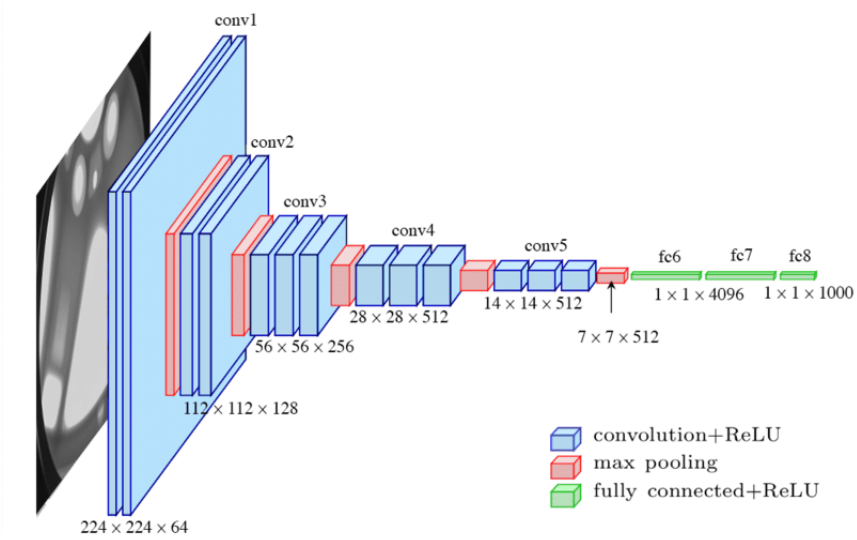


Figure 3.2: Model Architecture [8]

VGGNets are built on the foundations of convolutional neural networks. Small convolutional filters are used to build the VGG network. There are 13 convolutional layers and three fully linked layers in the VGG-16. The tensor that is put in as input is of dimension (224,224,3). VGG's convolutional layers use the smallest feasible receptive field while still capturing up/down and left/right. Additionally, 11 convolution filters operate as a linear transformation of the input. After that, there is a ReLU unit. The rectified linear unit activation function (ReLU) is a piecewise linear function that outputs the input if it is positive and zero otherwise. To maintain spatial resolution after convolution, the convolution stride is set to 1 pixel. The VGG network uses ReLU in every hidden layer. There are three completely linked layers in the VGGNet.The first two levels each contain 4096 channels, whereas the third layer has 1000 channels, one for each class.

This input is passed through Convolutional layers, which have a filter size of 3x3 and a stride value of 1. The number of filters used in the first layer is 64. The padding of all the Convolutional layers is always kept the same in the case of VGG. When it comes to the max-pooling layers, the filter is 2x2 and the stride is 2 and this is consistent for all the max-pooling layers. The entire architecture features this pattern as shown in the diagram above with max pooling, specifically spatial pooling being carried out after a certain number of convolutional layers (two or three). The

output dimensions of the max-pooling layers can be calculated and verified using the formula given as [(n+2p-filter_size)/stride]+1. If we observe the input of the first max-pooling layer and try to calculate the output dimensions, we get [(224+2*0-2)/2]+1 which equals 111+1=112, which is what we observe as the dimension for the subsequent two convolutional layers from the diagram. The filter size is also increased from 64 to 128, which is why we get 112x112x128. The final three layers out of the sixteen weighted layers just before the output layer are fully connected layers. In addition, all these hidden layers use rectification and non-linearity. The final layer is a softmax layer with a parameter value of the number of classes that need to be classified.

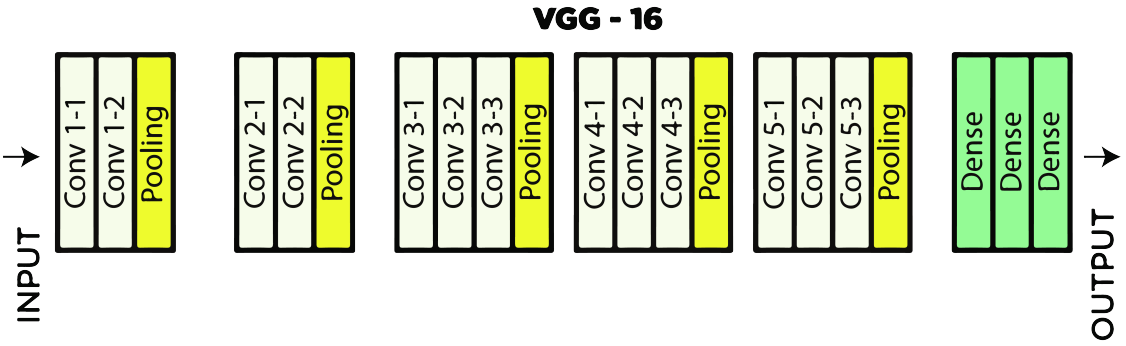


Figure 3.3: Layers of VGG-16

In conclusion we decided to use the VGG architecture even though it is slightly bigger in size due to the accuracy that it is known to be able to generate and how consistent the architecture of the model is.

3.3 Xception

The usual Inception module initially looks at cross channel correlations using a series of 1x1 convolutions. The input data is further mapped into 3 or 4 different smaller input data, and then all the correlations are mapped into these smaller 3D data using regular 3x3 or 5x5 convolutions. Although Inception modules are related to convolution layers, they experimentally appear to be capable of learning richer representations with fewer parameters [7]. A canonical inception V3 model has been illustrated in Figure 3.8.

A simple Inception module having one size of convolution (e.g. 3x3) and no average pooling tower (Figure 3.9).

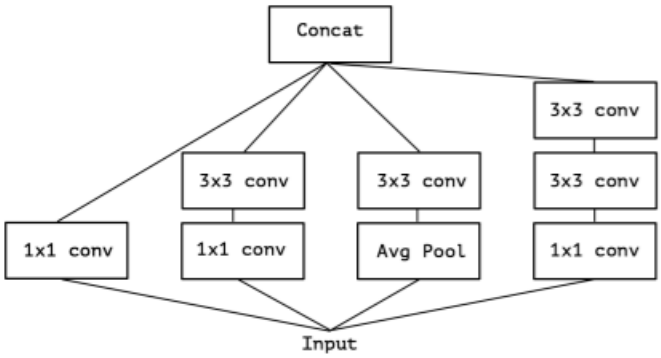


Figure 3.4: Inception Module (canonical) [7]

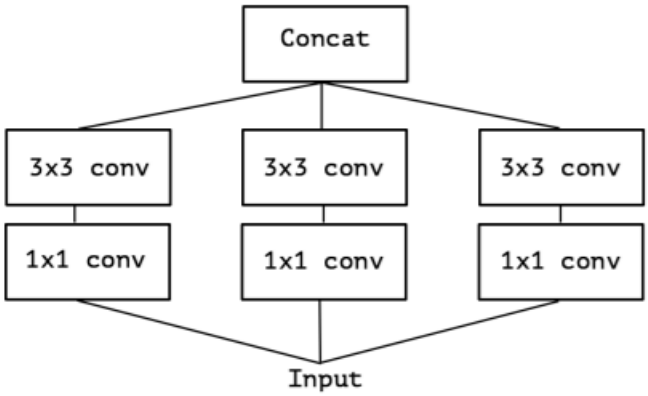


Figure 3.5: Inception module (simplified) [7]

This network can be further reduced to have a big 1x1 convolution followed by spatial convolutions on non overlapping outputs (Figure 3.10)

An ”extreme” version of an Inception module has been further introduced, which would first map cross-channel correlations utilizing a 1x1 convolution; the spatial correlations of each output channel should then be mapped independently. This model has been named the Xception model. In this module, Inception modules have been replaced with depth-wise separable convolutions having the same number of parameters as Inception V3. The architecture consists of 36 convolutional layers, which are further structured into 14 modules. Other than the first and last modules, the rest of the modules have residual connections around them. In conclusion, a linear stack of depth-wise separable convolution layers with residual connections makes up the Xception architecture. [7]

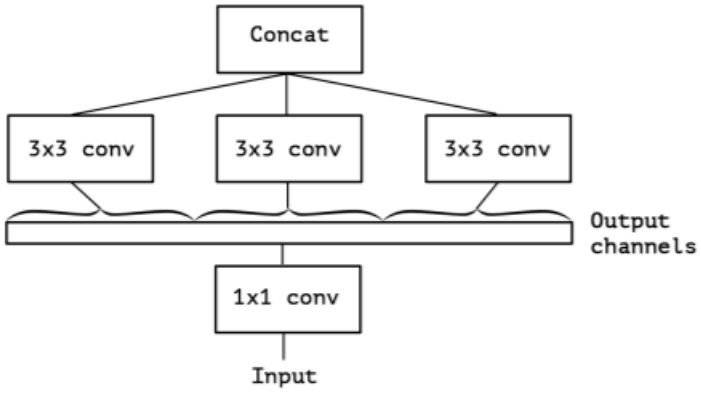


Figure 3.6: Reformulation (simplified) [7]

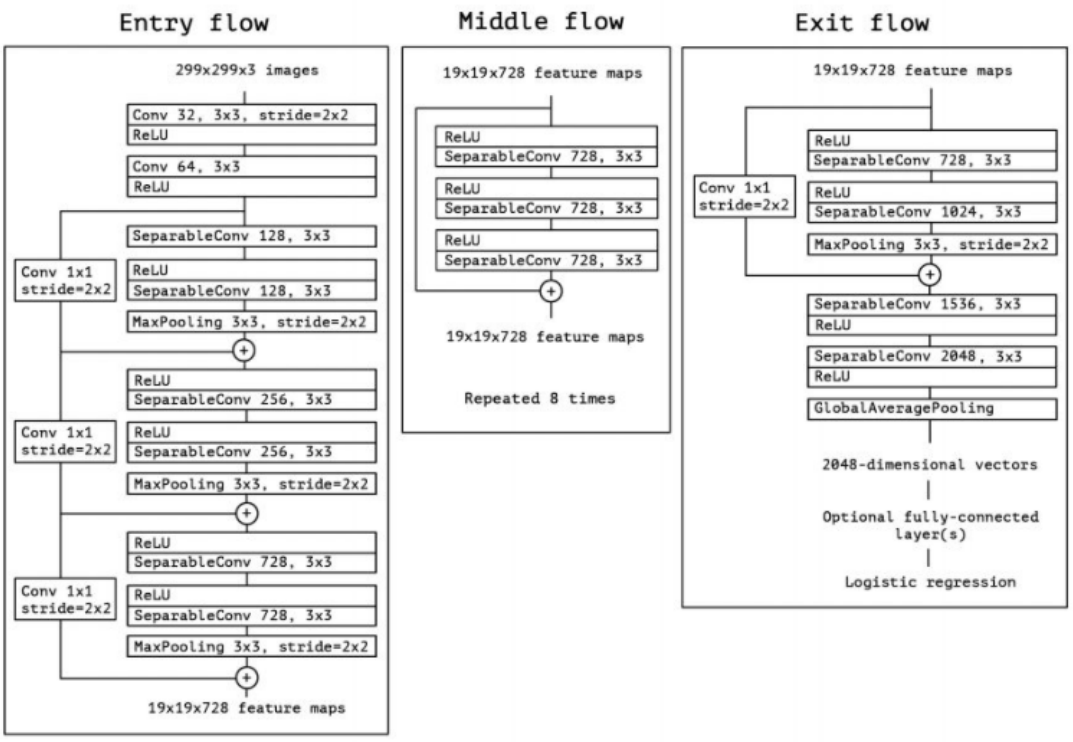


Figure 3.7: Architecture of Inception [7]

