Deep Learning Based Fish Species Classification

Deva Kumar Gajulamandyam, Sainath Veerla, Tiffany Li, Jinthy Swetha Mamillapalli

Department of Applied Data Science, San José State University

DATA 270 Data Analytics Processes

Dr. Eduardo Chan

December 10, 2023

Abstract

Accurate classification of fish species is essential for taxonomic research, providing a foundation for monitoring invasive and endangered species populations, marine ecosystem conservation, and sustainable fisheries management. Traditional classification methods which rely on trained technicians and taxonomists, are time-consuming and lack scalability, and the number of subject matter experts are declining. The rapid growth of underwater image data, coupled with advancements in machine learning, presents an opportunity for automated fish species identification. This study addresses the limitations of current classification methodologies by employing the latest datasets and advanced deep learning techniques to develop an automated, high-accuracy classification system for taxonomic and biological research. Although few systems achieve this task, they are limited to very few fish species. To identify wide varieties of species, two comprehensive datasets, DeepFish and Fish4Knowledge were integrated and this data comprising images of 37 species, undergoes several data engineering and transformation steps like noise removal, image augmentation, normalization, reduction, and image enhancement before being trained on deep learning models using Convolutional Neural Networks (CNNs), and fine-tuned existing state-of-the-art models like Inception-ResNet, MobileNet, and Vision Transformers. These models achieved accuracies of 95%, 93%, 96%, and 92%, respectively on unseen data. Model effectiveness was evaluated based on accuracy as the primary metric and with additional metrics such as Categorical Cross Entropy, precision, and recall. The MobileNet model outperformed other models. This study demonstrates the effectiveness of deep learning techniques for automated fish species classification, supporting conservation efforts and biological research.

Introduction

Project Background and Executive Summary

Project Background

Oceans, vast and mysterious, cover the majority of the earth's surface. Therefore, it is vital to understand the underwater ecosystem. However, this task has always been troublesome due to the challenges present in the exploration of the ocean. The species present underwater significantly contribute to the ecological balance of the planet. Among these, fish, in particular, are a crucial part of the food web, in the cycling of nutrients, and ocean biodiversity. It is of paramount importance to understand their distribution. Historically, there is a dependency on marine biologist's observations to understand fish diversity and classify them. This gave an extensive knowledge database but this process is time-consuming and vulnerable to human errors such as misclassification. In a nutshell, employing effective methods of fish classification is necessary. With advancements in technology such as Convolutional Neural Networks (CNNs), it is possible for more accurate classification of fishes. Our project, “Deep learning for fish species classification”, aims to classify fish species by developing a robust model using deep learning technologies.

Needs and Importance

Fish classification is a process of identifying and categorizing based on their characteristics. Accurate fish classification is a necessity in today's world as it helps monitor the fish population, enabling conservationists to implement targeted strategies for protecting endangered species. It helps scientists understand the evolution history of fish and the factors that influenced their diversification. For aquaculture industries, classification can help them

manage stocks by regulating fishing activities and avoiding overfishing. Certain species produce toxins harmful to the aquatic system, and identification of such species helps protect the habitat. According to Deely et al. (2022), marine tourism has been growing exponentially in recent times, and precise knowledge of fish species helps in magnifying the experience as well as implementing tourism practices to avoid any damage to marine life. Classification assists researchers in genetic studies by helping them choose appropriate groups of species for genetic sequencing. Therefore, it is important to develop an advanced, precise, and scalable system for classification.

Targeted Problem

When compared to human techniques, automation of the fish classification task yields more accurate results and speeds the classification process. Automation does, however, not come without difficulties. Small changes in color, fin shape, or scale patterns make it difficult for many of the current classification schemes to accurately define a species. It causes species to be incorrectly classified. The fact that the majority of current classification models were trained on small datasets renders them less resilient to new species, which is a serious disadvantage. Building a model that can effectively classify a wide range of species is therefore necessary.

Motivation and Goals

The advent of technology revolutionized fish classification. Advancements in the fields of high-resolution underwater cameras and sonar imaging help reach unexplored territories, which is the primary motivator of the project. The availability of high-resolution images and computational algorithms enhances the classification process. The employment of efficient technology makes it possible for efficient real-time monitoring. The goal is to deploy deep

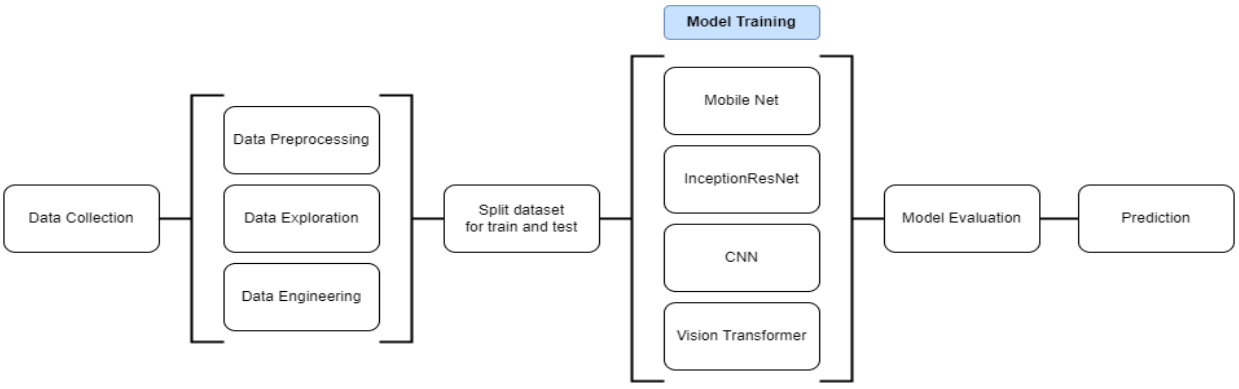
learning technologies to create a model that meticulously identifies and catalogs a wide range of fish with high degree of precision. This model should ensure scalability and quick processing without compromising on accuracy.

Proposed Approach

In this project, we use four deep learning techniques, CNN, MobileNet, InceptionResNet, and Vision Transformers, to classify the fish images. At the foremost step, data is collected and combined from two different sources. One of the many challenges in this project is the quality of the images. Underwater images are subject to noisy backgrounds, lighting conditions, occlusion, and water turbidity. Therefore, we preprocess the dataset to handle the quality of the images. After processing these images, exploration is done to evaluate the dataset and appropriate approaches for engineering and transformations can be decided. Figure 1 represents a simplistic overview of the proposed architecture used for research.

Figure 1

Proposed Workflow



Note. Detailed workflow used in the research

The final data is split into parts for training and validation. The training dataset is then fed into the model and all the performance is evaluated to identify the best performing model. This model predicts the class of the input image.

Expected Project Contributions and Applications

The major contribution of the project is the development of a sophisticated model which is capable of accurately classifying a wide range of species even in challenging environments. This model ensures low latency processing and robustness in varied environments. This enables on-the-spot identification of species, which helps biologists determine the characteristics of the species instantly. It has numerous applications in various domains. Climatic conditions have drastic impacts on the oceans, which will result in a change in the distribution of species. Monitoring these changes using fish classification helps determine the resilience of different species to climatic changes. Marine parks can utilize this technology for monitoring their biodiversity, ensuring the well-being of the ecosystem. Certain species hold cultural and religious importance. Accurate classification helps in the protection of these species. They also have significant applications in water quality monitoring, recreation and educational platforms.

Project Requirements

Functional requirements

Our goal is to develop an image recognition system capable of accurately classifying input images into respective fish species. To achieve this goal, the model will be trained on a diverse dataset of images varying in resolution and representing different underwater environments. To make the datasets as diverse and unique data captured at different locations has been used. The data is publicly available and does not have any regulatory requirements to be

followed. The data is open and accessible giving access to everyone without any license and authorization required. In order to facilitate centralized management and privacy of this large training dataset, data has been stored in Google cloud storage. This will ensure accessible data storage and enable seamless addition of new images as they are collected during the project timeline. The data will be initially downloaded, processed in a local system using Jupyter, a python IDE to run python scripts for better understanding of data and then uploaded to the cloud. For developing the model, pre-trained architectures from the Keras Python library have been leveraged. All the model training, building and processing will be performed in the Vertex AI environment provided by Google Cloud. The environment also facilitates in change of hardware as per the requirements of the model making it easier for computational purposes along with a seamless transfer of data from buckets to the environment. These existing models will serve as a strong foundation which we can customize and fine-tune for our specific fish classification task.

The entire project is built on cloud in google thereby restricting access to anyone outside the team members. By training on diverse images, customizing pre-trained models, and employing rigorous evaluation metrics, our goal is to develop an optimized fish classification system capable of generalized performance in real-world deployment.

AI Requirements

The project requires the utilization of models capable of efficiently handling extensive datasets while autonomously identifying relationships within the data. ML models help in differentiating between fishes in various images and finding intricate details which in turn help in classifying. The model should also be able to identify similar patterns between images, making it an integral part of classification. Given that the gathered data is already labeled and

predominantly balanced, supervised learning is the apt choice, enabling adaptation to various data patterns. Supervised learning thrives in scenarios where labeled data is abundant, and this project is no exception. The selected models process myriad images, discerning intricate patterns between them. While traditional methods can achieve this, they come with certain limitations. Notably, Principal Component Analysis (PCA), an effective approach for feature extraction, is unsuitable as it belongs to the unsupervised approach, as articulated by Wang et al. (2009).

Iqbal et al. (2018) reviewed that Support Vector Machines (SVM) exhibit resilience to noisy images but suffer from sluggish learning rates, demanding a substantial investment in time and computational resources. Artificial Intelligence (AI) models, on the other hand, offer robustness and efficiency. Many AI models come pre-trained, providing a head start in the development process. Convolutional Neural Networks (CNN) stand out for their inherent feature extraction capabilities and data augmentation techniques, especially valuable when dealing with imbalanced datasets, as outlined by Alzubaidi et al. (2021).

Managing extensive datasets places considerable strain on resources, necessitating solutions for optimization. In this context, the MobileNet algorithm emerges as a viable choice due to its efficient memory usage, making it adaptable to resource-constrained environments as noted by Yuan et al. (2022). Its implementation is versatile, extending to various platforms and settings.

The challenges posed by the size of marine life require meticulous attention to image quality. Even minor compromises in image quality can significantly impede fish detection. Vision transformers, typically associated with text models, demonstrate a surprising aptitude for working with low-vision images and uncovering long-term dependencies within them as

highlighted by Khan et al. (2022). Transformers facilitate cross-modal learning allowing a more intricate understanding of data.

To elevate the capabilities of our model in detecting marine life and capturing fine-grained details, a fusion of Inception and ResNet architectures has performed better. This strategic combination yields improved performance and results, marking a pivotal enhancement in the project's overall effectiveness. These AI models harbor the added advantage of adaptability, especially when complemented by other models and techniques.

Data Requirements

To effectively detect and classify fish species underwater, we have gathered two key datasets - Fish4Knowledge and DeepFish. The Fish4Knowledge dataset contains 23 unique species recorded around coral reefs off southeastern Taiwan, with each species organized into folders and consisting of masked background images. Meanwhile, the DeepFish dataset captures 20 distinct species adapted to Australia's marine ecosystems. Additionally, videos were collected and each frame was converted into images to further expand the datasets. This provides more diversity in capture angles, lighting conditions, and fish orientations.

By combining species from two distinct regions and video frames, our model is exposed to a broad range of real-world underwater conditions, species diversity, and habitat characteristics. This comprehensive training data equips the model to enhance its adaptability to challenging deep-sea scenarios.

Ideally, having equal sample sizes for each species removes bias. However, some species are rarer or harder to capture, leading to imbalanced data. We use techniques like aggressive data augmentation and weighted loss functions to account for this imbalance while training the model.

Through training on this diverse and expansive dataset, the goal is to develop a robust model capable of generalized detection and classification. By exposing the model to variability, it can potentially survey remote deep sea areas typically inaccessible to humans.

Project Deliverables

Project Proposal

The project proposal outlines the research problem, background information, value and importance of the research, relevant literature, data sources, and SWOT analysis. It introduces the project to relevant stakeholders and gains their support and feedback.

Project Management Plan

The project management plan is a roadmap that guides the team through the execution, monitoring, and control of the project. It defines the project development methodology and project management software. For this machine learning project, CRISP-DM (Cross-Industry Standard Process for Data Mining) framework is applied to guide project execution. The WBS (Work Breakdown Structure) will be used to break down the project into manageable tasks and subtasks, providing a clear picture of the project scope. For each task, the effort estimation will be conducted in the group to estimate the time needed, allowing team members to plan ahead and allocate resources effectively. The Gantt Chart will be used to represent the project's schedule, visualizing the project timeline and assigned team members. Moreover, a PERT (Program Evaluation and Review Technique) chart will be used to map individual tasks and dependencies, helping to identify the critical path and optimize the project schedule. The plan also specifies the required project resources, such as hardware, software, and associated costs, to ensure that all necessary elements are in place before the project starts.

Data Management Plan

The data management plan provides an overview of how data is handled through its lifecycle. It includes information on data sources, acquisition methods, standards, data processing procedures, data analysis steps, metadata/documentation practices, plans for long-term storage, protocols for data sharing, access, and data citation.

Data Collection Plan

The data collection plan is developed based on the data management plan. It outlines data collection procedures, sources, schedules, and initial data storage methods.