3.4 DenseNet-121

The authors of the DenseNet architecture argue that the technique of summation of the identity function and the output of a layer used in traditional ResNet architectures may cause obstruction in the information flow in the network. In simpler terms, DenseNet architectures propose a much more extreme system of shortcut connections that connects all the layers to each other. The number of connections can be calculated with the equation L*(L+1)/2, where L is the number of layers in the network instead of L layers having L direct connections between each and the next, as we see in ResNet. The DenseNet architecture utilizes the concept of fea-

ture reuse through concatenation instead of the summation observed in the ResNet architectures. The DenseNet architecture alleviates the vanishing gradient problem through the use of shortcut connections (each layer connected to every other layer) which makes gradient flow easier because of more robust feature propagation. [9]

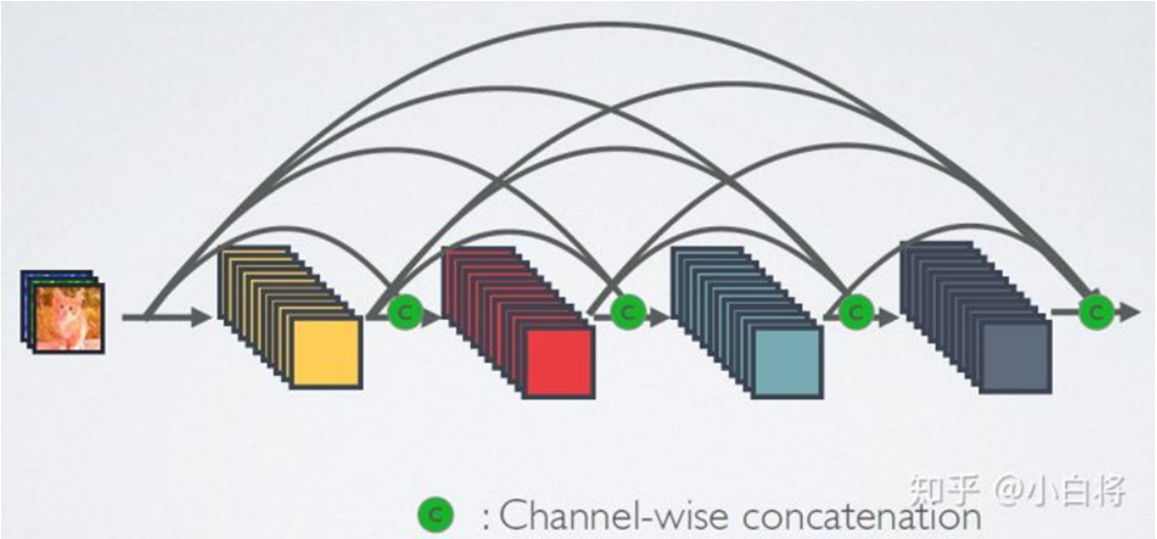


Figure 3.8: Concatenation of features in DenseNet [17]

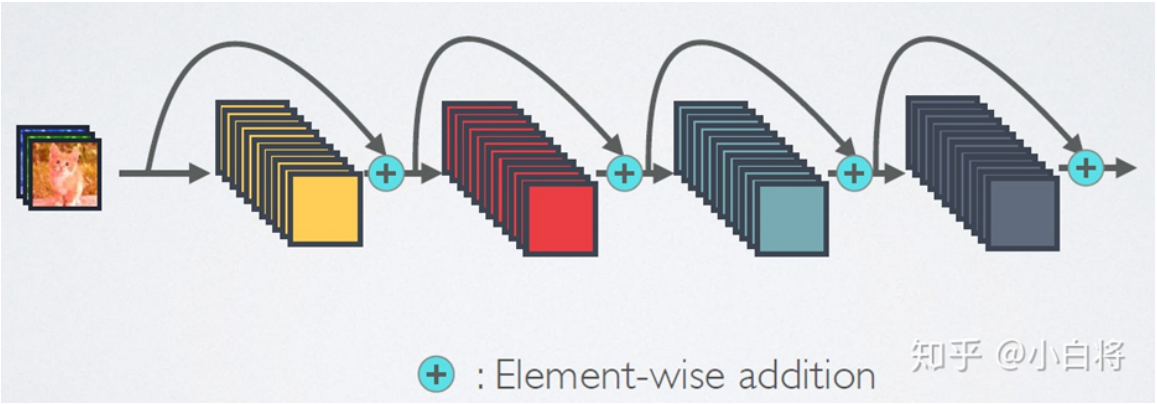


Figure 3.9: Addition of features in ResNet [17]

Due to the concatenation of feature maps from the previous layers, the next layer can work with the feature maps of the earlier layers and the feature maps generated from the convolution operation after the last layer. This process is known as feature reuse, where features from the previous layers are preserved and taken into account in the current layer, which allows DenseNets to increase variation in the input of subsequent layers and improve the overall efficiency. The difference between addition and concatenation is precisely here. In ResNet, addition takes place, which correlates the features, whereas in DenseNet, the concatenation of features provides diversified features. However, due to this process of concatenation and the use of such a large number of convolutional layers, DenseNet becomes computationally costly and takes a long time to run, which gives rise to the question of whether it outperforms other networks significantly.

DenseNet architecture consists of a series of dense blocks followed by transition blocks as detailed in the diagram below where it shows a DenseNet with 3 Dense blocks :

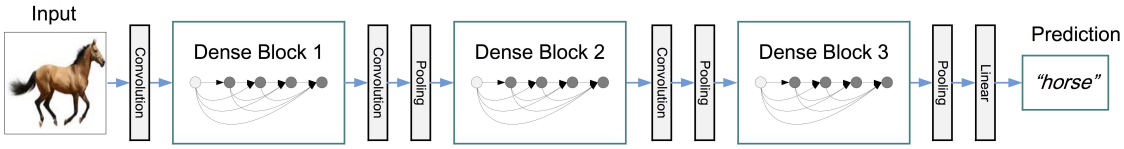


Figure 3.10: Dense blocks with transition blocks in between [9]

The transition blocks are made up of one convolution layer and one average pooling layer, and the job of this block is to reduce the number of channels, while the Dense blocks are made up of one 1x1 followed by a 3x3 layer, and these 2 layers are repeated a particular number of times, e.g. 6,12,24. The 1x1 layer works as a bottleneck layer which is used to improve the speed and efficiency as the number of inputs can get high, particularly for the further layers. As the input passes through the dense blocks, the size of the feature map grows, with each block adding K features over the existing features. This K is known as the growth rate, which determines the number of feature maps added each time. The diagram below shows the DenseNet architectures:

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 3.11: Different DenseNet Architectures [9]

In conclusion, DenseNet-121 is a compact, modern state-of-the-art model made up of 120 convolutional layers and 4 average pooling layers that provides a high level of accuracy which is why we chose to work with this architecture.

3.5 YOLO

We use YOLOv4 with a purpose of increasing processing speed and parallelizing computing in order to detect the fishes in the frames, and this detection process

is independent of the classification/recognition part. Backbone network CSPDarknet53, space pyramid pooling SSP, Path aggregation Network PANet, and YOLOV3 are all part of the method. PANet is chosen by YOLOv4 for network feature aggregation. After CSPDarknet53, YOLOv4 adds an SPP block to expand the receptive area and isolate the most significant elements from the backbone. Then, to boost accuracy and subsequent item recognition, two ubiquitous talents were introduced: bag-of-freebies (BoF) and bag-of-specials(BoS). BoF Technique is implemented for improving model accuracy while lowering inference costs (computation or inference times). BoS are techniques that enhance accuracy while raising the cost of inference marginally. These strategies are often implemented as plugin modules, which may be added or removed from the model at any moment. Non-max suppression, nonlinear activation, and spatial attention modules are some examples. [23]
The feature-extraction architecture is referred to as the backbone. In the YOLOv4 backbone, a Dense Block has numerous convolution layers, each of which is made up of batch normalization, ReLU, and convolution outputting four feature maps. As a result, the number of feature maps at each layer is raised by four.

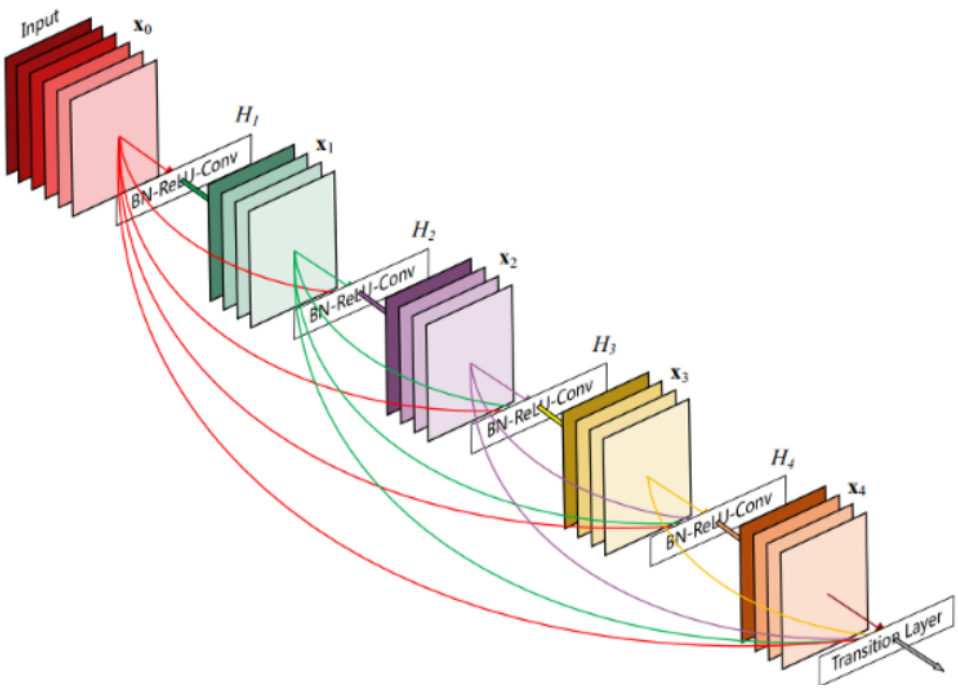


Figure 3.12: [9]

Then, by combining many Dense Blocks with a transition layer made up of convolution and pooling, a DenseNet may be created. Cross-Stage-Partial connections are abbreviated as CSP. The feature map of the base layer is split into two parts by CSPNet. A dense block and a transition layer will be applied to one section. To proceed to the next level, the second portion will be integrated with a sent feature map. An illustration is shown below.