Data Exploration Plan

The data exploration plan is aligned with the data management and collection plans. It describes the techniques and software used to explore data features and identify potential problems in the dataset before formulating the data engineering plan.

Data Engineering Plan

The data engineering plan includes procedures to integrate data from different sources, select target data, clean and preprocess data, split the data into training and testing sets, and define the roles and responsibilities for data engineering tasks.

Abstract

The abstract is a summary of the research paper. It states the research problem, methods, results, and the significance and implications of findings in a short and concise manner.

Presentation

The final presentation will be delivered in PowerPoint to present the research project outcome. It will cover the research problem, data collection and preprocessing, data exploration,

the rationale behind machine learning model choices, model training, accuracy and evaluation, limitations, suggestions for future work, a conclusive summary, and references.

Model Building and Evaluation Report (Individual Research Paper)

The individual research paper will conclude the machine learning model development and evaluation process. It will document software packages used to build and evaluate the model, justify the selection of specific machine learning algorithms, describe the data used, explain the training and hyperparameter tuning processes, and record model accuracy and performance. In addition, the paper will discuss model limitations and potential bias.

Final Group Report

The final group report will integrate all project components including the abstract, project and data management plan, data collection and exploration plan, data engineering plan and individual research results. It will provide a full report of the project and the contributions from all team members. A detailed overview of project deliverables along with the description and due date has been provided in Table 1.

Table 1

Project Deliverables List

Deliverable	Description	Due Date
Project Proposal	A document outlining the project idea, background information and objectives	September 25, 2023
Project Management Plan	A detailed plan including WBS, effort estimation, Gantt Chart and PERT Chart to ensure effective project management	October 30, 2023
Data Management Plan	A blueprint for handling data throughout its lifecycle	October 30, 2023

Deliverable	Description	Due Date
Data Engineering Plan	A plan outlining data processing and transformation steps	November 6, 2023
Data Collection Plan	A set of procedures for collecting and storing data	November 13, 2023
Data Exploration Plan	A plan for understanding data characteristics and patterns	November 13, 2023
Abstract	A short summary of the research paper	November 20, 2023
Presentation	A formal presentation summarizing the research project	November 27, 2023
Individual Research Report	A research paper on individual the individual machine learning model	December 8, 2023
Prototype	A demonstration of the machine learning process and outcomes	December 10, 2023
Final Group Report	A comprehensive research paper summarize the project	December 10, 2023
Production Application	A user-friendly front-end application enabling users to interact with the machine learning models	December 10, 2023

Prototype

A prototype will be provided to demonstrate the machine learning models' capability to classify fish species based on input images. Input an underwater fish image into the model, and it will promptly provide the corresponding fish species as the output.

Production Application

With a robust model trained by different datasets, a user-friendly front end can be

developed to form a production application. Based on user feedback and new data, the model will be continuously trained to adapt to more use cases and maintain high accuracy.

Technology and Solution Survey

Biodiversity and Conservation of wildlife lately has become an important issue as more and more species are becoming extinct every day. Especially, the flora and fauna within marine ecosystems are endangered with numerous threats and suitable measures must be taken to counteract their extinction. One of the ways to address this problem is the identification of fish species driven by technological advancements.

The traditional way of classifying fish species is a time intensive process, starting with data collection where the data is collected, processed by videographers and domain experts manually reviewing and labeling the data. This way of classification takes lots of manual effort by humans and is not scalable. An alternative approach for collecting underwater data is quintessential to save time and additional costs. One such method is collecting data using unmanned underwater vehicles where an array of cameras are mounted on the surface of vehicles and controlled externally to capture data. Han et al. (2019) discuss the use of multiple Autonomous Underwater Vehicles (AUVs) for high-availability data collection within a sensor network. This paper addresses how AUVs collect data and the deployment of robust mechanisms to assure malfunction detection and self-repair. They simulated an underwater environment with arbitrarily distributed nodes and a solitary drain in the network's center on the water surface. They intend to address two main issues: the high energy utilization of AUVs close to the node and the management of AUV failure. In the proposed method, the entire network is made dynamic and the AUVs' trajectories change periodically, thereby resolving the first problem (hot

region problem). In the event of an AUV failure, the node flags the data arriving from that AUV and informs all other nodes to disregard it. This strategy enhanced the packet ratio, network lifetime, and energy balance, allowing the system to efficiently collect data.

Salman et al. (2016) propose a Machine Learning based solution to classify various species of fish in an underwater environment which is prone to noise. They adopt Convolutional Neural Networks (CNNs), in a hierarchical setting to distinguish unique features of different species based on images. The paper emphasizes challenges posed by the water habitat such as unclear data, noisy images, irregular light intensity, inclusion of non-fish objects such as coral reefs, sea bed garbage like ship or plane wrecks. They have combined CNN with classical ML algorithms such as KNearest Neighbors (KNN) and Support Vector Machines (SVM) as this combination proved to be effective against environment variability. The model training was done on LifeCLEF2014 (LCF-14) with 10 classes and LifeCLEF2015 (LCF-15) with 15 classes of fish image datasets which are subsets of Fish4Knowledge dataset which is extracted from multiple video footage. Both datasets together account for 21 distinct image classes. To generalize the model to suppress noise, the images were transformed with blurriness, sharpness, noise degradation using average and Gaussian filtering. Dataset is split into training, validation and testing sets. The distribution of images after sampling is shown in Table 2 which is taken from Salman et al. (2016). The model performance is evaluated on metrics like Average Count, Average Precision, Average Recall. For comparison of proposed models, they trained the dataset on other models like SVM, KNN, Sparse Representation of features (SRC), SVM and KNN with Principal Component Analysis. From the experimentation results, it was evident that CNN-SVM,

CNN-KNN stood out in producing the highest performance out of all other models for both datasets. The entire modeling was done using MATLAB.

Table 2

Distribution of images after sampling

Dataset	Original	Generated training set	Generated validation set	Generated testing set	Total generated Images
LCF-14	20,000	70,000	30,000	6956	106,956
LCF-15	29,000	85,700	32,000	7500	175,200

Rathi et al. (2017) worked on a similar approach to detect and classify various species of fish in an automated way leveraging CNNs. They constructed a series of filter to preprocess images and built a model on top of this data. Pre-processing steps included noise removal such as non fish objects, plants, garbage etc. This is achieved by grayscaling an image, implementing Otsu's thresholding which classifies each pixel as either black or white based on the threshold value specified and produces a binarized image with fish in focus. Later, Erosion and Dilation are applied to this binarized image to reconstruct the broken image. This image is passed on to the CNN network with convolution layers, max-pooling layers for non-linear downsampling, and subsampling layers to store relative feature information. Three such blocks are constructed with a dropout and fully connected layer at the output. This architecture has been trained separately with various activation functions like ReLU, sigmoid, and tanh. The dataset used for training is F4K with a total of 27,142 images comprising 21 species. Comparison is done based on accuracy and the model with ReLU activation gave the highest accuracy of 96.2% whereas tanh and softmax gave considerably lower accuracies of 72.6% and 61.9% respectively. This could be

further enhanced by applying additional feature extractions to account for lost features in the images. Another takeaway is that the latency of prediction is 1.83 milliseconds per frame which makes this suitable for real time deployments.

Tharwat et al. (2018) in their paper discusses a novel approach based on a biometric method for fish classification. They have handpicked four classes of fish species and collected a total dataset of 241 images and trained on this data using a high dimensional technique called Weber's Local Descriptor (WLD) and Linear Discriminant Analysis (LDA) is used to compress the features and get the most relevant features on top of which AdaBoost model is used to predict the label of the input image. The WLD method constructs a histogram based on two features - differential excitation and orientation (Chen et al., 2009). This is a three-step process. The first step is calculating differential excitation for every pixel as the arctan of ratio of difference between the center pixel and surrounding neighbors to the center pixel. Next step is finding the orientation of pixels as arctan of ratio of differences in horizontal and vertical directions and finally the WLD histogram is computed at each pixel. After this, the color and texture features were extracted and normalized (using z-score normalization). Then, LDA is applied to this histogram to reduce the number of features and an AdaBoost classifier is implemented on this to train the model. Models with different features were compared with a varying number of single classifiers. The accuracy of all models increased as the number of classifiers increased and WLD with Color features normalized gave the highest accuracy of over 96%. Also, AdaBoost is replaced with other conventional algorithms such as KNN, Naive Bayes and MLP but they underperform when compared to AdaBoost.

Villon et al. (2021) explored the idea of few-shot learning (FSL) as an alternative to deep learning for cases when the dataset is smaller as deep learning requires large amounts of data to train reliable models. They propose an optimization based few shot learning method to train models on extremely small datasets. Comparison of this model with another deep learning model ResNet trained on a huge dataset is presented in the paper. They have used three thumbnail datasets (T0, T1, T2) extracted from 175 underwater videos as data during the experimentation. Two versions of ResNet model and FSL model were built each with T0 and T1 dataset and FSL is implemented using Reptile algorithm which learns by dividing the training data into multiple tasks producing a quick learner which is capable of learning tasks rapidly with fewer data samples (Nichol et al., 2018). FSL model trained on 10 images per species showed better performance than ResNet which is remarkable and FSL required much fewer images to reach asymptotic accuracy. FSL showed considerable accuracy of 64.92% in cases where its ResNet counterpart attained 78% as shown in Table 3.

Table 3

Comparison of accuracies evaluated by DL and FSL models

	Deep Learning		**Few Shot Learning**					
	T0	**T1**	**T0**			**T1**		
Images per species	3458	315	1 shot	5 shots	30 shots	1 shot	5 shots	30 shots
Mean	78	42.21	34.57	50.23	64.92	32.04	41.47	51.77
Std. Deviation	15.16	24.95	11.14	14.75	14.55	12.7	16.93	18.96

Note. Mean and Std. Deviation refers to the mean and standard deviation accuracy for all the species combined.

While training FSL for each support set, the number of classes were fixed with five and hyperparameter tuning was performed on the number of shots ranging from one to 30. Performance evaluation was done on the unseen T2 dataset and accuracy is used as a metric. However, these types of models are not robust due to large differences in standard deviation and are applicable in scenarios where the number of training samples are very low.

Iqbal et al. (2021) presented a system that can identify species of fish using deep convolutional networks using transfer learning. They have used a reduced version of AlexNet, a deep CNN model with fewer layers allowing for faster training and inference time but still able to perform equally well with other deep CNN models. The final convolution layer and fully connected layers are removed from AlexNet and softmax function is used at the output layer following a dropout layer. The proposed system was built on the Tensorflow framework. This model is trained on QUT fish dataset consisting of 3960 images encompassing six species taken from controlled, in situ and out-of-water environments. This produces variability in quality of images, background information. The train test validation split was 65:20:15 for each class. To test performance on untrained dataset, LifeCLEF-15 dataset is used and results are compared with AlexNet and VGGNet. The proposed model shows an accuracy of 90% on test data and 98% on validation data after the inclusion of the dropout layer before the output layer. In Spite of having less computation power and memory, it has outperformed traditional AlexNet by 4% on test data but could not match VGGNet's accuracy. Table 4 represents a summary of all the studies related to this project.

Table 4

Comparison of existing solutions

Author	Dataset	# of species	Architecture	Result
Salman et al. (2016)	LifeCLEF-14, LifeCLEF-15	10, 15	CNN-SVM, CNN-KNN	96.75% (LCF-14), 93.65% (LCF-15)
Rathi et al. (2017)	Fish4Knowledge	21	CNN	Accuracy of 96%
Tharwat et al. (2018)	Undisclosed data source	4	Weber's Local Descriptor + LDA + AdaBoost	Accuracy of 96%
Iqbal et al. (2021)	QUT (training) LifeCLEF-15 (testing)	6	Reduced AlexNet	Accuracy of 90%
Villon et al. (2021)	T0, T1, T2 (from 175 underwater videos)	20	Few-Shot Learning	Mean accuracy of (64.92 ± 14.55)% for k=30 shots on T0
Prasetyo et al. (2022)	Fish-gres, Fish4Knowledge	8, 23	MLR + VGGNet (VGG16 & VGG19)	Accuracy of 98.5% (Fish-gres) Accuracy of 97% (F4K)

Note. Comparison of various existing models to classify fish species in terms of dataset used for training, number of species considered, model architecture, and accuracy.

Literature Survey of Existing Research

This section covers an in depth review of various papers addressing our topic of interest or related field with advantages and drawbacks of each paper.

A new variation of algorithm for classification of images based on shape and texture on a large scale was developed by Zhang et al. (2006) by fusing KNN with SVM. Either of the algorithms individually have drawbacks as KNN is sensitive to outliers and SVM is relatively

better in image tasks due to their high dimensional data, combining both gives new capabilities to the model. This algorithm first computes K nearest neighbors in the training data and simply returns the labels if all neighbors belong to a single class. Else, it calculates and converts a distance matrix into kernel matrix for SVM based classification. They incorporated Directed Acyclic Graph based SVM for its speed for Multi class classification as a new version and evaluated on popular image datasets like MNIST, USPS, CUReT, Caltech-101. They have achieved a major improvement over plain baseline models (KNN and SVM) on Caltech-101 where the correctness rate increased from 40.98 to 59.08%.

Perronnin et al. (2010) in their paper discusses improvements on Fisher Kernel (FK) which is superior to Bag-of-Visual-Words (BOV). They also compare models trained on two different image datasets, ImageNet and Flickr with an aim to devise highly accurate computationally efficient image representation. One limitation of the BOV approach is their compute overhead as they take quadratic and cubic time complexities for feature extraction of image channels. Based on existing research, they proposed three improvements on top of FK - Normalization of values for decomposing images into background and specific information implicitly to capture higher order statistics of image, applying a power function for normalization, and applying spatial pyramids for dividing images into regions to improve discrimination between classes. With this setup, they have achieved a significant improvement in average precision from 47.9% to 58.3% on the PASCAL VOC 2007 dataset. They achieved state-of-the-art accuracy on Caltech 256 dataset with linear classifiers and Scale Invariant Feature Transform descriptors. They also inferred that training on Flickr dataset which has detailed annotations for images gave better results than ImageNet dataset that has just labels.

Peña et al. (2014) proposed an object based image analysis for classification of summer crops using satellite images. They implemented two different algorithms in hierarchical fashion and experimented with various combinations of Logistic Regression, Decision Tree, Multilayer Perceptron (MLP) and SVM. Initially, the textural features were obtained from satellite images using consecutive segmentation processes as used in eCognition software. These images were fed to two level hierarchical models where first level acts as narrow classifier in classifying Woody or Herbaceous crops and second level classifies specific crops based on parent category. In most cases, this model showed similar or better performance over each individual model but the improvement was in Minimum Sensitivity i.e., in predicting classes with lower accuracy. The combination of SVM and SVM in both levels got the best accuracy compared to all others. Also few models were compared with using only spectral features and using both spectral and textural features and the former proved to be equally productive or better than latter.

Liu et al. (2016) introduced an innovative approach known as the Single Shot MultiBox Detector (SSD). Unlike traditional methods that primarily focus on individual pixels of image features, the SSD method revolves around the application of a deep learning network. This network is constructed as a feed-forward convolutional network, emphasizing a unidirectional flow of information from the input layer to the hidden layers, thus avoiding the formation of feedback loops. Its initial layers are dedicated to high-quality image classification. Convolutional feature layers incorporating a variety of features are appended towards the end of the network for predictive purposes. The model's design represents a departure from recurrent or loop-based models. To validate the efficacy of the SSD model, extensive training was conducted on diverse datasets, including the PASCAL VOC 2007, VOC 2012, and COCO datasets. In a

direct comparative analysis with the well-established Faster R-CNN model, the SSD model delivered exceptional results with recall of 85-90%.

Knausgård et al. (2021) introduced a two-step deep learning approach aimed at revolutionizing temperate fish detection and classification This novel method combines two crucial components: the You Only Look Once (YOLO) algorithm for efficient object detection and a neural network empowered with a squeeze-and-excitation (SE) architecture for image classification. The YOLO algorithm, serves as the initial stage, focuses on rapid and precise object detection, swiftly identifying the presence of fish within images. Following this detection phase, the model transitions into the classification process, leveraging the neural network enriched with the SE architecture. The SE architecture operates in Squeeze and Excitation phase. In the squeeze phase, the architecture adeptly decouples the network's channels, thereby enhancing its ability to extract vital image features independently. The excitation phase plays a pivotal role in capturing channel-wise dependencies. Through the strategic deployment of reduction and activation layers, the network gains the capacity to discern intricate relationships between channels. To gauge the model's performance, extensive training was conducted on the Fish4Knowledge dataset, renowned for its significance in underwater image analysis. Impressively, the model showcased an accuracy rate of up to 83.68%, underscoring its proficiency in image segmentation tasks.

Although Deep learning models provide much better accuracy over traditional machine learning algorithms in image based tasks, they rely heavily on input data. The model requires a lot of training examples to better capture the underlying patterns within images and produce higher accuracy. However, this can be solved to some extent by usage of Transfer learning

techniques. Transfer Learning at its core refers to training our dataset on a neural network that is pre-trained on a huge database of images. This enables the model to capture relevant parts of the image with just a few examples and produce reasonably high performance. Agarwal et al. (2022) proposed an approach where they've used various pretrained models like VGG16, ResNet, InceptionResNet, DenseNet, InceptionNet etc.. since the size of the dataset is limited. The problem statement was to detect the dog breed from a given image into one of the 70 possible breeds. They split the dataset of 10000 images for training and testing in a 7:3 ratio averaging to 100 images per class and evaluated performance on metrics like accuracy, recall, precision and area under the curve. After experimenting with various optimization techniques, they concluded that InceptionResNet with AdaDelta optimizer outperformed all other models. Even with a smaller dataset they were able to achieve high accuracy of around 80% on unseen data with precision close to 82%.

Akbugday (2019) have taken three classifiers: Naive Bayes (NB), K-Nearest Neighbors (K-NN) and SVM, to investigate their accuracy. In this paper, the author has taken a breast cancer dataset from the UCI Machine Learning Repository in 1992. The author has used Weka software for classification. Three classifiers (K-NN, NB, SVM) are fed to the dataset and accuracy achieved for different classifiers was compared. For the K-NN classifier, the relation of accuracy for different K values is investigated. The highest accuracy is achieved for K=3. For NB classifiers, the accuracy obtained is almost 96 %. For SVM classifiers, we have taken C-SVM and nu-SVM. For C-SVM, results are computed for different cost(c), gamma parameters and kernel functions. Similarly, the accuracy is obtained for the nu-SVM classifier. In a nutshell,

the effectiveness of different classifiers is determined. From the results, we can conclude that 3-NN, C-SVM with C=2^15 are the most accurate classifiers with an accuracy of 96.85%.

Rodrawangpai and Daungjaiboon (2022) combined transformers with linear normalization for faster training and dropout layers to prevent overfitting of the data. The objective of the research was to improve text classification and use tip-over incidents from CPSC's NEISS database spanning the years 2010 to 2015. The dataset was divided into training, validation and test data. Training and Validation dataset were used to train the model. The proposed model was compared against multiple pre-trained models such as BERT, ALBERT, MPNet and DeBERTa. The model showed an AUC score of 0.95 and also showed an improvement of 11% when compared with pre-trained models. The additional layers increased the F1 score and recall by 4% - 7%.

Pan et al. (2020) proposed a new method for image classification using fine-tuned MobileNet. The model starts with a MobileNet architecture pre-trained on ImageNet, and then fine-tunes the weights and features on a welding defect dataset. To improve accuracy, the output of the pretrained model is connected to a defect classifier fully connected layer with 128 layers. The proposed model achieves 97.75% accuracy on the welding defect dataset, with lower computational cost than MobileNet, Xception and ResNet. Experiments found 96x96 image size gave better predictions than 128x128. The model is limited to offline defect detection and requires prior knowledge for selecting defect images to avoid misclassification. Future work focuses on improving detection, efficiency and online deployment.

Zhu et al. (2018) proposed a solution for insufficient training data for underwater object classification using the transfer learning method. SAS dataset is used, and CNN is trained to

classify three different shapes (Cylinder, truncated cone, sphere). All the images in the dataset are resized, and the grabcut algorithm is employed for image segmentation. The CNN is divided into two parts: the Convolution part and the Classification part. The trail dataset is first fed into the SVM model, achieving an accuracy of 77.55% (Cylinder-92.59%, sphere-88.89%, Truncated cone-38.46%). The trail is again fed into CNN, which achieved an accuracy of a total 82% (Cylinder-77.78%, sphere-77.78%, Truncated cone-100%). The simulated dataset is fed into the convolutional part of CNN and the trial dataset to the classification part, obtaining an accuracy of 91% (Cylinder-96.30%, sphere-92.31%, Truncated cone-77.78%), which is more than the original CNN. This is because the convolutional part is trained and used for feature extraction, which enhances the classification.

Wäldchen and Mäder (2018) emphasized the importance of accurate species identification in biological research and sustainable human development. Traditionally, species identification relied on morphological diagnoses provided by taxonomic studies. However, with a decreasing number of taxonomists, efficient and accurate identification through machine learning technology has become significant. The authors provided insights into applying machine learning techniques in species identification and introduced frameworks and software packages suitable for addressing identification challenges. CNNs have proven to be suitable for the majority of image-based species identification tasks. To improve CNN accuracy and robustness in such cases, there are strategies beyond augmentation and preprocessing. Architectures incorporating unique task-specific properties can yield better results than straightforward CNNs. One approach is employing Multiview networks, where cropping different parts of an image will provide additional information. Smaller-scale patches can reveal un-notable details of the

organism, like fish fin teeth in our case. In contrast, larger-scale patches can provide insights into the surrounding environment of the organism. Another approach is multi-scale networks, which adjust input sizes considering the required resolution and context to solve a problem.

Zhang et al. (2018) proposed a novel approach to small sample image recognition, leveraging an innovative hybrid model that combines Convolutional Neural Networks (CNN) with a General Regression Neural Network (GRNN). This model aimed to address the challenge of image recognition when faced with limited training data, a common predicament in various computer vision applications. The key innovation in this approach involves replacing the backpropagation neural network, typically used in traditional CNNs, with a GRNN. The resultant hybrid CNN-GRNN model uses an input layer characterized by an equal number of neurons, mirroring the dimensions of the learned instances. This input layer feeds into the pattern layer, tasked with discerning the disparities between the input data and the learned features. The model's output is determined through a pair of neurons in the summation layer. One of these neurons computes the cumulative output from all neurons, while the other calculates the aggregate sum of these neurons. Unlike traditional CNN models that often necessitate extensive iterations and a substantial volume of training data to perform effectively, the proposed CNN-GRNN model excels in small-sample scenarios. The researchers conducted an extensive evaluation of the model's performance on two distinct datasets: the Oxford-IIIT Pet Dataset and the Keck Gesture Dataset. The outcomes of this evaluation were striking, as the CNN-GRNN model consistently outperformed traditional CNN models and the CNN-SVM model in both accuracy of 97% and time.

Ghosh et al. (2019) discussed the intricacies of deep learning methodologies, with a primary focus on their efficacy in tackling constraints within the realm of computer vision (CV) tasks. The central objective of this paper is to provide an extensive analysis and a profound understanding of the concepts and techniques that play a pivotal role in addressing image processing and segmentation tasks. By using traditional techniques with cutting-edge developments in the field, the paper offers a holistic view of the evolution in image segmentation. To facilitate comprehension, the paper employs intuitive examples, enhancing the clarity of the concepts discussed, and thus making it accessible to a wide audience. These examples serve as practical illustrations of the theoretical frameworks, bridging the gap between theory and real-world application. Furthermore, the paper emphasizes the necessity of empirical analysis to augment the theoretical survey. By advocating for empirical validation of the proposed methods and techniques, the authors advocate for a practical approach to deep learning in image segmentation. This empirical dimension adds depth and real-world relevance to the insights and concepts presented in the paper. Table 5 contains the literature review of existing research around using deep learning techniques to solve similar problems.

Table 5

Summary of Literature Survey of Existing Research

Authors	Goals	Approach	Targeted Problem	Conclusion
Zhang et al. (2006)	Develop a new algorithm for large-scale image classification based on shape and texture	Fuse KNN and SVM algorithms through a novel distance matrix conversion	Overcome limitations of KNN and SVM	Correctness rate increased from 40.98% (baseline models) to 59.08%)

Authors	Goals	Approach	Targeted Problem	Conclusion
Perronnin et al. (2010)	Improve Fisher Kernel (FK) for efficient and accurate image representation	Three improvements on FK: value & power normalization, and spatial pyramids	Limitations of Bag-of-Visual-Words	Average precision increased from 47.9% to 58.3%
Peña et al. (2014)	Classify summer crops using satellite images	Hierarchical object-based image analysis with Logistic Regression, Decision Tree, MLP, and SVM	Classification of summer crops	Hierarchical models with SVM in both levels achieved the best accuracy.
Liu et al. (2016)	Introduce an innovative object detection model	Single Shot MultiBox Detector (SSD) - a feed-forward convolutional network	Traditional object detection focus on individual pixels, leading to low recall	SSD achieves recall of 85-90% outperforming Faster R-CNN
Knausgård et al. (2021)	Improve temperate fish detection and classification	Two-step deep learning: YOLO for object detection + SE-enhanced neural network for classification	Traditional methods lack efficiency and accuracy in underwater fish detection and classification	Model achieves 83.68% accuracy on Fish4Knowledge dataset
Agarwal et al. (2022)	Detect dog breeds from images with limited data	Transfer learning using pre-trained models (VGG16, ResNet, InceptionResNet, DenseNet, InceptionNet)	Deep learning models require large datasets, which can be limited for specific tasks	InceptionResNet with AdaDelta optimizer achieved 80% accuracy and 82% precision on unseen data with only 100 images per class

Authors	Goals	Approach	Targeted Problem	Conclusion
Akbugday (2019)	Investigate and compare the accuracy of different classifiers for breast cancer diagnosis	Used Weka software to compare Naive Bayes, K-NN, SVM	Selecting the most suitable classifier for breast cancer detection	3-NN and C-SVM with $C=2^{15}$ achieved the highest accuracy of 96.85%
Rodrawangpai and Daungjaiboon (2022)	Improve text classification using tip-over incidents data	Combined transformers with linear normalization and dropout layer	Traditional text classification methods are time-consuming and not scalable	Proposed model achieved an AUC score of 0.95 and outperformed pre-trained models like BERT by 11%. Additional layers increased F1 score and recall by 4-7%.
Pan et al. (2020)	Develop an accurate and computationally efficient method for welding defect classification	Fine-tune MobileNet pre-trained on ImageNet with a fully connected layer for defect classification	Welding defect detection and classification	97.75% accuracy on welding defect dataset, lower computational cost compared to other models, 96x96 image size outperformed 128x128
Zhu et al. (2018)	Address insufficient training data for underwater object classification	Transfer learning with CNN and SVM	Limited training data for underwater object classification	Transfer learning improves accuracy by using pre-trained features from simulated data for classification
Wäldchen and Mäder (2018)	Highlight importance of accurate species identification with modern approach	Introduce machine learning approaches	Species identification	CNNs are effective for image-based species identification

Authors	Goals	Approach	Targeted Problem	Conclusion
Zhang et al. (2018)	Address the challenge of image recognition with limited training data	Hybrid CNN-GRNN model	Small sample image recognition	CNN-GRNN outperforms traditional CNN models and CNN-SVM model in both accuracy and time
Ghosh et al. (2019)	Analyze and explain deep learning methodologies for image segmentation tasks	Combine traditional techniques with cutting-edge developments	Image processing and segmentation	Deep learning is a powerful tool for image segmentation, but empirical validation is necessary

Data and Project Management Plan

Data Management Plan

In order to build a robust model that is capable of classifying the species of fish, solely based on image in an underwater setting requires complicated neural networks that are trained on a huge dataset of fish images. This project uses a combination of DeepFish and Fish4Knowledge datasets for model building and evaluation. Both these datasets are observational datasets.