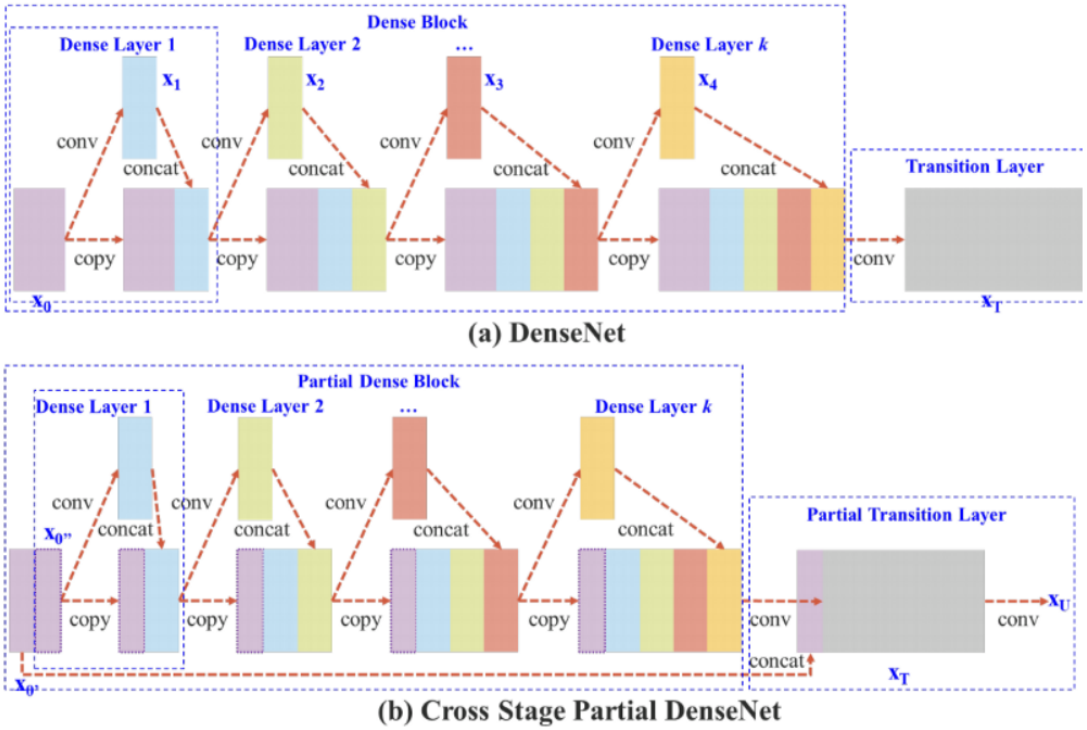


Figure 3.13: [21]

The neck block's aim is to build layers between the backbone and the head (dense prediction block). Before being fed into the head, surrounding feature maps from the bottom-up and top-down streams are element-wise joined together or concatenated to augment the information. As a result, the input to the head will include both spatially rich data from the bottom-up stream and semantically rich data from the top-down stream. The following picture shows how different feature maps from other layers are used:

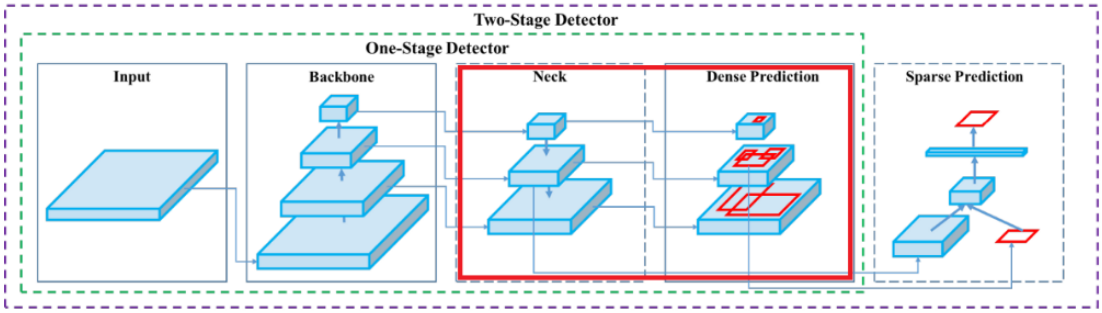


Figure 3.14: [15]

FPN up samples (2*) the preceding top-down stream and adds it to the bottom-up stream's adjoining layer when formulating predictions for a certain scale. With improved SAM, PAN, and SPP, the FPN idea is gradually applied in YOLOv4. To construct two sets of feature maps in SAM, full pool and average pool are applied individually to input feature maps. To make spatial attention, the results are put into a convolution layer followed by a sigmoid function. To produce finer feature maps, this spatial attention module is applied to the input feature. However, unlike the

original SAM implementation, YOLOv4 uses a modified SAM that does not employ maximum and average pooling. A modified version of the PANet is another technology employed (Path Aggregation Network). The objective is to gather information in order to improve accuracy.To keep the output spatial dimension in YOLOv4, a modified SPP is employed finally. A sliding kernel of size 1x1, 5x5, 9x9, or 13x13 is subjected to a maximum pool. There is no loss of spatial dimension. The result is created by concatenating the features maps from various kernel sizes. In the head section the bounding box coordinates (x,y,w,h) and the confidence score for a class are detected by the network. The purpose of YOLO is to partition a picture into a grid of many cells and then use anchor boxes to forecast the likelihood of each cell holding an object. A vector containing bounding box coordinates and probability classes is returned. Finally, post-processing methods such as non-maxima suppression are employed.Overall structure of YOLOv4 is given below:

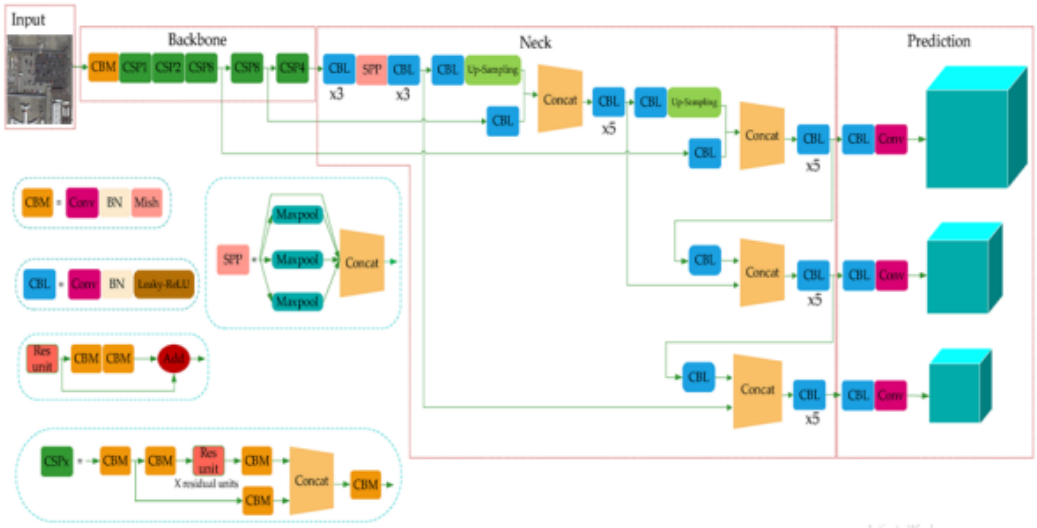


Figure 3.15: [26]

Chapter 4

Dataset Analysis

This chapter deals with the dataset related information where we perform an in-depth discussion about data collection, preprocessing, and splitting of the dataset according to our needs.

4.1 Dataset Acquisition

Data acquisition refers to mainly the collection of data in order to come up with a dataset suitable for the need of a model training experiment. The best way to do this is to fold your sleeves, get your hands dirty and collect the data through surveys, or in the case when experimenters need an image-based dataset (our case), by capturing a sufficient number of high-quality images of the object in question. Our initial plan was to collect the image data ourselves by surveying different fish markets, large-scale fish hatcheries, and fish research facilities. However, due to several difficulties regarding the current pandemic situation (Covid-19) and other challenges, a full-scale survey/data collection was not feasible. Thus under the circumstances, we opted to select two reliable datasets, namely BDIndigenousFish [22] and A-Large-Scale-Fish-Dataset [20], extracted from their official sources [13] [19]. BDIndigenousFish provides a smaller-scale dataset that features 8 species of fishes native to the country where this research work is being conducted, i.e., Bangladesh. BDIndigenousFish dataset comprises 2600 images separated into the 8 different folders for the designated species. The A-Large-Scale-Fish-Dataset, on the other hand, provides a more significant number of images featuring around 9000 images of 9 species of fishes collected from a habitat of Izmir, Turkey. A-Large-Scale-Fish-Dataset, which is considerably larger than the BDIndegenousFish dataset, therefore, provides suitable opportunities for insights and comparisons between the two datasets.

4.2 Data Preprocessing

The next step in our experiment involves preprocessing the collected/selected raw dataset. In order to aid our models in the decision-making process or prediction process, this step plays a crucial role. In other words, preprocessing is done in order to reduce the noise and anomalies which may be present in the dataset and bring it

into a form that assists our algorithms to better do their job.
Due to the circumstances, we could not collect the dataset by our own fieldwork, which is why the first step in preprocessing is to filter out any corrupted jpeg files that may be present in the dataset. This process is necessary due to the fact that we collected the datasets from external online sources, and in most real-world scenarios, corrupted data is a common incident. In order to rid the dataset of any such badly encoded images, we check the JFIF string in the image header. Any image not featuring the JFIF is discarded from the dataset.
The next task is to create a dataframe. The dataset we used consists of images of 8 different species of fish (in the case of BDIndigenousFish) organized into 8 folders with a folder name the same as the name of that species. Thus we created the labels of the fishes with the names of folders that the image is in with the image column containing the path to the image. And in case of the The A-Large-Scale-Fish-Dataset the same task has been done but into 9 folders.
For the purpose of preprocessing and augmentation of the original image data, we used the ImageDataGenerator class. ImageDataGenerator in Keras provides real-time data augmentation with different augmentation techniques like rotation, standardization, flips and shifts, brightness changes, etc. [14]. The series of preprocessing and augmentation techniques that we implement starts with standardizing our image data by firstly setting the image size to 224 x 224, which will be the input size for our models. However, the images still hold their RGB channel values, which range from 0 to 255. This is less than optimal for a model as it is more convenient to take smaller input values. In order to achieve this we standardized the values between 0 and 1 by rescaling the image with the parameter value of : rescale= 1./255. Next, we perform random rotations as part of image augmentation by setting the parameter rotation_range. We can set a value between 0 to 360, and this value specifies the degree at which the random rotation occurs. Below are some demonstrations of the before and after.

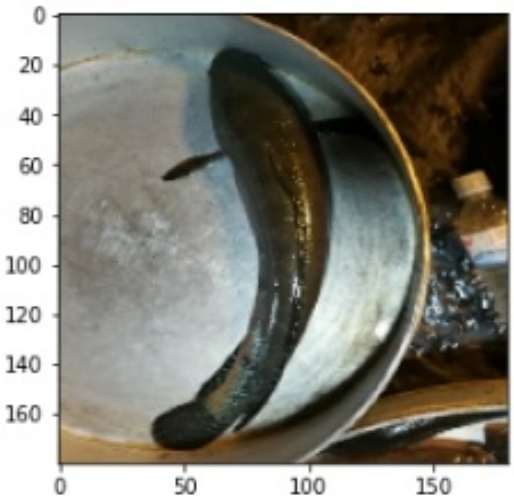


Figure 4.1: Original image

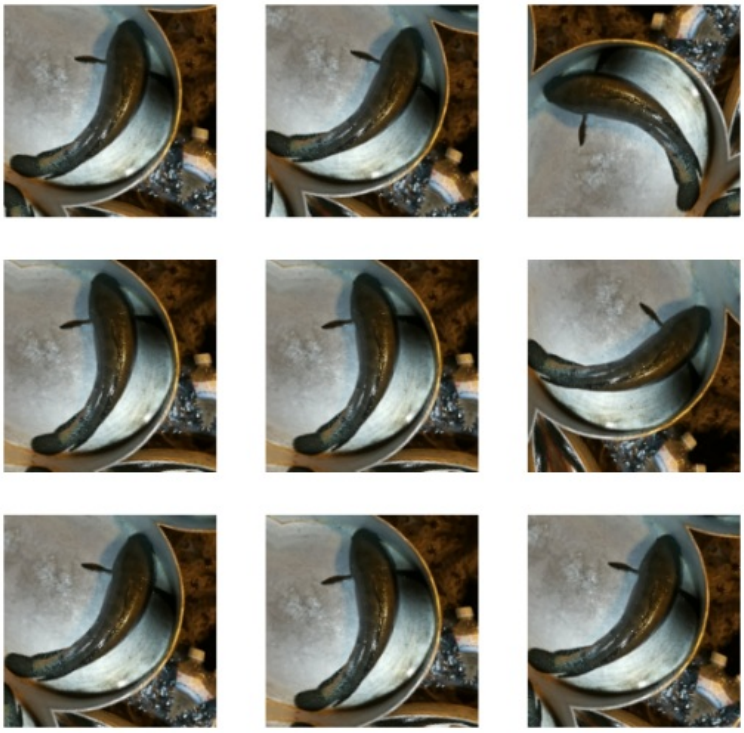


Figure 4.2: Performing random rotations

The following parameters we set are width and height range shifts. Random shifts are done to ensure that the object is in the center or at least provide better positioning of the object in the image and provide more variance in the image inputs. Demonstrations are given below:

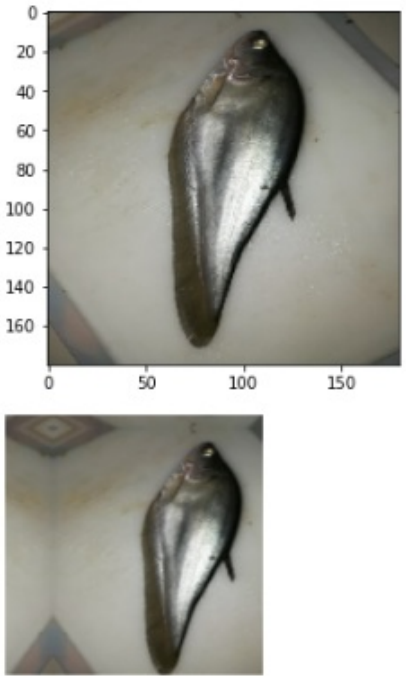


Figure 4.3: Performing random translations

After this, we provide a zoom_range in order to make a random zoom in or zoom out of the image again to provide more variance in the data. In this example, we can observe that a zoom out is performed:

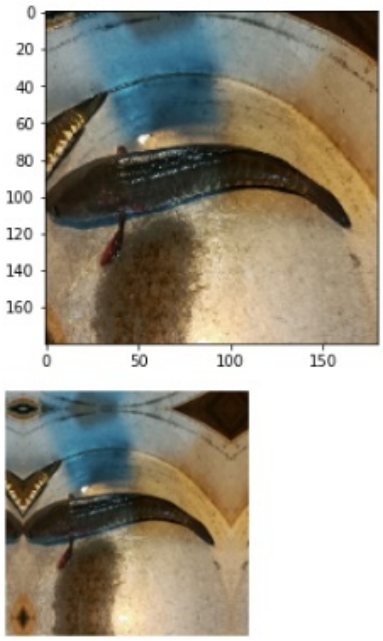


Figure 4.4: Performing random zooms

Lastly we perform horizontal flip by setting horizontal_flip=True.

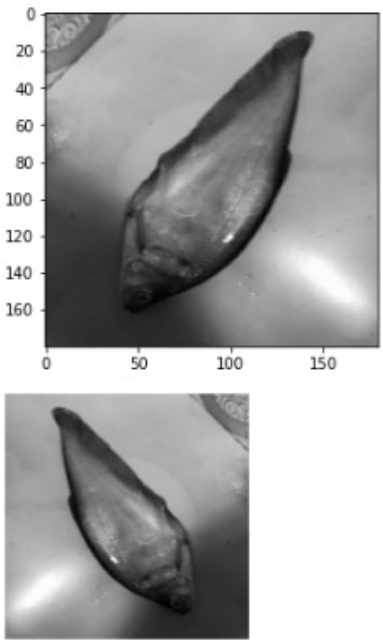


Figure 4.5: Performing Horizontal flip

Due to these random rotations and flips that we had performed, the images will now contain certain empty pixels. Leaving empty pixels in the image is not ideal, which is why we need to deal with this by setting the fill_mode. In our case, we chose to reflect as the value of the fill mode, which in simple terms performs a reflection and fills the empty pixels with those reflected pixels.

4.3 Data Splitting (Train-Test)

In order to evaluate the performance of a machine learning model, the data fed into it needs to be split. This splitting is called the train-test split, which is a fundamental step for supervised learning techniques implemented by certain classification and regression models [24]. After the split, one division of the data is called the train, which is the data used when the fit function is executed on the model, while the second part of the data after splitting, the test, is kept unused in the training phase and rather used to test how well the model has learned by feeding the data later on in the testing phase of the program. In other words, this test data introduces the model to previously encountered data and evaluates the model's adeptness in its estimates or predictions by comparing these predictions to the expected values. Thus the performance of the model is evaluated with new, previously unseen data after the training is complete.
In summary, in the training part, we provide the model with both the known inputs and the outputs when executing the model's fit function. However, in the testing phase, we provide the model with just the input, get the models' predictions for the expected output, and compare these predictions with the actual expected output. If the evaluation scores are satisfactory, we can conclude that our model can predict from unknown data effectively.
In order to split, we use sklearn's train_test_split. The only configuration parameter to set is the size of these train and test sets, which is specified, with a float between 0 and 1, let us say x, which indicates the percentage of the test data, thus making the train data equal 1-x. Common splitting ratios include train 67%, test 33% and train 80%, test 20% or even 50% 50% in certain cases [25]. In our case, we chose to go with the 80-20 ratio. In addition, we further split the 80% train data into the 80-20 ratio in order to create another set known as validation. This is done in order to tune the classifier's hyperparameters.

Chapter 5

Model Implementation

5.1 Model Proposed

Deep learning-based breakthroughs in marine object recognition and classification have accelerated substantially in the last decade. Raw images acquired from sources can be integrated directly and can extract necessary features in deep learning. While algorithms like Faster R-CNN break the detection problem in two, detecting potential areas of interest using Region Proposal Network and then performing recognition on these regions, YOLO performs the task in one layer. Furthermore, compared to other algorithms, YOLO has been proven to have given results with more precision and speed. While YOLO gives better results in detecting an object, Other Convolutional Neural Network algorithms such as VGG16/DenseNet learns distinctive features given a dataset and classify images without having any kind of human supervision better. In CNN, a fully connected feed-forward network, abatement of the parameters is efficiently handled while maintaining the quality of a model.

Taking into consideration the above mentioned accounts, we have decided to implement YOLO architecture along with architectures that provide superior results in picture classification and compare which of the architecture provides best results for our project. After the data acquisition process, we have split our training and testing phase into two. Which allowed us to separately pre-process our data as per the model. We have augmented our data followed by splitting the dataset for validation, training and testing data. The training and testing ratio has been kept at 80:20. All the data has been re-scaled to a size for the model implementations. By training and testing our dataset separately, we have higher chances of detecting and classifying our object with higher precision. As we have already mentioned, YOLO and other architectures used are highly efficient for detection and classification individually, but together they might reach heights that have not been achieved yet. To our knowledge, In Bangladesh, projects like this haven't been attempted. Since deep learning is efficient with large datasets, we have taken a large enough Bangladeshi dataset containing well-known fish species. We take the dataset and feed it to YOLO first and then the other models. As for the implemented architecture, we have used YOLOv4, VGG-16, DenseNet and Xception for our experiment and result analysis.

5.2 YOLO implementation

Yolo architecture, a state of the art object detection system, consists of 54 convolutional layers each connected with a Mish activation layer and a batch normalization layer.

5.2.1 Initial setup

Annotated data which contains the object class, height, width, and bounding box coordinates information, an essential part of detecting objects,is required in different formats in different object detection architectures in the training process.The annotation information for training YOLOv4 should be in the form of a text file. For annotation a python based GUI tool has been used which allows the annotation to be saved in PASCAL VOC and YOLO format. For our model we would be saving the annotation in YOLO formal. We used the Darknet pre-trained model after the annotation. To produce an executable Darknet model we need a makefile which can be installed in a NVIDIA GPU based machine followed by installing CUDA.