Data Collection Approach

The data does not contain any sensitive or personal information. The data collection process adhered to relevant ethical guidelines and regulations in the specified region. Collection methods were specifically designed to avoid disturbing the fish population and did not involve catching or harming any fish. In this project, the data will only be used for non-profit research purposes.

Fish4Knowledge. This dataset aims to facilitate research in marine ecosystems, monitor fish behavior, and create a database for environmental studies. The copyright is owned by the University of Edinburgh, it is open-sourced, available publicly, and intended for public use for research purposes. The dataset contains 27,370 fish images belonging to 23 classes. These images are collated from another public dataset of underwater video footage recorded over a 2 year timeframe around coral reefs off southeastern Taiwan.

DeepFish. This is a publicly available benchmark dataset created for research purposes. Governed by a Creative Commons Attribution 4.0 International License, this dataset allows for free usage, sharing, adaptation, distribution, and reproduction as long as proper citation is given to the original authors. The dataset includes around 40,000 images taken from 20 different habitats of coastal marines of tropical Australia (Saleh et al., 2020). Initially, several digital cameras are mounted on an underwater vessel and taken to the seabed to capture high definition videos (1920 x 1080 pixels) while maintaining a 100m range distance with the vessel. The videos were captured during daylight and at low turbidity points. A total of 39,766 frames were captured using this low disturbance technique. This method is highly efficient for capturing raw, unaltered images in in-situ environments. The collected dataset caters only for classification tasks but to make it feasible for segmentation, counting tasks etc., point-level annotations are made for every fish image around its centroid using the Labelme tool for object counting. Semantic segmentation labels are also created.

Data Management Method

The publicly available datasets will be stored in the Google Cloud Platform (GCP) for its seamless integration with other software and data storage. Once the datasets are uploaded to

cloud storage buckets, IAM roles are created for each team member with admin-level permissions to access that particular bucket and public access for that bucket is prevented as shown in Figure 2. This makes the storage unit secure and available only to this team.

Figure 2

Access to storage only to project members using gcloud user accounts and IAM role

Type	Principal ↑	Name	Role
	539682043524-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor
	devakumar.gajulamandyam@sjsu.edu	Deva Kumar Gajulamandyam	Owner
	jinthyswetha.mamillapalli@sjsu.edu	Jinthy Swetha Mamillapalli	Storage Admin
	sainath.veerla@sjsu.edu	Sainath Veerla	Storage Admin
	yuanting.li@sjsu.edu	Tiffany Li	Storage Admin

Note. Only Project Members Are Allowed To Assess The Dataset In Cloud Storage

Since the datasets do not contain sensitive data, there is no requirement to employ encryption techniques. However, we are using the standard Google managed encryption key for all the data within the bucket as a safety protocol. With the help of IAM roles the data is only shared within the group. Backup of necessary files will be taken at regular intervals to prevent accidental loss. For version control, an open-source tool called Data Version Control (DVC) integrates with GCP to track changes in the dataset.

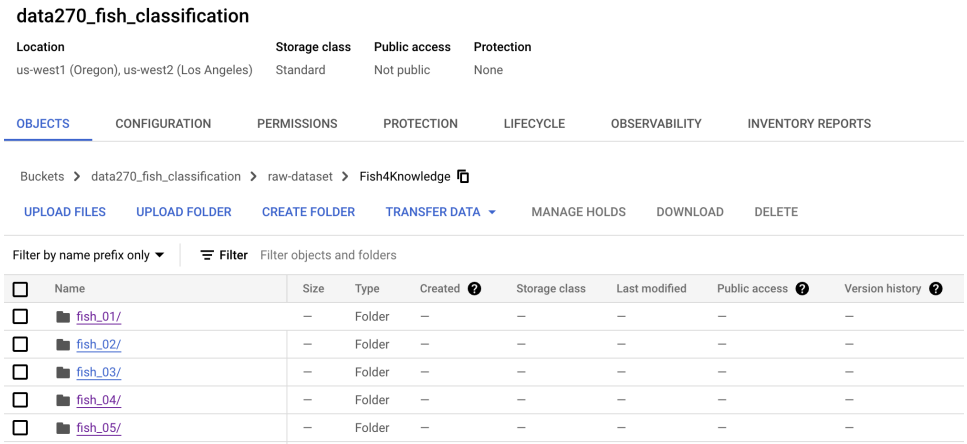
Data Storage Methods

Both the datasets together add up to 7 GB. Since the dataset size is huge, a low cost, centralized storage system is required to store and manage data across team members. Google Cloud Platform (GCP) offers Cloud Storage, an object based storage solution at very low cost and high scalability and availability. It also provides connectivity with several other platforms and tools making it a top choice for storage. Both the datasets are uploaded to a storage bucket as

the primary location from the cloud console. To combat machine failures, the storage is replicated across multiple regions as shown in Figure 3 for fault tolerance.

Figure 3

Data stored on Google Cloud Storage with replication



Note. Location column shows the two regions where the data is replicated.

Additionally, a copy of the raw dataset is saved in the local machines of team members as a backup. Different directories and subdirectories have been created to organize data and all the images belonging to a class are located in the same folder named as class name or code. The dataset folders have different naming conventions such as numbering. Metadata files with decoding of folder names and image names have been provided to align with images in the downloaded zip file.

As the raw data is publicly available and can be downloaded again anytime, it will be deleted after project completion. The processed data, including model input and outputs, is retained in GCP cold storage for one year to facilitate potential revisits and training of new models. Cold storage ensures cost-effective long-term data preservation. Access to the processed

data will remain consistent with the existing IAM strategy, restricting access to the project team members only.

Data Usage Mechanisms

All the team members are equally responsible for implementing the data management plan and ownership of the data is shared across the team. The intermediate data generated after data preprocessing and data engineering steps are stored in archival storage for future data discovery and retrieval. Compression techniques are applied to reduce the bandwidth of storage while archiving. The tools used for the data management plan are outlined in Table 6.

Table 6

Resources for Delivering Data Management Plan

Resource	Tool	Usage
GCP	Cloud Storage	Data Storage Platform
	IAM	Role and Access Control
	DVC	Version Control for Data
DMP Tool	DMP Template	Facilitate developing data management plan

Project Development Methodology

A project development methodology is a set of principles, tools, and techniques used to plan, execute, and manage projects. Common project development methodologies include the traditional waterfall and agile approaches. Waterfall methodology strictly follows a sequence of project phases, including requirements analysis, solution design, coding, integration, testing, and maintenance, with each step completed before moving on to the next. The output of one phase serves as the input for the next phase. Highsmith (2004) discussed that agile methodology is a

more iterative and incremental project management approach that emphasizes delivering products in short cycles and adapting to change quickly. It is suitable for small teams and close collaboration.

In the field of data science, CRISP-DM (Cross-Industry Standard Process for Data Mining) is a widely accepted standard for data science and data mining projects. Wirth and Hipp (2000) stated how the comprehensive framework covers all aspects of projects. For this project, the CRISP-DM methodology will be implemented with an agile approach to achieve the project's goals.

Business Understanding

Business understanding is the initial phase of CRISP-DM and the foundation of the project. It includes determining business objectives, assessing situations, defining data mining goals, and producing project plans. The first step is to identify the business objective. This involves understanding the significance, motivations, and goals of the project. It’s essential to collaborate with stakeholders to clarify and document these objectives. Once the business objective is established, conduct an assessment of the current situation. This involves conducting a literature review on the topic to identify gaps in existing research as well as examining the technology and data available that can support the research. With the insights gained from literature review helps refining the data mining goals. The goals should be aligned with the research objectives to ensure the project remains on target and delivers value. The Last step is to shape the project’s scope and generate business, software and hardware requirements. Document this information and produce a project plan.

Data Understanding

The data understanding phase starts with initial data collection, followed by describing data, exploring data and verifying data quality. For data collection, it is imperative to select a dataset that is not only relevant to the problem but also sufficiently large to train machine learning models effectively. The dataset also should be diverse and representative of the real-world situation. It is essential to read the metadata and understand any instructions, policies, or rules provided by the data owners. Next, we configure Google Cloud Platform for storage and modeling purposes. We analyze and study the target classes of dataset in Google Colab for collaboration and to stay up to date on the tasks. Data is visualized and patterns in the dataset are recognized using exploratory data analysis this give a clear understanding of the datasets

Data Preparation

Data preparation is the process of transforming raw data into a format that is compatible with machine learning models. Data quality and authenticity is verified by checking for missing values, inconsistencies and outliers. Based on insights from data exploration, data cleaning can be done by computing missing values using appropriate methods and removing outliers. For image data, clean data also includes resizing or cropping the images to a consistent size. Augmentation tasks such as flipping, rotating or cropping of images are performed to handle the imbalance in target variables. Data from multiple sources is combined into one single dataset and resolves any conflicts between image data from different sources. Then the images are enhanced and noise is removed for better clarity and understanding of the images. Next, normalize data so that all features are on the same scale and reduce data dimension if necessary. Once the images

are uniformed we select relevant data, using autoencoders. Finally, split the data into training, validation and testing sets for model training and model prediction.

Modeling

In the modeling phase, various machine learning models are selected and applied. To first select modeling techniques, the strengths and weaknesses of machine learning models are compared to choose the most suitable model for data mining goals, data type, and desired accuracy levels. Next, feed the data into the selected model and allow it to learn.

Convolutional Neural Networks (CNN) has inbuilt feature extraction capabilities and data augmentation techniques, especially valuable when dealing with imbalanced datasets. MobileNet algorithm works well for less memory usage, making it adaptable to resource-constrained environments. Vision transformers can identify patterns in low-vision images and uncover long-term dependencies within them. This is a new technique compared to other approaches. InceptionResNet is a combination of Inception and ResNet models. This fusion provides better performance.

Kühn and Johnson (2013) discussed that the results of initial model testing can reveal problems in data. If model performance is lower than expected, the data errors might be exposed. By checking the structure of the model, the set of features that the model depends on is not what it expects, so revisit the data to check if these features are correctly selected. It is common for a project to go through multiple iterations between the modeling and data preparation phases. These models are run in Vertex AI on GCP which gives users an option to add additional GPU which helps in faster construction of the model.

Once the data and model selection are determined, there is a repetitive process of training the model, fine-tuning hyperparameters to enhance performance, and evaluating the model's results. The best sets of hyperparameters are selected, and choose the optimal model through evaluation. This iterative approach is instrumental in achieving the desired level of model accuracy and effectiveness.

Evaluation

In the evaluation phase, the results from the modeling phase are assessed against the established business success criteria. This phase involves approving models, and determining the next steps. To the business criteria, model accuracy, performance, robustness, computational efficiency, and interpretability are considered to determine whether the model can solve the research problem. After the results are evaluated, a comprehensive review of the project processes, including data preparation, model selection, and hyperparameter tuning is conducted to identify areas for improvement. These can be done by documentation review, consultation with experts and error analysis. In this phase, the entire process is reviewed and additional ideas are added for betterment of the project. Based on these findings, a decision is made to deploy the model or to iterate the CRISP-DM process again to create more suitable models.

Deployment

In the deployment phase, the approved machine learning models are ready for deployment to production applications. The deployment process includes planning deployment, monitoring and maintenance, and producing a final report.

Planning deployment includes developing a deployment plan, preparing the deployment environment, packaging the model, testing the model in a pre-production environment, and

releasing it to users. The model should also be continuously monitored to ensure stability. And define monitoring metrics aligned with business goals, such as accuracy, precision, and latency. At the end of the CRISP-DM process, summarize the project into a report and presentation and communicate with stakeholders. Share project experience and lessons learned with peers.
The following section will discuss the use of Work Breakdown Structures (WBS), PERT, and Gantt charts to organize, plan, and track a project.

Project Organization Plan

The project organization plan describes the structure, roles and responsibilities of the project team. According to Siami-Irdemoosa et al. (2015), A WBS is the process of dividing a project's overall work into several more manageable hierarchy-structured tasks. The level of detail should represent the overall scope of the project while keeping the tasks manageable. The WBS is typically designed through a top-down procedure. The upper levels of the WBS are decomposed into logical groupings of work, followed by the next level down, and so on. For this research project, the CRISP-DM framework is integrated to create a WBS by decomposing each phase into smaller tasks.

To facilitate this process, Jira was extended with an additional plugin “Move and Organize” to create the WBS. The project theme is “Deep Learning Based Fish Species Classification.” The first level of the WBS corresponds to the six phases of CRISP-DM: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. These phases are defined as epics in Jira. Each phase is further broken down into manageable tasks and subtasks to ensure a comprehensive and organized project management approach.

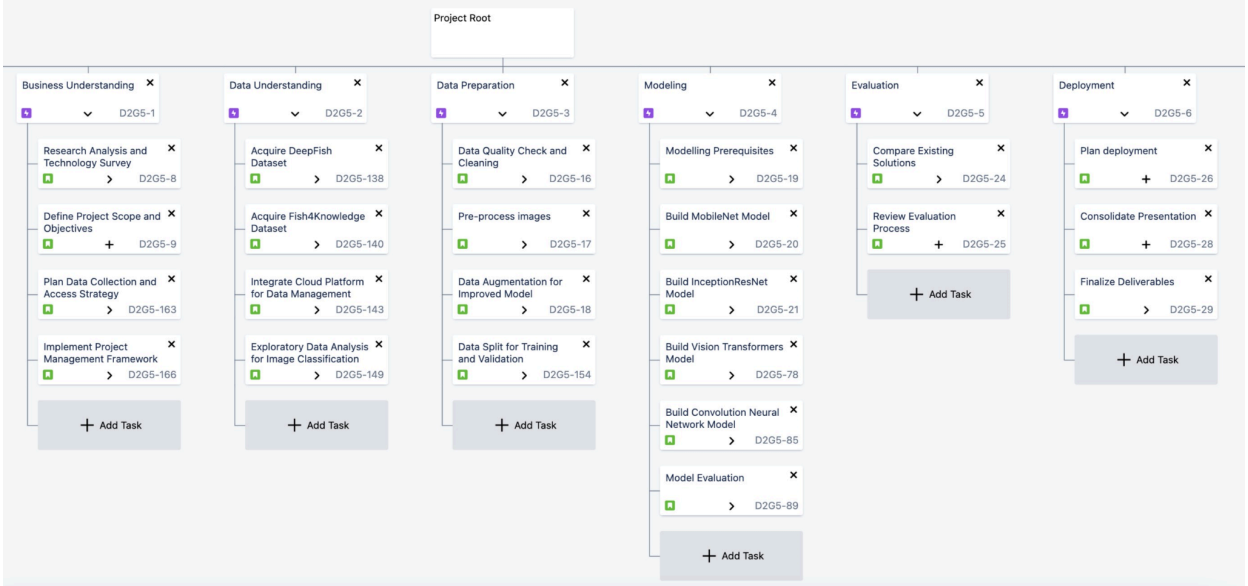
In WBS the first phase is Business understanding where the project requirements are discussed and intense research is done on the field of the project to determine its scope and functionality. Additional research helps in finding the limitations of existing solutions giving a jist of what is to be implemented. Based on research a plan for data collection and framework on handling the entire project are designed. The data collection plan is thoroughly followed aiding in gathering of data for the project. In the next phase the data is studied to get a clear understanding and to align with the project goal. The main idea of this phase is to perform an in-depth analysis such as different data types in the data and distribution of the collected data upon these insights the next required actions will be taken meticulously. The data is stored in GCP for easy access within the team. Following the data understanding phase there is data preparation phase.

In this phase pre-processing, data transformation, feature extraction, dimension reductionality are performed such that the model can adjust to the given data and draw better insights. Missing data, imbalance data and other issues are resolved in this phase and the data is split for train, test and evaluation. The data is now ready for the next phase modeling. This is an integral part of the WBS as machine learning models are built in this phase. Machine learning models to be used are decided based on how they tackle the problem. The models are built on test data in Vertex AI environment enabling a smooth scaling for computational resources. After training the model its performance is evaluated using different metrics such as accuracy, precision, root mean square error etc. based on the problem. For better performance the models are optimized and hyper tuned by changing parameters or adding new functions. In the evaluation phase the models are compared against existing solutions. In the final phase the

model is deployed using an interface for better interaction and understanding of the user. Consolidated reports detailing the entire project along with individual reports on machine learning models are finalized. Figure 4 shows the WBS structure deployed on Jira

Figure 4

Work Breakdown Structure



Note. WBS Depicting Six Phases In The Project

Project Resource Requirements and Plan

Hardware Requirements

The project involves training machine learning models on large image datasets. For such operations it is necessary that the computational demands are handled and chosen as per the requirement. Appropriate selection of the model helps in better performance and optimal time for the model to run. The below Table 7 represents the hardware requirements (CPU, GPU, Storage) of the project. In order to accelerate model training, we make use of GPUs provided by Vertex AI platform in GCP. Local machines are made sure to have at least 8GB of RAM, intel i7 CPU

processor, 256GB of storage to work with documentation, communication and other tools. For better performance of model NVIDIA Tesla A100 GPU has been used on the cloud.

Table 7

Hardware Requirements and usage

Hardware	Specifications	Justifications	Cost
CPU	Intel i7	High performance processor,ensures rapid execution of algorithms.	$800
Storage	256GB	Ample space to store large datasets	$0.00
RAM	16GB	Rapid data access and large datasets can be loaded into memory without disk swapping	$0.00
Network Bandwidth	4G/5G Bandwidth	Provides rapid data transfer	$0.00
GPU	NVIDIA Tesla A100	Faster execution of machine learning models	$3.4/hr

Software Requirements

The project also necessitates software requirements for the functioning purposes. It is important to ensure that the softwares is compatible with hardware and different environments for smooth execution of the project. It is also necessary to have knowledge on the libraries used in the project such that any additional requirements can be dealt with without any major changes. Windows 11 is chosen as it provides a robust graphical user interface (GUI) which helps simplify the interaction between hardware and software components. For communication and team coordination, the Zoom platform is used. For speed and advanced features latest versions of chrome and Safari were utilized. Python 3.10 ensures versatility and it is widely accepted in the data science community and anaconda distribution complements python which provides a

compatible platform for all the processes. Libraries such as Numpy 1.26 and Pandas 2.1.2 are helpful for data manipulations and OpenCV is essential for image processing. For model development, TensorFlow and Keras 2.13 along with PyTorch 2.10 are chosen, which are forefront libraries for developing machine learning models. Table 8 showcases different softwares, specifications along with their purpose concerning the project and their cost.

Table 8

Softwares and versions used and their usage

Software	Specifications	Usage	Cost
Operating System	Windows 11	To provide GUI for hardware and manage resources for tasks	$600
Zoom	5.16.6	Sprint meetings to provide updates and ensure alignment on assigned tasks	Free
Chrome/Safari	Latest Version	Data research and working with tools	Free
Python	3.10.0 or above	Programming language for data processing, cleaning and preparation	Free
Anaconda Distribution	Anaconda 3	Platform for data processing and model building	Free
Numpy	1.26	Library for image pixels as matrices and mathematical operations	Free
Pandas	2.1.2	For data manipulation using data structures	Free
Plotly	5.18.0	Create interactive data visualization	Free
Tensorflow and Keras	2.13	Machine learning library for building and training models	Free
PyTorch	2.10	Machine learning library for model building	Free
OpenCV	4.8.1	Perform image processing	Free

Software	Specifications	Usage	Cost
MS Office	MS Office 2019	For documentation and reports	$0.00

Tools and Licenses

Along with hardware and software requirements the project needs other licenses for better understanding and performance of the project. The team should ensure that the licenses are up to date and renew any license that is expired or about to expire. By keeping software licenses and subscriptions current will ensure the team has consistent access to the resources needed for development. Most of the tools mentioned are free and few of them such as GCP have fees based on usage. Google services such as cloud storage, colab, docs and Vertex AI have been used for storage, running python scripts, documentation and building models respectively allowing shared access to all team members and updated content. Whatsapp and Gmail Have been used for communication across the team for sprint calls and updates on the assigned work. For agile methodology implementation Jira software has been leveraged. Along with these tools draw.io and lucid charts have been used for better visual representation of data flow and other diagrams. Table 9 shows details of different tools used along with their use case.

Table 9

Table describing Tools and Licenses used

Tool	License	Justification	Cost
Google Cloud SDK	Free	To establish connection between local and remote GCP platform via CLI	$0.00
Jira	Free Tier Usage	To set up project management plan and track project tasks and issues	$0.00
Lucid Charts	Free Tier	To prepare PERT Chart and workflows	$0.00

Tool	License	Justification	Cost
Draw.io	Free	Data flow and visual representation of concepts	$0.00
DMPTool	Educational Free Tier	To construct data management plan	$0.00
Google Colab	Free Tier	To perform data preprocessing, modeling etc.	$0.00
Google Drive - Docs, Sheets	Free	For documentation, report, shared access to team members	$0.00
WhatsApp	Free	For unofficial communication within the team	$0.00
GMail	Free	Email communication and updates on tasks	$0.00
Cloud Storage (GCP)	Paid	Storage and manage data in GCP	$0.473/GB /month
Vertex AI (GCP)	Paid	Leverage ML capabilities for model development	$3.4/hr

Specifications, Costs and Justification

The project incurs a cost of $800 for the laptop which included CPU, RAM and storage and $52 bill accrued by google cloud platform for storage and GPU compute. The entire cost sums to $852 which is reasonable considering the scale of the project has been mentioned in table 10. This information is necessary for project budget planning.

Table 10

Costs and Justifications

Product	Cost	Justification
Google Cloud Storage	$0.473/GB per month	To create a highly available, fault tolerant storage solution
Vertex AI	Pay-as-per-use	Provides GPU run time to accelerate model training and pricing is based on GPU used
Laptop	$800	Laptop with required configurations is procured

Project Schedule

The work breakdown structure provides a high-level overview of the tasks associated with the six phases of CRISP-DM. Based on WBS, PERT charts, Gantt charts are used to further define the project timeline and task dependencies, which helps track project progress and ensure that the project is completed on time within budget.

Gantt Chart

A Gantt chart is a visual representation of a project schedule in the form of a horizontal bar chart. It is often referred to as a roadmap. Jira has a built-in timeline, but the WBS Gantt Chart plugin can be used to create more detailed and customized Gantt charts. The plugin is synchronized with Jira tickets and displays epics, user stories, and subtasks in a hierarchical view, along with the assignee, status, start date, end date, and duration for each task. The bars on the chart represent the timeline for each step, providing a clear roadmap for the project. The timeline and task dependencies are updated as needed.

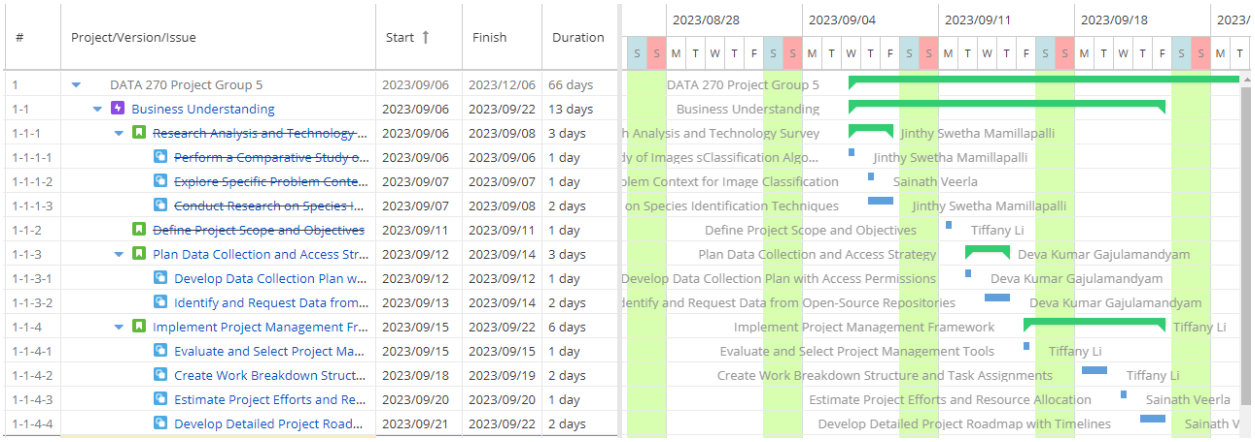
The project is developed in an agile approach. Before the project starts, tasks distribution and effort estimation are aligned in the project team. Each task and its duration are assigned based on task complexity, the experience and skills of the individual, and the urgency of the task. The effort for each person is equal. As the project progresses, the team becomes more familiar with the project and the technology. Task assignments are continuously adjusted based on feedback from team members.