5.2.2 Parameter for training

Since we'll be working on one class (fish) we will have the index set to 0. The batch size which refers to the number of images chosen, for training it has been set to 64 and the subdivision is 16 in the cfg file. The batch size is usually chosen based on the memory system size. By default in YOLO the width and height is 608 we changed it to 416. And channel=3 indicating we will be processing RGB images. The height and width have to be a multiple of 32. The maximum number of batches can be run using this model however must be 2000 which depending on the number of classes may increase. we had to take 80-90 percent of the batch number as steps to get less average loss. we penalize huge weight changes between iterations In order to avoid overfitting, meaning the model could not grasp the underlying concept [1]. The parameter momentum handles the penalizing while decay controls the penalty term. We have kept momentum 0.09 and decay 0.005. The learning rate can as well be controlled. We observed in our model that the learning rate updates from 0.00 to 0.001 as we go along. Using the darknet framework longer we train the model less loss and more precision it gives us.

5.3 VGG-16 implementation

5.3.1 Initial setup

Prior to the discussions of implementation, let us discuss the setup and resources we utilized in performing these training sessions. Tensorflow and Keras libraries were used for model description, training and validation. In terms of the environment, we chose Google's Collaborative platform, equipped with Tesla K80 GPU and cloud computing support, in order to run our code. Another advantage of using Google Colab is that it makes it possible to host our data on the Google cloud again, cutting down the computational strains. Apart from these advantages working as a team

where multiple researchers are to contribute simultaneously becomes simpler with a platform like Google Colab.
In terms of training the different architectures we implemented, a batch size of 32 and an Epoch number of 30 with steps per epoch being 15 were used. In terms of optimization, we utilized Adam optimizer, which optimizes by varying the learning rates when it comes to the model's parameter searches for training.
Before model creation, we need to import certain libraries, which include tensorflow, keras, types of layers for the model including Convolutional layer, Max Pooling layer, activation layer etc. We also import libraries like numpy, seaborn, os etc., for various functions that we need to perform. Most of the imports are shown below:

```
import os
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sn
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import np_utils
from pathlib import Path
```

Figure 5.1: Imports in initial setup

5.3.2 Model creation

When it comes to model creation, we set the input shape to (224,244,3) and start organizing the layers according to the VGG-16 architecture discussed in the background analysis section. We add two Conv2D layers followed by a MaxPooling2D layer forming each block of layers. We form two such blocks. This process continues; however, in the next 3 blocks, we use three Con2D layers followed by a Maxpooling2D layer. Finally, we finish building the model by adding 3 fully connected layers, of which the last layer has the hyperparameter of 8 due to the 8 classes of fishes that our model needs to classify from the data. Activation function is set to sigmoid, again due to the fact that our dataset contains more than two classes, and the model needs to classify multiple classes. A section of the output of the model summary is shown below:

```
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

flatten (Flatten)            (None, 25088)             0

dense (Dense)                (None, 512)               12845568

dense_1 (Dense)              (None, 8)                 4104

=================================================================
Total params: 27,564,360
Trainable params: 12,849,672
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 5.2: Section of the model Summary

Next, we compile the model with the Adam optimizer and with parameters of loss set to categorical-cross entropy and metrics to ['accuracy']. In addition, we specify a callback with an early stopping condition by monitoring the accuracy, which in simple terms means that if the accuracy of multiple consecutive epochs does not increase, then the training process will stop. After doing all these, we finally call the fit function, thus training the model with 30 epochs which gives us a test set classification accuracy of 92.7%.

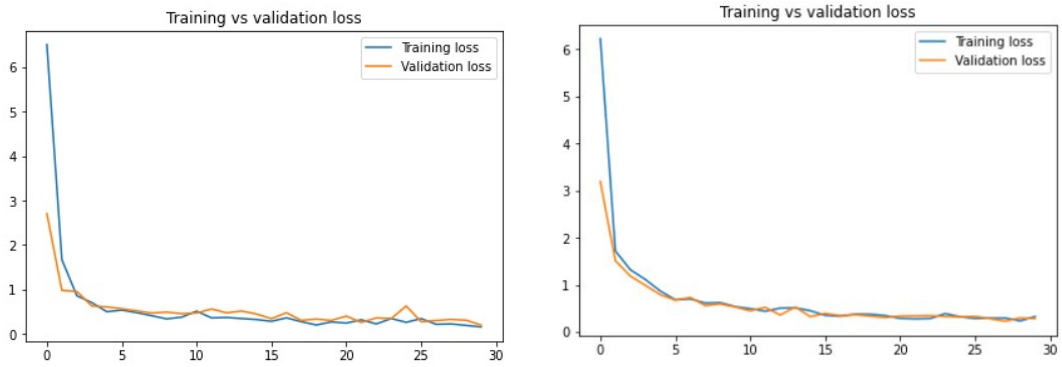
5.3.3 Graphics and visualization

We visualize the accuracy, loss, f1-scores and history of precision for both our train and test sets. In order to do so, we make use of the matlib.pyplot library, which was imported earlier. Accuracy is the metric that tells us how many predictions our model made correctly from all the inputs. On the other hand, loss is the metric that denotes the prediction error of our model, and unlike accuracy, this is not a percentage and rather an aggregation of the errors. This loss is actually how the model improves as the loss is used to update the weights of the neural network. Below accuracy and loss for both train and validation are plotted in two different graphs with the epoch numbers on the x-axis and the values of loss and accuracy

on the y-axis.

(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 5.3: Train vs validation Accuracy



(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 5.4: Train vs validation Loss

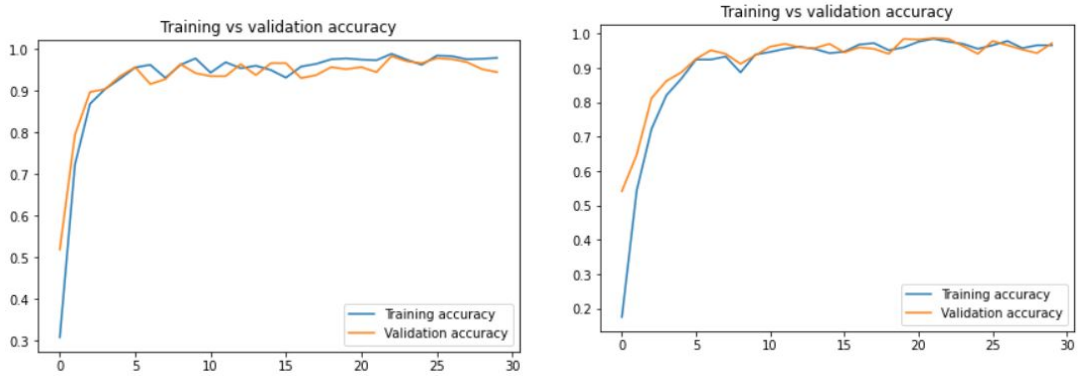
5.4 Densenet implementation

The initial setup remains the same as VGG-16 with similar imports of libraries and the Adam optimizer on Google Colab. The number of epochs is kept at 30 with a step size of 15 per epoch with an input size of (224,224,3), similar to the initial setup of VGG-16. After the initial setup and taking the input shape, we move on to model creation, where firstly, we start with a convolutional and pooling block. Next, we create the dense blocks. Densenet-121 architecture consists of a total number of 4 dense blocks, with the first block consisting of 6 repetitions of one convolutional layer with kernel size of 1 and one convolutional layer with kernel size of 3, after which a concatenation operation is performed. This is followed by 12, 24 and 16 repetitions of the same combination of convolutional layers for the three remaining blocks, respectively. In between these dense blocks, we implemented the transition blocks as discussed in the chapter detailing the architecture of the network. The transition blocks are made up of one convolutional layer with a kernel size of 1 and

an AvgPooling2D layer with the parameters of pool size = 2 and stride = 2.

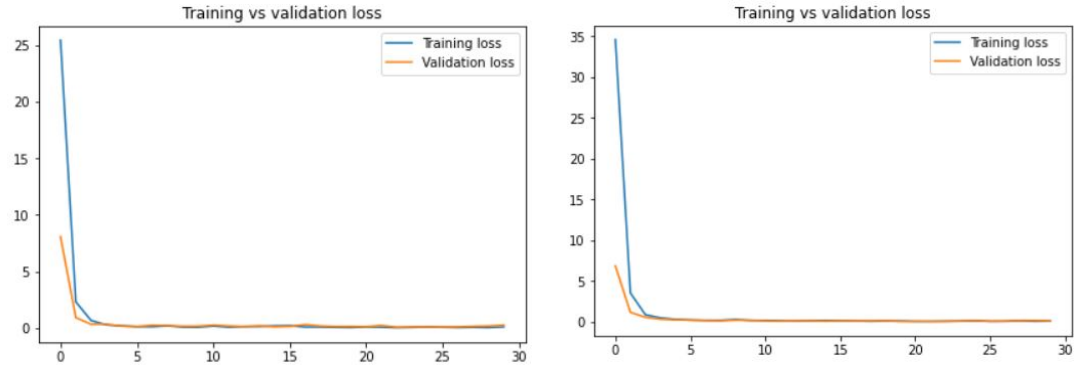
After the creation phase, we compile the model with the Adam optimizer, with parameters of loss set to categorical-cross entropy and metrics to ['accuracy']. Moreover, we utilize a callback with an early stopping condition by monitoring the accuracy, which in simple terms means that if the accuracy of multiple consecutive epochs does not increase, then the training process will stop. After doing all these, we finally call the fit function, thus training the model with 30 epochs which gives us a test set classification accuracy of 94.5%.

Similar to the VGG-16 implementation, we visualized the accuracy loss, f1-scores and history of precision. Accuracy and loss for both train and validation are plotted in two different graphs with the epoch numbers on the x-axis and the values of loss and accuracy on the y-axis:



(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 5.5: Train vs validation Accuracy



(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 5.6: Train vs validation Loss

5.5 Xception implementation

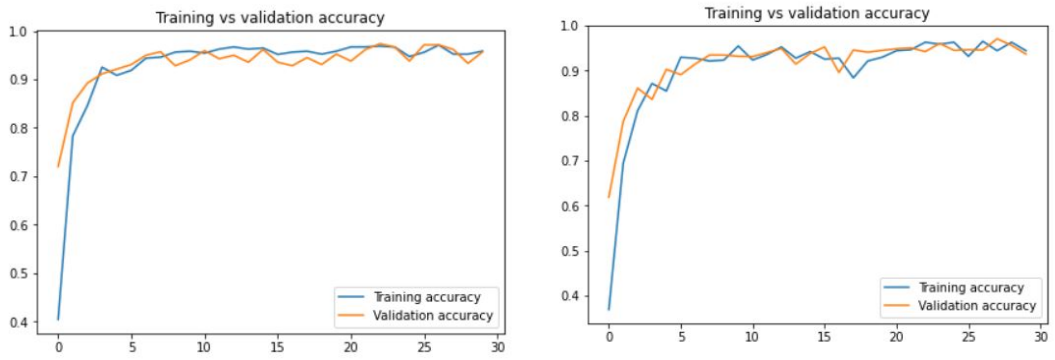
XCeption's architecture consists of Depth Wise Separable Convolution blocks combined with Max Pooling, all of which are coupled via shortcuts in the same way as ResNet implementations are. We have kept our input size as before. The depthwise convolution is not followed by a point-wise convolution in Xception; rather the sequence is reversed. Building the model consists of building the entry flow, middle flow and exit flow separately. The kernel size has been kept at 3 for each flow.

After the initial setup, we start by implementing the entry flow by defining a function where a conv2d layer is followed by batch normalization, and ReLU layers and this set of layers are repeated once. For the next part of the entry flow, a SeparableConv2D and batch normalization is followed by a ReLU layer and SeparableConv2D layer. We add the outputs of the first two parts of the entry flow using Add()(). The third part of the entry flow contains the following layers of ReLU, separableConv2d, and batch normalization. These layers are repeated once, and the part ends with a max pooling layer. Again we use the Add()() function for the skip connection. The fourth part of the entry flow follows the same pattern of layers as the third, and the entry flow ends with the function Add()().