In the first sprint i.e., Business Understanding we research on problem statements and current technology being used. This gives us an overview of the entire project and directs us on how to proceed further. Research also helps in finding the shortcomings of existing solutions and

to come up with solutions that surpass the boundaries. In this sprint, we also create a data collection plan where we look for answers to questions such as how to collect data, points considered when selecting a dataset.The following figure 5 shows the Gantt chart of the business understanding phase which only counts workdays and does not consider weekends. The plan also includes finding efficient data flow and data storage methods. The end goal of the sprint is to have clear background research on fish species and image classification along with tasks that are to be performed.

Figure 5

Gantt Chart - Business Understanding Phase

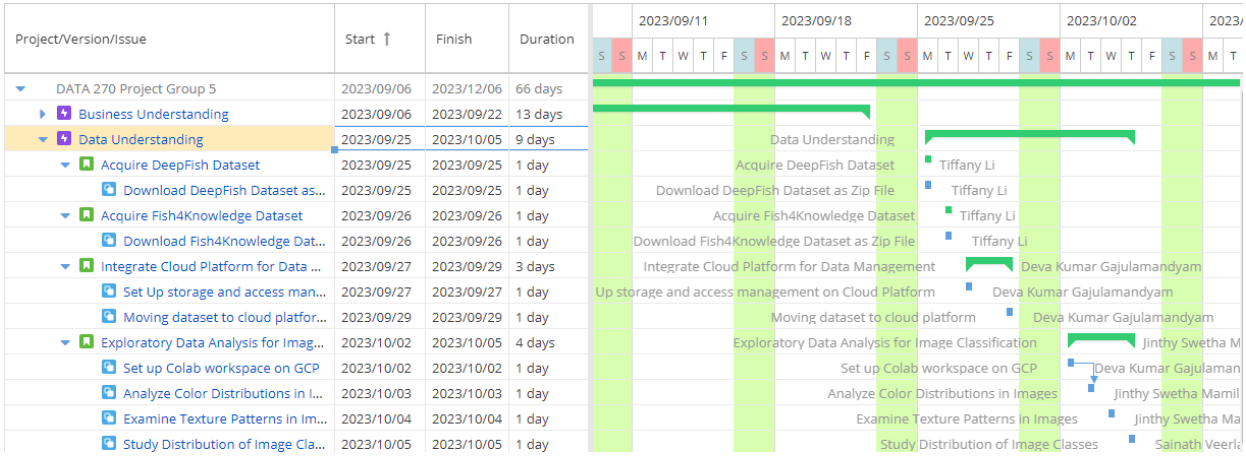


Note. Detailed Representation Of Business Understanding Phase In Gantt Chart.

In the second sprint, we move to the data understanding phase. Based on insights from the previous phase we have concluded to download DeepFish and Fish4Knowledge datasets from open source websites due to their coverage of a wide variety of fish species. In this phase we understand the data and also look for patterns in the dataset. Figure 6 shows a Gantt chart representation of the data understanding sprint.

Figure 6

Gantt Chart - Data Understanding Phase

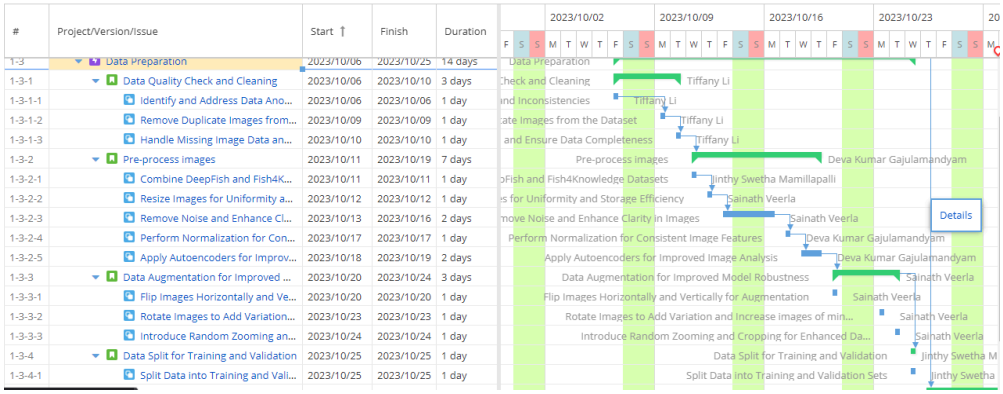


Note. Detailed Gantt Chart built on Jira demonstrating Data Understanding Phase

In the third sprint we proceed with data preparation where we initially check for anomalies, redundancy and missing values. All the issues are handled accordingly and transform the data into a complete dataset. We then combine the datasets into a single dataset. Then noise is removed from the images enhancing their quality. All the pictures are brought down to a desired resolution to maintain uniformity in the entire dataset. The features in the dataset are filtered using autoencoders giving importance to selective features which help in increasing the prediction accuracy. Data augmentation is performed to remove any imbalance in the dataset. Finally, the dataset is split for model training and model evaluation. In this phase not only we infer insights from the data but also transform the data in each subtask therefore all the tasks are dependent on previous tasks. Figure 7 shows a Gantt chart representation of the data preparation phase.

Figure 7

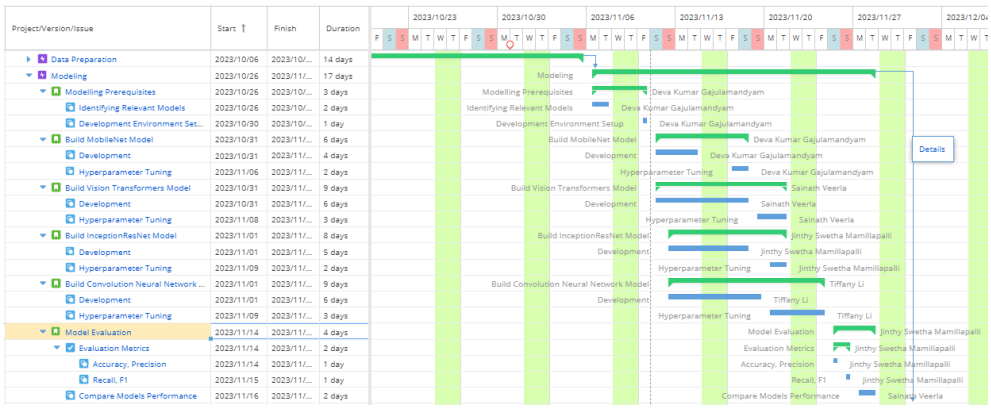
Gantt Chart - Data Preparation Phase



In the Data Modeling phase, we implement the proposed algorithms and train the models. We use notebooks in Vertex AI which comes with integrated CPU and GPUs which can be customized for faster inference of models. The pre-trained models from keras are imported and then changes are made accordingly to meet the project requirements. Once the training is completed we evaluate the model based on accuracy, precision, F1-score to check the better performed model. Figure 8 shows a gantt chart representation of the data modeling sprint using Agile.

Figure 8

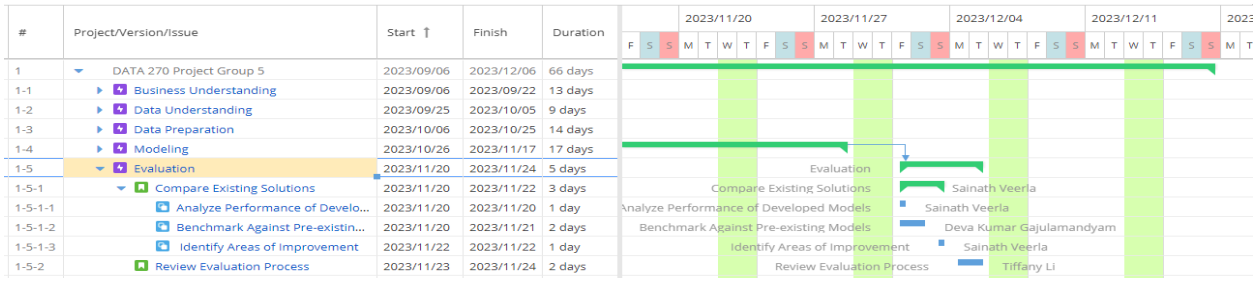
Gantt Chart - Data Modeling Phase



In the Evaluation phase, we take the best performing model out of four models that we trained and compare its performance with existing approaches discussed in the Business Understanding phase. Further, we identify areas of improvement and include potential future scope to enhance this work. Figure 9 shows a gantt chart representation of the evaluation phase.

Figure 9

Gantt Chart - Evaluation Phase

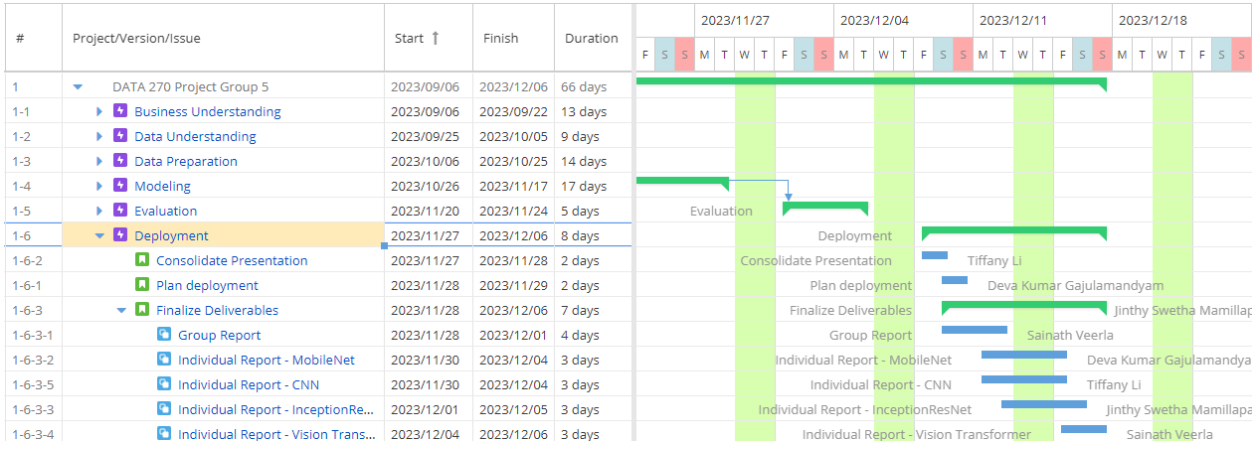


#	Project/Version/Issue	Start	Finish	Duration
1	DATA 270 Project Group 5	2023/09/06	2023/12/06	66 days
1-1	Business Understanding	2023/09/06	2023/09/22	13 days
1-2	Data Understanding	2023/09/25	2023/10/05	9 days
1-3	Data Preparation	2023/10/06	2023/10/25	14 days
1-4	Modeling	2023/10/26	2023/11/17	17 days
1-5	Evaluation	2023/11/20	2023/11/24	5 days
1-5-1	Compare Existing Solutions	2023/11/20	2023/11/22	3 days
1-5-1-1	Analyze Performance of Develo...	2023/11/20	2023/11/20	1 day
1-5-1-2	Benchmark Against Pre-existin...	2023/11/20	2023/11/21	2 days
1-5-1-3	Identify Areas of Improvement	2023/11/22	2023/11/22	1 day
1-5-2	Review Evaluation Process	2023/11/23	2023/11/24	2 days

In the final Deployment phase, all the materials are aligned by the team members. After evaluation and reviewing of the process we deploy the model. The entire process performed up until then is clearly documented and reported including the group and individual reports. Figure 10 shows a gantt chart representation of the deployment phase sprint.

Figure 10

Gantt Chart - Deployment Phase



#	Project/Version/Issue	Start	Finish	Duration
1	DATA 270 Project Group 5	2023/09/06	2023/12/06	66 days
1-1	Business Understanding	2023/09/06	2023/09/22	13 days
1-2	Data Understanding	2023/09/25	2023/10/05	9 days
1-3	Data Preparation	2023/10/06	2023/10/25	14 days
1-4	Modeling	2023/10/26	2023/11/17	17 days
1-5	Evaluation	2023/11/20	2023/11/24	5 days
1-6	Deployment	2023/11/27	2023/12/06	8 days
1-6-2	Consolidate Presentation	2023/11/27	2023/11/28	2 days
1-6-1	Plan deployment	2023/11/28	2023/11/29	2 days
1-6-3	Finalize Deliverables	2023/11/28	2023/12/06	7 days
1-6-3-1	Group Report	2023/11/28	2023/12/01	4 days
1-6-3-2	Individual Report - MobileNet	2023/11/30	2023/12/04	3 days
1-6-3-5	Individual Report - CNN	2023/11/30	2023/12/04	3 days
1-6-3-3	Individual Report - InceptionRe...	2023/12/01	2023/12/05	3 days
1-6-3-4	Individual Report - Vision Trans...	2023/12/04	2023/12/06	3 days