The middle flow consists of three parts, each containing a ReLU followed by a SeparableCon2D and batch normalization. The flow ends through the use of Add()(), similar to the entry flow. The middle flow is repeated 8 times.

Finally, we implement the exit flow where two repetitions of a ReLU followed by a separableConv2d and batch normalization is executed. The first part is completed with a max pooling layer and an Add()() function. In the second part of the exit flow, two repetitions of a separableConv2d, a batch normalization, followed by a ReLU is used, and the exit flow ends with a GlobalAveragePooling.

A visual representation of the loss and accuracy achieved in the test and train of the model is enclosed below:



(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 5.7: Train vs validation Accuracy

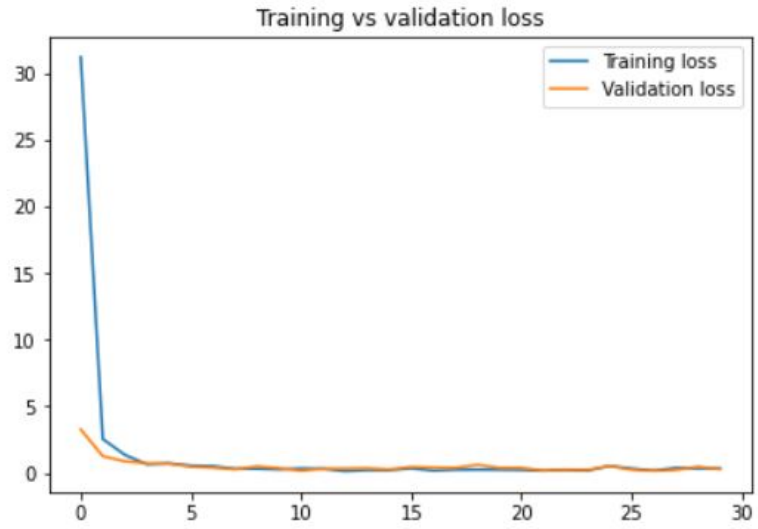


Figure 5.8: BDIndigenousFish: Train vs validation Loss

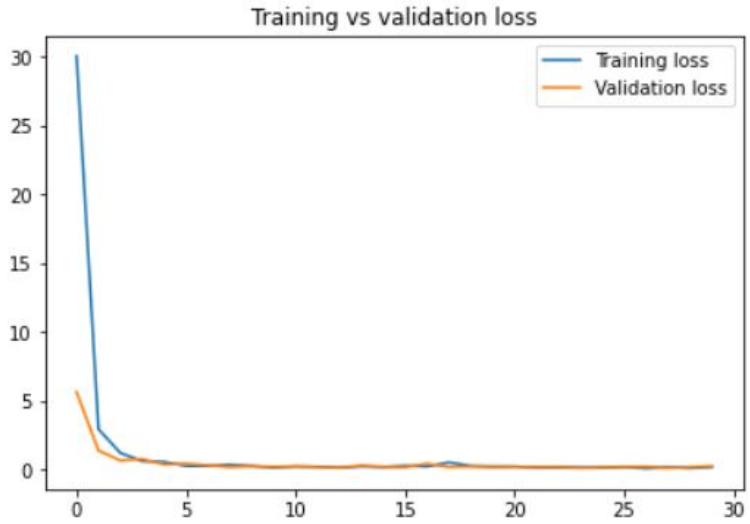


Figure 5.9: A-Large-Scale-Fish-Dataset:Train vs validation Loss

Chapter 6

Experimental Results and Analysis

In this project, we have worked with 3 models for classification, namely VGG-16, DenseNet, and Xception, for the classification task along with YOLOv4 for detection, and now, in this chapter, we will discuss the performance of the algorithms and provide relevant comparisons. In order to compare, we use confusion matrices and classification reports to get accuracy, precision, recall, and f-1 scores, which we will use to evaluate the performances. The equations for precision, recall, and f-1 scores are given below:

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

Here TP stands for True Positives, FP stands for False Positives and FN stands for False Negatives. In the equation below, we can see how the precision and recall are used to calculate the F_1 score.

$$f1\_score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6.3}$$

The tables below illustrates the accuracy and loss achieved by the models on the datasets. To have a fair comparison we kept the number of epochs constant at 30:

Model VGG-16				
	BDIndigenousFish		A-Large-Scale-Fish-Dataset	
	Accuracy	Loss	Accuracy	Loss
Train	0.912	0.222	0.894	0.267
Validation	0.923	0.196	0.893	0.286

Table 6.1: Accuracy and Loss scores for VGG-16

Model DenseNet-121				
	BDIndigenousFish		A-Large-Scale-Fish-Dataset	
	Accuracy	Loss	Accuracy	Loss
Train	0.979	0.113	0.967	0.107
Validation	0.945	0.266	0.973	0.116

Table 6.2: Accuracy and Loss scores for DenseNet-121

Model Xception				
	BDIndigenousFish		A-Large-Scale-Fish-Dataset	
	Accuracy	Loss	Accuracy	Loss
Train	0.958	0.334	0.944	0.208
Validation	0.957	0.276	0.936	0.281

Table 6.3: Accuracy and Loss scores for Xception

In the table below we have shown the precision and f1-scores for all the models we have used over the two datasets.

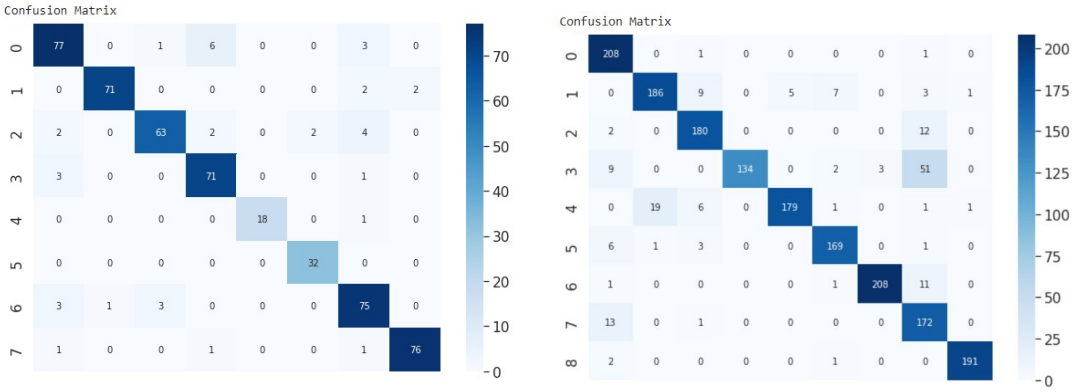
	BDIndigenousFish		A-Large-Scale-Fish-Dataset	
	Precision	f1-score	Precision	f1-score
DenseNet-121	0.97	0.96	0.96	0.95
VGG-16	0.94	0.94	0.91	0.90
Xception	0.88	0.83	0.87	0.79

Table 6.4: Precision and f1-scores

Yolov4				
Datasets	Accuracy	Loss	Precision	f1-score
BDIndigeneousFish	0.89	0.205	0.90	0.90
A-Large-Scale-Fish-Dataset	0.91	0.191	0.91	0.91

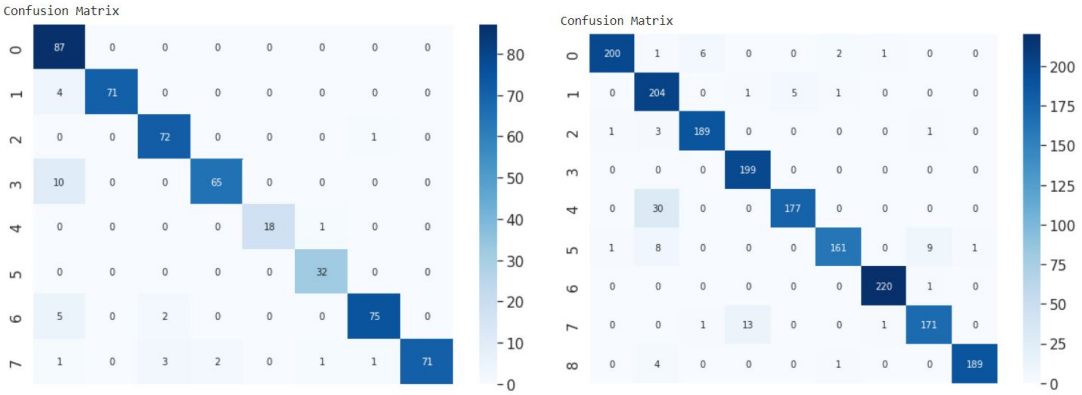
Table 6.5: Accuracy, loss, precision & f1-scores for Yolov4

Confusion matrices for three models in the classification stage (VGG-16, Xception and DenseNet-121), trained with the BDIndigenousFish dataset and A-Large-Scale-Fish-Dataset is given below.



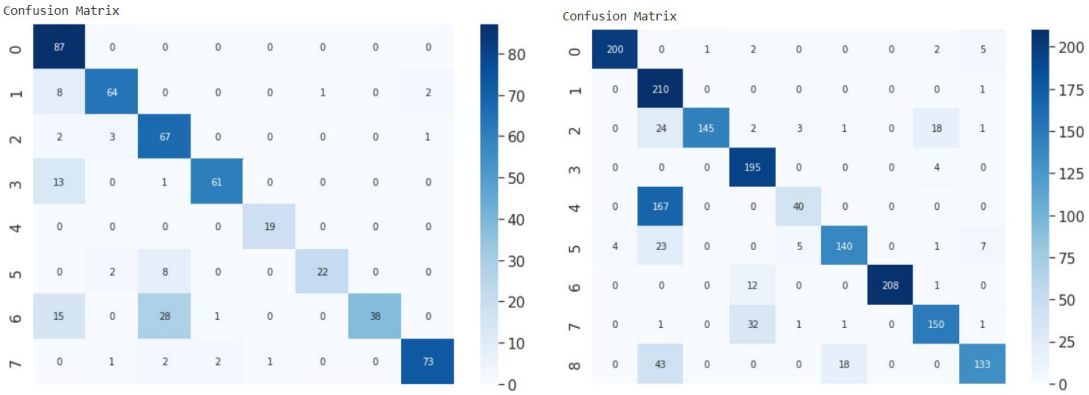
(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 6.1: Model VGG-16: Confusion matrices for both datasets



(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 6.2: Model DenseNet: Confusion matrices for both datasets



(a) BDIndigenousFish dataset (b) A-Large-Scale-Fish-Dataset

Figure 6.3: Model Xception: Confusion matrices for both datasets

6.1 Comparisons

We have prepared bar charts to illustrate the differences we had using our models with both datasets. We have made comparisons on the accuracy, loss, precision, and f1-scores. We have used 3 models, VGG-16, DenseNet and Xception, and run them over our datasets, BDIndigenousFish and A-Large-Scale-Fish-Dataset. We noticed that even though we were using datasets of 2 different sizes, we still had very similar values for the aforementioned accuracy, loss, precision, and f1-scores. For example we had a accuracy of 0.91, 0.98 and 0.96 in VGG-16, DenseNet and Xception respectively for BDIndigenousFish and 0.89, 0.97 and 0.94 for A-Large-Scale-Fish-Dataset.

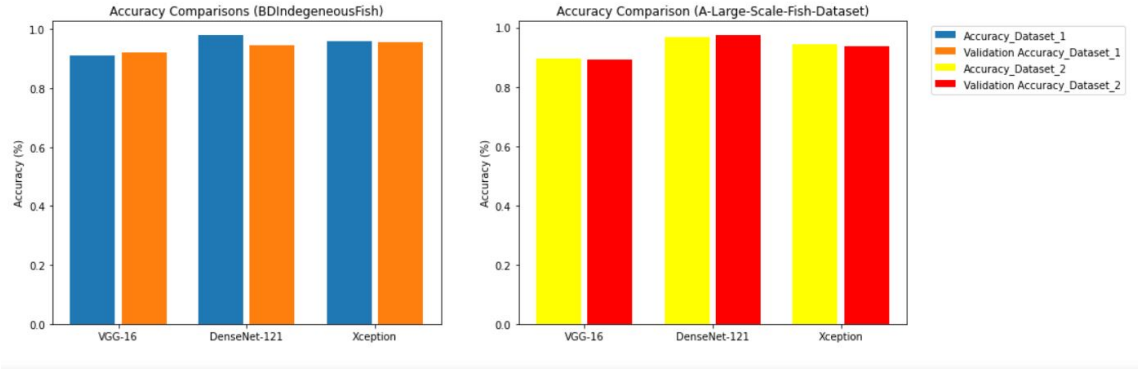


Figure 6.4: Accuracy Comparison

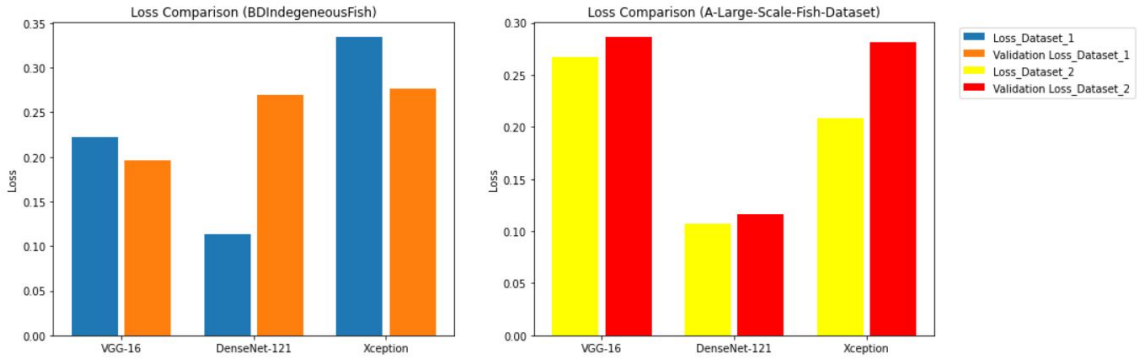


Figure 6.5: Loss Comparison

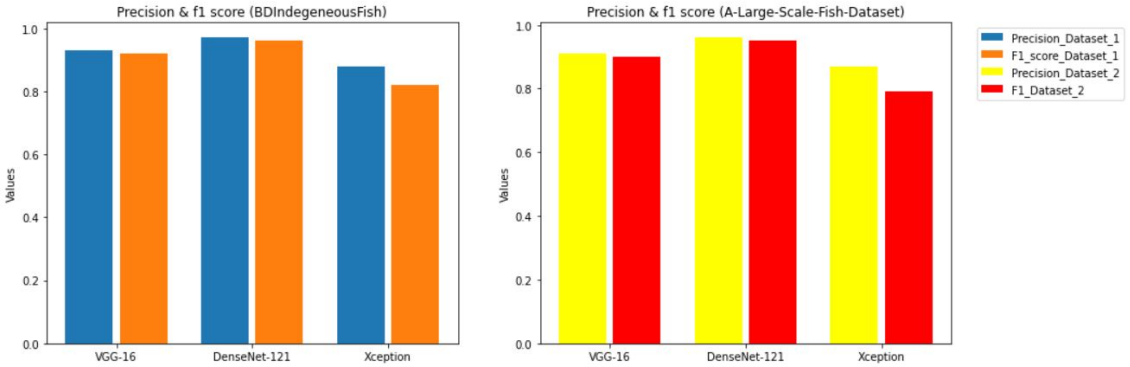


Figure 6.6: Precision & F1 scores

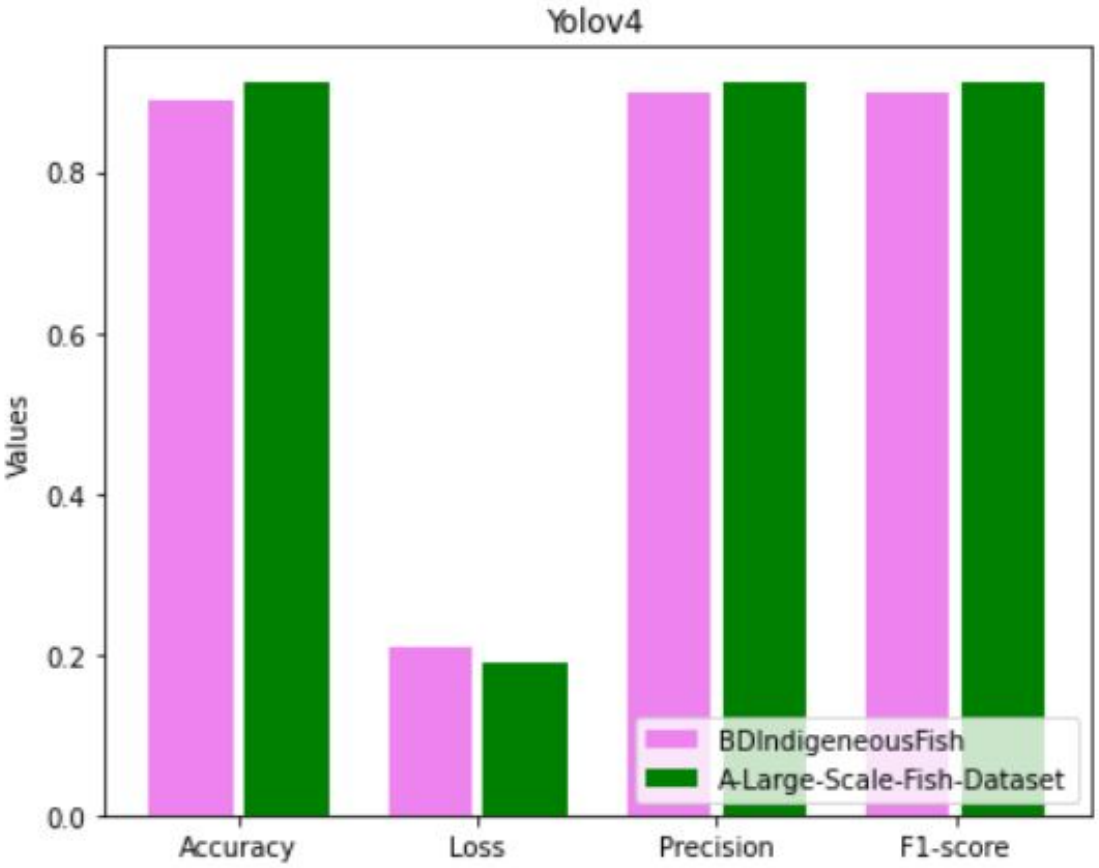


Figure 6.7: Yolov4 results

Some examples of our given outputs are given below. Figure 6.7 shows our algorithm detecting a fish and Figure 6.8 shows fish being classified by their species:

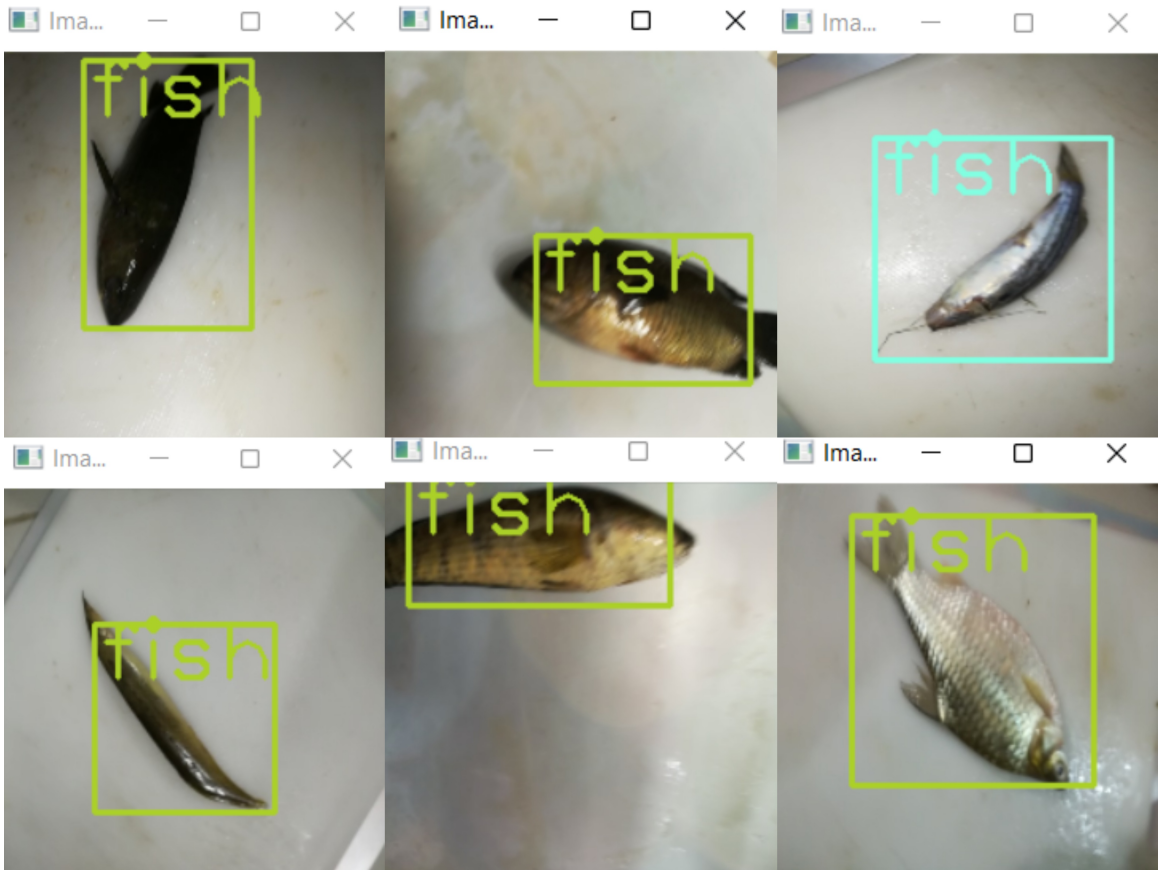


Figure 6.8: Detection

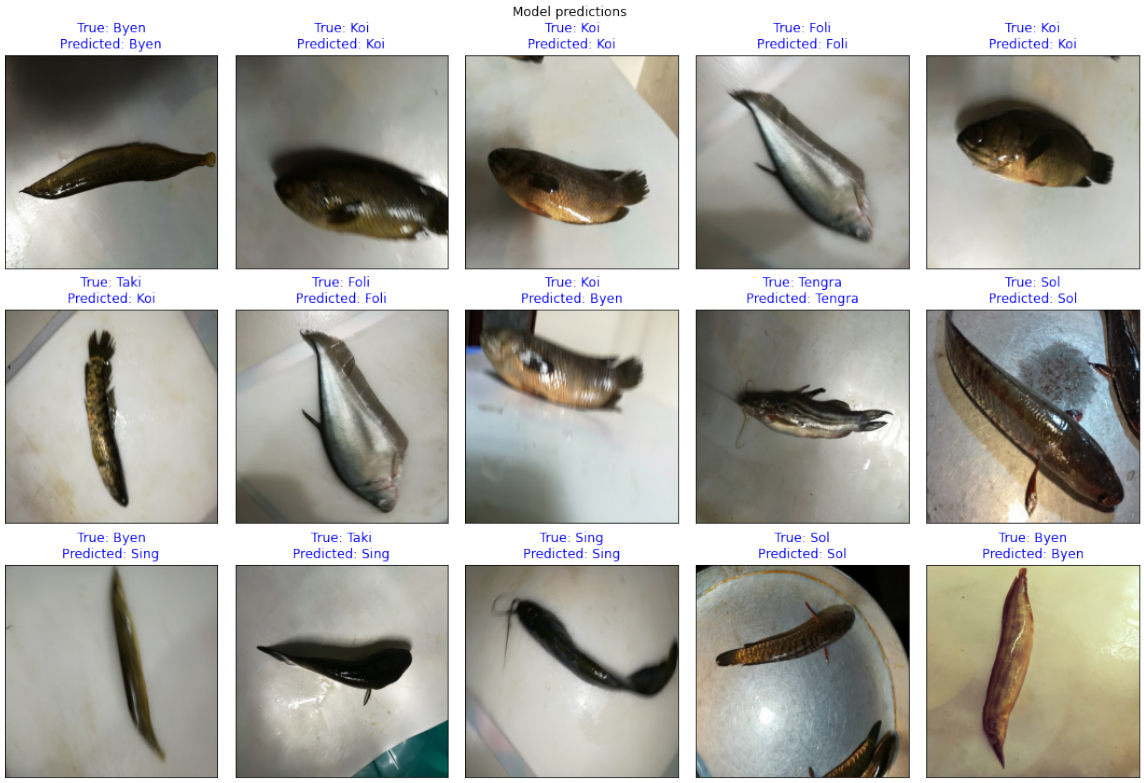


Figure 6.9: Classification

Chapter 7

Conclusion

In conclusion, our research is a study of different deep learning techniques that we implemented and compared in light of fish detection and classification. We discussed various previous works and experiments related to the topic. In our work, we used two datasets, namely BDIndigeneousFish and A-Large-scale-Fish-Dataset, and ran the data through algorithms that include YOLOv4, VGG-16, DenseNet-121, and Xception. An in depth study and analysis of the architectures of the models are presented in the paper along with how we implemented the respective models. The results that we achieved are compared and contrasted using various visualizations at the end of the paper. We feel that our paper will provide an excellent comparison for any future researchers of topics relating to fish classification.

7.0.1 Limitations and shortcomings

We faced various challenges and limitations while performing our research project. The initial plan was to produce our own dataset of different fish species native to Bangladesh through fieldwork. However, due to various restrictions imposed on us because of the pandemic (Covid-19), we could not follow through with this planning and used datasets retrieved from the internet instead. In terms of the accuracy that we achieved, the numbers are satisfactory as all the models had accuracy and val_accuracy values over 90%, a percentage threshold we aimed to acquire from the start. In addition, loss values were also acceptable. However, we would like to prevent the loss of information even more significantly through the application of specific techniques that we could not implement. Furthermore, the number of fish species that our model is trained to recognize remains an area of improvement, and the use of an even larger-scale dataset can improve our research considerably.

7.0.2 Future Work

Our research obtained a considerable level of accuracy from the models we used, which is why our research can provide a renewed interest in the research of fish species and thus the preservation of species in the context of Bangladesh. However, some additional components and ideas need to be implemented and researched in order to effectively study fish habitats in addition to our work. In the future, our research can be improved by finding ways to deduce the health of a particular species of fish (here, health is a value calculated depending on the number of individuals within a habitat). Furthermore, detecting the presence of invasive non-native specie

that might pose threats to the health of a given habitat can be the next step in the improvement of our work. In such a case, the model should be able to calculate this health value from the number of individual fishes it detects for different species of fishes. Moreover,calculating the health might help identifying endangered specie within the given ecosystem. In addition, to truly understand habitat changes, certain environmental changes need to be studied, which research can be done with the knowledge of our research. Furthermore, our failure in collecting and producing a large-scale image dataset of fish species in a Bangladeshi context can also be attempted by future researchers.

Bibliography

[1] K. Blanc, D. Lingrand, and F. Precioso, "Fish species recognition from video using svm classifier," in *Proceedings of the 3rd ACM International Workshop on Multimedia Analysis for Ecological Data*, 2014, pp. 1–6.

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

[3] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[4] X. Li, M. Shang, J. Hao, and Z. Yang, "Accelerating fish detection and recognition by sharing cnns with objectness learning," in *OCEANS 2016-Shanghai*, IEEE, 2016, pp. 1–5.

[5] X. Sun, J. Shi, J. Dong, and X. Wang, "Fish recognition from low-resolution underwater images," in *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, IEEE, 2016, pp. 471–476.

[6] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[7] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[8] M. Ferguson, R. Ak, Y.-T. T. Lee, and K. H. Law, "Automatic localization of casting defects with convolutional neural networks," in *2017 IEEE international conference on big data (big data)*, IEEE, 2017, pp. 1726–1735.

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[10] D. Rathi, S. Jain, and S. Indu, "Underwater fish species classification using convolutional neural network and deep learning," in *2017 Ninth international conference on advances in pattern recognition (ICAPR)*, IEEE, 2017, pp. 1–6.

[11] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[12] B. V. Deep and R. Dash, "Underwater fish species recognition using deep learning techniques," in *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, IEEE, 2019, pp. 665–669.

[13] M. A. Islam, *Bdindigenousfish2019*, https://github.com/knowaminul/BDIndigenousFish2019, Mar. 2019.

[14] A. Rosebrock, *Keras imagedatagenerator and data augmentation*, https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/, Jul. 2019.

[15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[16] Y. Han, C. Wei, R. Zhou, Z. Hong, Y. Zhang, and S. Yang, "Combining 3d-cnn and squeeze-and-excitation networks for remote sensing sea ice image classification," *Mathematical Problems in Engineering*, vol. 2020, 2020.

[17] V. Kurama, *A review of popular deep learning architectures: Densenet, resnext, mnasnet, and shufflenet v2*, https://blog.paperspace.com/popular-deep-learning-architectures-densenet-mnasnet-shufflenet/, 2020.

[18] K. Muri Knausgård, A. Wiklund, T. Knutsen Sørdalen, K. Halvorsen, A. Ring Kleiven, L. Jiao, and M. Goodwin, "Temperate fish detection and classification: A deep learning based approach," *arXiv e-prints*, arXiv–2005, 2020.

[19] O. Ulucan, *A large scale fish dataset*, https://www.kaggle.com/datasets/crowww/a-large-scale-fish-dataset, 2020.

[20] O. Ulucan, D. Karakaya, and M. Turkan, "A large-scale dataset for fish segmentation and classification," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, IEEE, 2020, pp. 1–5.

[21] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn. 2020 ieee," in *CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 1571–1580.

[22] K. Dey, M. M. Hassan, M. M. Rana, and M. H. Hena, "Bangladeshi indigenous fish classification using convolutional neural networks," in *2021 International Conference on Information Technology (ICIT)*, IEEE, 2021, pp. 899–904.

[23] G. Maindola, *Yolov4 object detection tutorial with image and video : A beginners guide*, https://machinelearningknowledge.ai/yolov4-object-detection-tutorial-with-image-and-video/, 2021.

[24] J. Tan, J. Yang, S. Wu, G. Chen, and J. Zhao, "A critical look at the current train/test split in machine learning," *arXiv preprint arXiv:2106.04525*, 2021.

[25] B. Vrigazova, "The proportion for splitting data into training and test set for the bootstrap in classification problems," *Business Systems Research: International Journal of the Society for Advancing Innovation and Research in Economy*, vol. 12, no. 1, pp. 228–242, 2021.

[26] Z. Zakria, J. Deng, R. Kumar, M. S. Khokhar, J. Cai, and J. Kumar, "Multi scale and direction target detecting in remote sensing images via modified yolo-v4," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2022.