PERT Chart

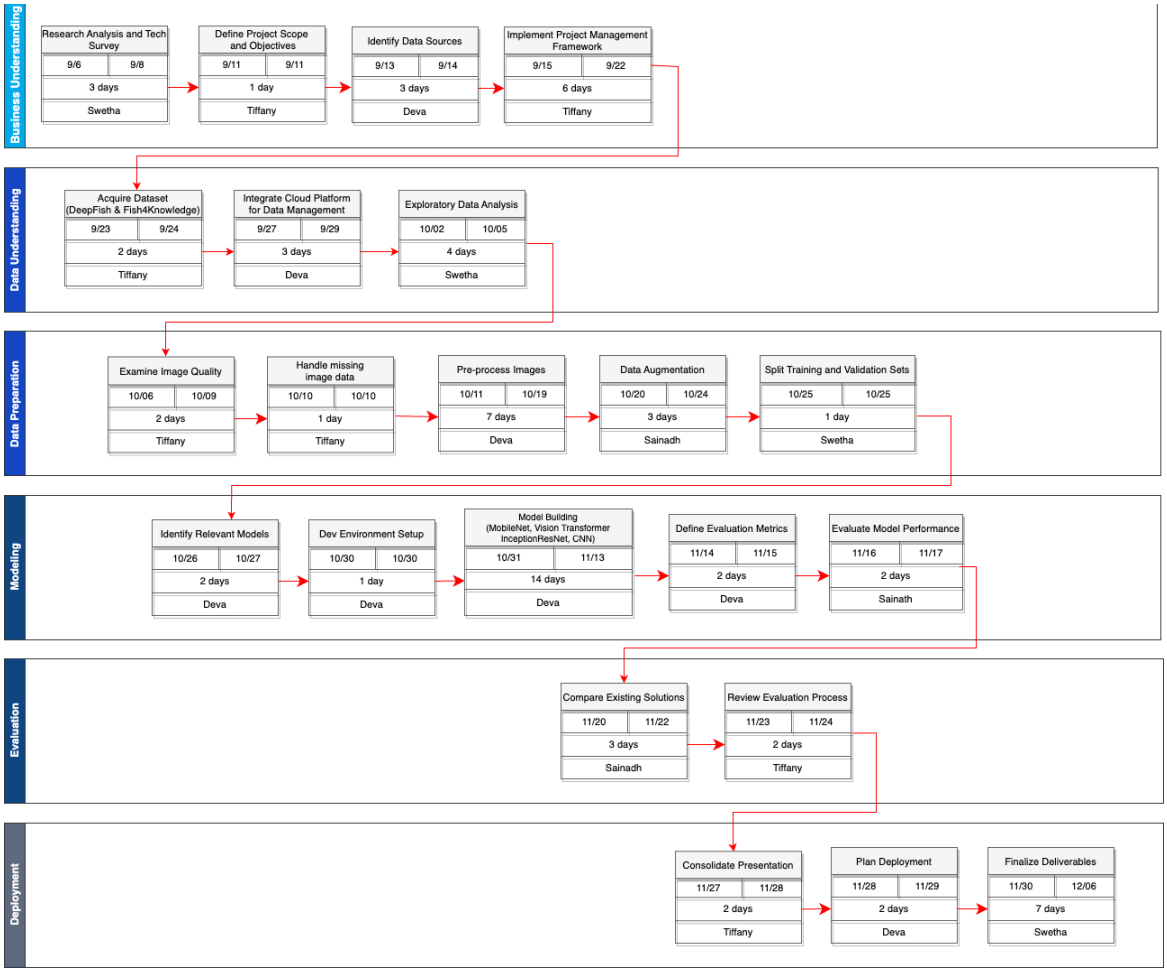
A PERT chart (Program Evaluation and Review Technique) is a visual project management tool used to map out and track the tasks and timelines of a project. It identifies the critical path and therefore estimates the minimum time required to complete all project tasks. PERT charts come in different forms based on project complexity and the level of detail they provide, such as milestone-oriented PERT chart and task-oriented PERT chart. PERT method is utilized in managing task sequences and to handle dependencies within the project to enhance project management and on-time delivery (Vasudevan et al., 2020).

The PERT chart creation process begins with identifying all tasks and estimating the duration of each task. Next, the dependencies between tasks are identified. A network diagram is then created, with nodes symbolizing tasks and arrows denoting task dependencies. This allows for the calculation of the critical path and overall project duration.

The following figure 11 shows the PERT Chart created by draw.io based on tasks in Jira. It has six lanes, each corresponding to one of the six phases of CRISP-DM. Each block represents a task with details like task name, task ID, duration, start and end dates, and assignee. The lines represent the dependencies. The critical path in red shows the sequence of tasks that must be completed on time in order for the project to be completed on time.

Figure 11

PERT Chart



Note. PERT Chart Depicting The Critical Path Of Sequence Of Tasks

Data Engineering

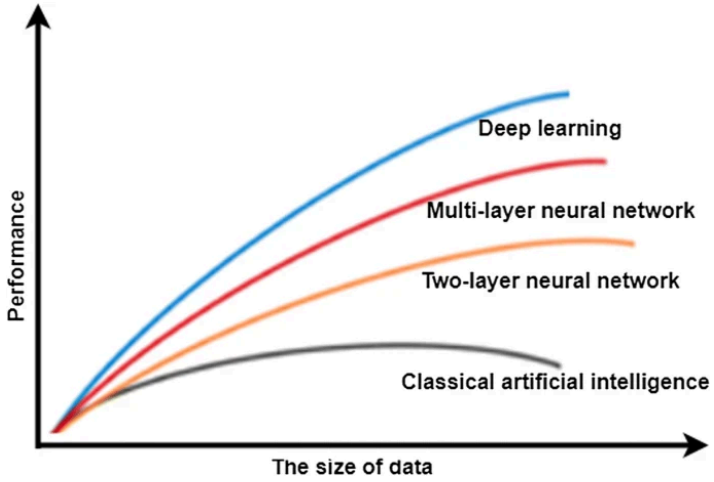
Data Process

As Dorfman (2022) mentioned, the more data there is, the better for deep learning and machine learning tasks. Based on the type of problem at hand, the quantity of raw data required to build an accurate model can be estimated. While there are a lot of factors that influence the model itself like quality of data, complexity of learning algorithms, etc., it's always good to have

sufficient data. The performance of deep neural network based models increases as the size of data increases as shown in figure 12 taken from Serin et al. (2020).

Figure 12

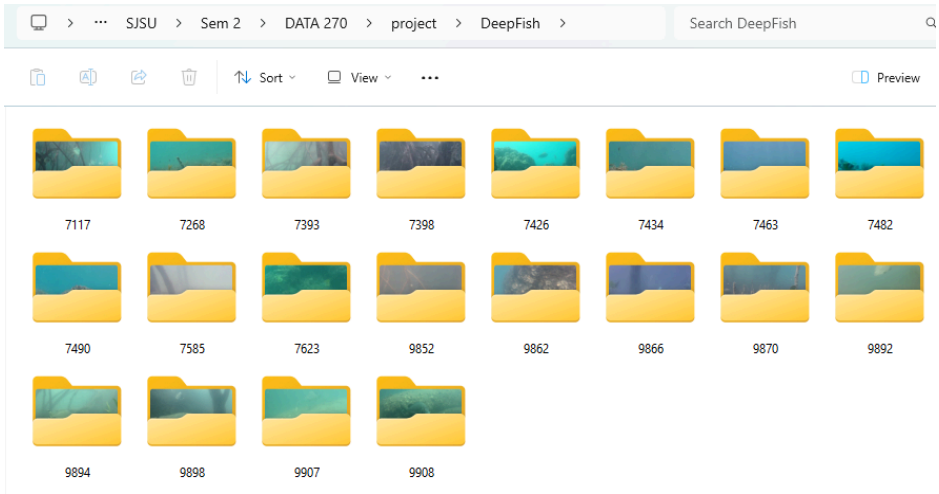
Performance of neural networks based on training data volume



Two distinct datasets, DeepFish and Fish4Knowledge, are gathered and used for training and testing to account for the enormous amount of raw data. When combined, the two databases have over 44,000 photos from various courses. Images in DeepFish are organized based on the habitats they are gathered whereas in the Fish4Knowledge dataset, they are arranged into folders according to the species of fish. It is difficult to execute operations on data because of this irregular positioning of data. Figure 13 shows the organization of images within the DeepFish dataset where images are grouped based on region codes where the data is collected. Figure 14 shows Fish4Knowledge images grouped by species name.

Figure 13

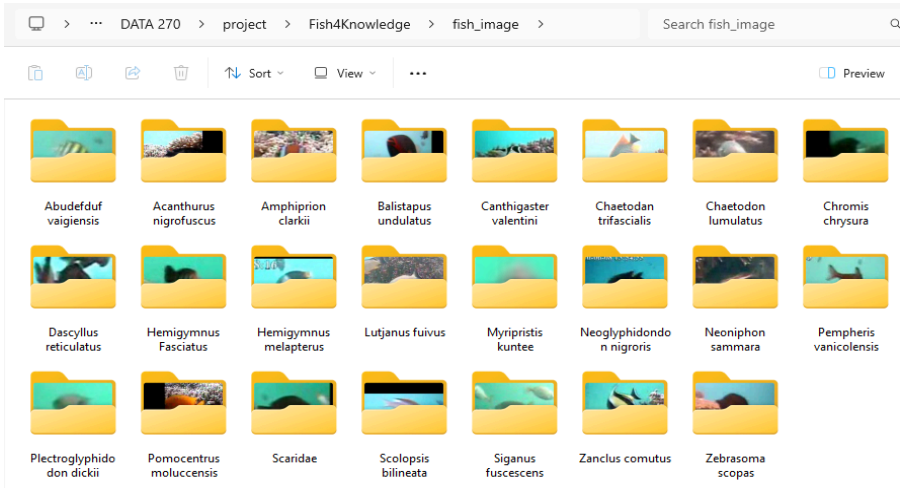
Organization of images in DeepFish database



Note. Each folder name represents the region code where samples were collected.

Figure 14

Organization of images in Fish4Knowledge database



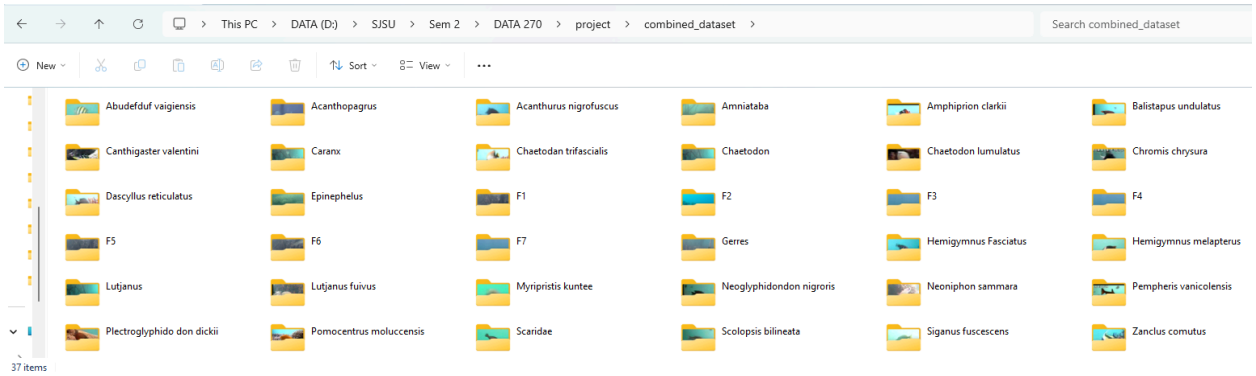
Note. Each Folder Represents A Class Of Fish

This data is preprocessed to extract species names from images in DeepFish and segregate these images into folders with each folder named after a species, containing all the photos of that species, to make them uniform. All these folders are stored under a single

directory as shown in Figure 15. This directory is uploaded to Google Cloud Storage and Drive for further data engineering and modeling.

Figure 15

Organization of combined dataset based on class names



Note. Combined Dataset Containing 37 Classes Of Fish Species

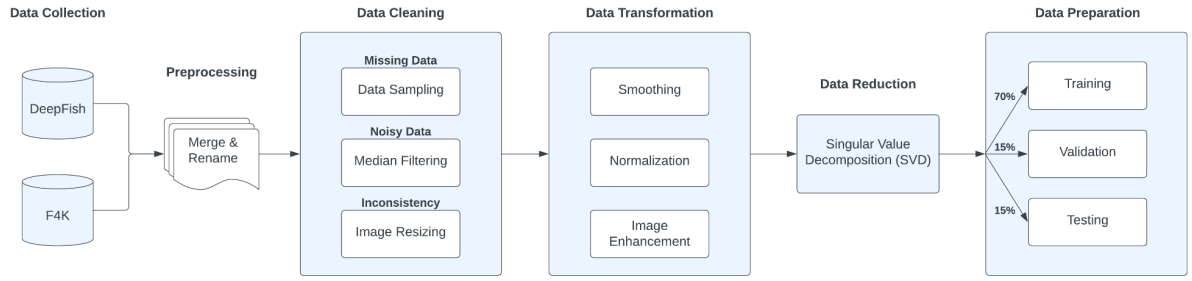
After the preprocessing step, data cleaning and transformation steps are performed. In their paper, Obaid et al., (2019) mentions that data cleaning and transformation is an important step before modeling as this helps in the overall accuracy of the model. In order to rectify bad images, convert inconsistent data types in the raw dataset, data cleaning is required. To achieve this, several techniques like Oversampling and Undersampling, Median based filtering, resizing images to a consistent dimensions are performed to handle missing data, noise in the data and inconsistencies within raw data respectively.

In the data transformation stage, all these images undergo a series of operations like Image Smoothening, Pixel Value Normalization, Image Enhancement with brightness and contrast adjustment, Dimensionality Reduction using Singular Value Decomposition (SVD). These steps help in eliminating noise from images, attain better representation of images to optimize model training and lowering data reduction.

On top of this data, summary statistics are done and few visualizations are built to better understand distribution of data. In the next step, the whole dataset is split into three disjoint subsets for training, validation and testing in the ratio of 70:15:15. The training and validation data is used for training the model recursively and the performance is measured on test data. This entire pipeline is represented in figure 16.

Figure 16

Data Process Pipeline



Data Collection

Data Collection Requirements

The following proposed data collection methods are based on the work of Fisher et al. (2016) and Saleh et al. (2020). In order to avoid data bias, the dataset will be collected from habitats across different regions to ensure it is representative of the diversity of fish habitats. By cooperating with institutions with common research objectives, two sites in Taiwan and Australia were selected to collect data. The primary approach for data collection is deploying underwater cameras, which not only provide a safer approach for humans but also enable continuous data gathering without disturbing fish populations.

To ensure a sufficiently large data volume, the target dataset size is approximately 500,000 videos from the Taiwan site. For the Australia site, the target dataset size is

approximately 40,000 video frames. These videos will undergo image extraction to generate a corresponding image dataset for training and evaluation purposes. The image extraction process will involve carefully selecting and extracting frames from the videos that clearly show fish in their natural habitat. By combining the extracted image datasets, the goal is to end up with around 45,000 images for the fish species classification task.

The fish image data collection is essential for monitoring fish populations, understanding fish behavior, and assessing the impact of human activities on fish habitats. For this project, the objective of the data collection is to gather data for training machine learning models focused on fish species classification. The plan is to collect image and video footage from different marine habitats in tropical Australia and Taiwan. The data collection will be carried out by using arrays of cameras mounted on metal frames on underwater vessels and deployed in the water capture fish communities. These videos will be further processed to identify and label the fish species encountered. The frames which capture fish are extracted from the video and grouped into appropriate directories. The raw image data requires a manual selection of well-composed fish photos and cropping them to have a single fish at the center, avoiding occlusions between fish and background elements. The selected images will be resized to a consistent size and labeled with the fish type and location. A new directory is created for each type of species and all the images containing that particular species are copied to this directory. On this data, preprocessing, noise removal, data quality checks are performed to ensure reliability and uniformity of data. Data Transformation steps like regularization, reduction by PCA are done.

Table 11

Data Collection Plan

Key Variables			
What?	Variable Title	Image	Class Name
	Input (X) or output (Y) variable	Input	Output
	Unit of measurement	Pixels (for video resolution) Meters (for depth and distance) Frames (for number of video frames taken)	Text
	Data type	Image Data	Text / String
	Collection Method	Video Recording	Human Annotation
	If manual	No	No
	Gauge / Instrument	Camera, Acoustic depth sensor, GPS	N/A
	Location	Taiwan, Australia	
MSA	Gauge liberated?	Yes	Yes
	Measurement System checked?	Yes	Yes
	Precision (R&R) adequate?	Yes	Yes
	Accuracy adequate?	Yes	Yes
Historical data	Historical data exist?	No	No
	Source of historical data	N/A	N/A
	Historical data representative or reliable?	N/A	N/A
	Mean	N/A	N/A

Key Variables			
Historical Data	Upper specification limit	N/A	N/A
	Lower specification limit	N/A	N/A
	Standard deviation	N/A	N/A
	Target	N/A	N/A
Sampling	Minimum sample size (MSS)	241 images per class	
	Sampling Frequency	Daily	
	Sub-grouping needed?	No	
	Sub-group size	N/A	
	Stratification needed	No	
Who?	Data Collector	Taiwan Site: University of Edinburgh, Taiwan Ocean Research Institute and Kenting National Park Australia Site: Saleh, A et al. (2013) at James Cook University	
	Operational definition exist?	Yes	
	Data collectr trained?	Yes	
	Resource available for data collector?	Yes	
When?	Start date	Taiwan Site: October 1, 2010 Australia Site: 2012	
	Due date	Taiwan Site: September 30, 2013 Australia Site: 2013	
	Duration (in days)	Taiwan Site:1095 Australia Site:365	

Dataset Samples

Fish4Knowledge. For data collection at the Taiwan site, data was collected in collaboration with the University of Edinburgh, the Taiwan Ocean Research Institute, and Kenting National Park. The project was named "Fish4Knowledge" and planned for three years, from October 2010 to September 2013. From the preview of the collected data, the video quality is good. The footage recorded various marine life such as fishes, corals, and seagrass. Figure 17 represents a few raw samples corresponding to each target species.

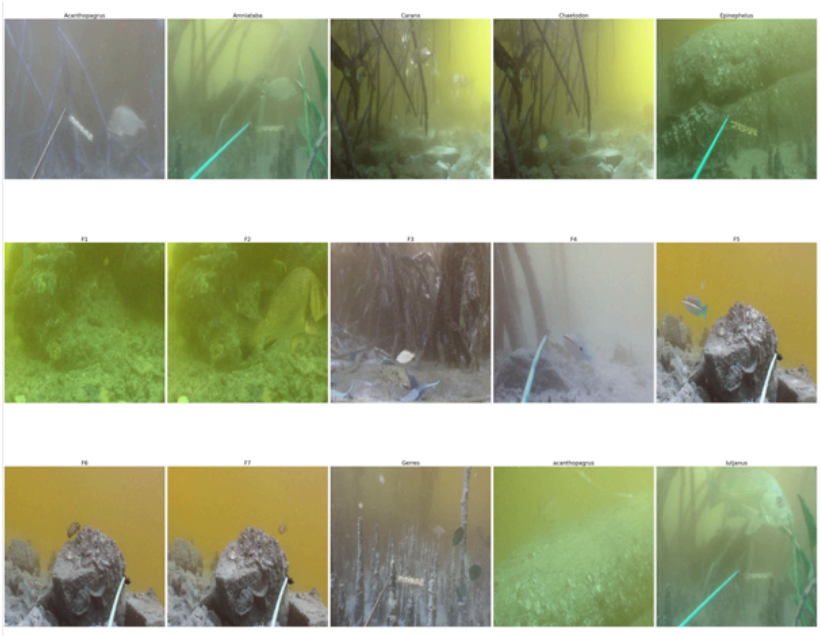
Figure 17

Sample Images from Fish4Knowledge Dataset

DeepFish. For data collection at the Australian site, the data was collected in collaboration with James Cook University. The project was named "DeepFish" and planned to span one year, from 2012 to 2013. A sample image each from one of the habitats is mentioned in Figure 18.

Figure 18

Sample Images from DeepFish Dataset



Data Pre-processing

Data preprocessing plays a vital role in the data analysis process, as it converts the data into a clean and organized format which eliminates any inconsistencies or anomalies in the dataset. This process helps to increase the efficiency of the classification models. It involves processes such as Handling missing data, Noise reduction in the images and handling inconsistent data.

Missing Data

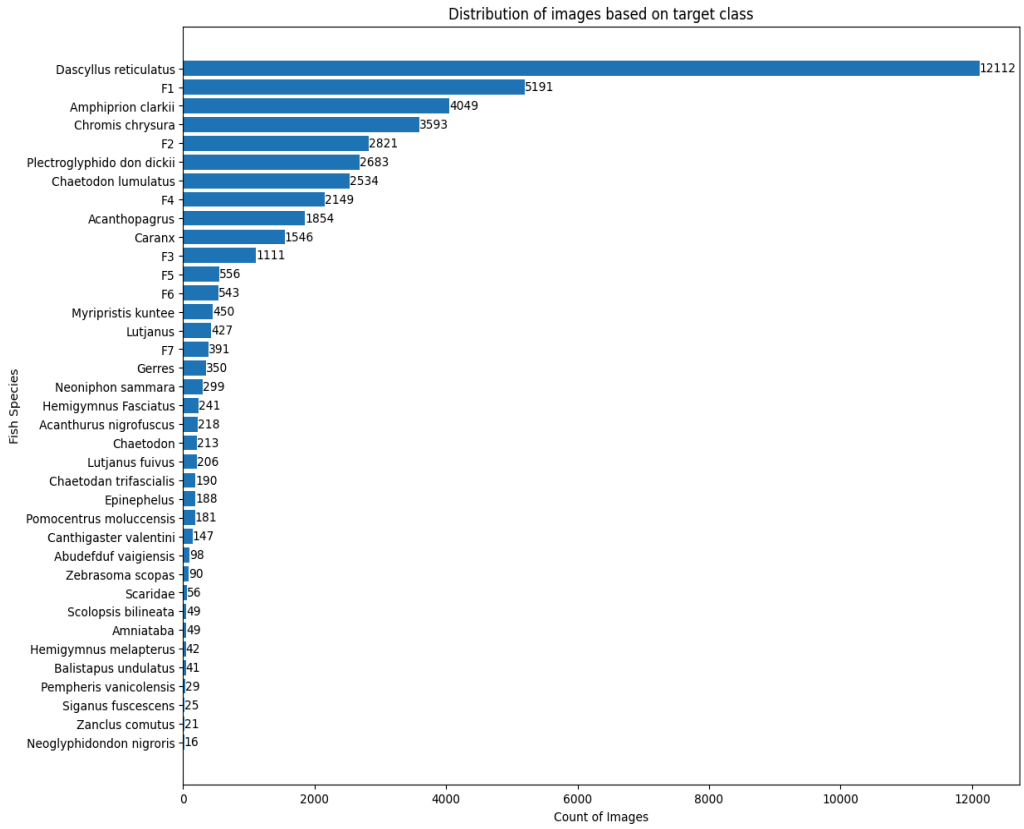
In machine learning, handling missing values is of paramount importance. It is a process that involves managing and treating incomplete or missing entries. These missing values can have significant importance on the accuracy of machine learning models. Bardoliwala et al. (2023) discussed the impact of dealing with missing values before modeling. After applying the

KNN imputer method on a few missing values and dropping a few null values, there was a slight increase in accuracy. This underscores the importance of handling missing values within the dataset.

Initially, the DeepFish dataset has a large number of images with no fish present and these are segregated into separate folders named ‘empty’. These images are excluded from training along with images that are not labeled and hence can not be used for training purposes. So, these too are ignored in the training pipeline. The number of images belonging to each species varies abnormally as shown in figure 19 inferring an imbalance in the dataset.

Figure 19

Distribution of images based on species



Dablain et al. (2023) found that augmentation leads to changes in weights, support vectors when applied to CNN models . By doing so, it results in a varied range of feature amplitudes which help in better association of features with their labels. The authors have concluded that this process enhances the model's generalization and robustness.

The figure 20 shows the image path and names of the images. The first part of the image name denotes the code where the image has been captured and the second part denotes the fish name. This naming convention helps in moving files during the processing phase and modeling phase.

Figure 20

Image naming convention

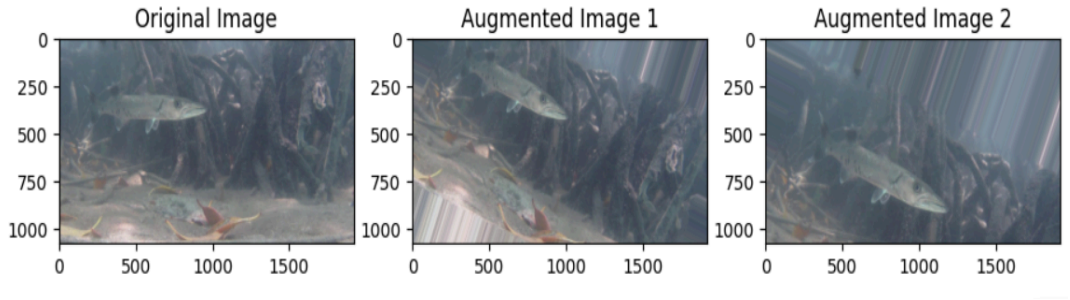
```
Subfolder 'Epinephelus':Path - D:/SJSU/Sem 2/DATA 270/project/processed_data\Epinephelus\9908_Epinephelus_f000000.jpg
Subfolder 'F1':Path - D:/SJSU/Sem 2/DATA 270/project/processed_data\F1\7268_F1_f000031.jpg
Subfolder 'F2':Path - D:/SJSU/Sem 2/DATA 270/project/processed_data\F2\7398_F2_f000006.jpg
Subfolder 'F3':Path - D:/SJSU/Sem 2/DATA 270/project/processed_data\F3\7398_F3_f000007.jpg
Subfolder 'F4':Path - D:/SJSU/Sem 2/DATA 270/project/processed_data\F4\7398_F4_f000007.jpg
```

Note. Depicts the naming convention used

To handle this class imbalance, image augmentation is used as an Oversampling technique which helps in sampling images by zooming in and out, cropping, rotating upto defined extent as shown in figure 21. Apart from this, Random Undersampling is applied to remove the excess images from a few classes. The threshold for sampling is concluded using the median of cardinality of images for each species. This well balanced dataset helps deep learning models in generalizing well while training.

Figure 21

Sample Augmented images for oversampling



Noisy Data

After handling missing data, noise removal techniques are employed to eliminate the noise present in the images, particularly in the Fish4Knowledge dataset as they are more prone to noise due to image quality after initial inspection. The noise must be removed or reduced to some extent since "the presence of noise in a data set can increase the model complexity and time of learning which degrades the performance of learning algorithms. Therefore, there is a need to identify and handle these noise in data sets" (Gupta et al., 2019, p. 466).

There are various methods such as the Gaussian filter, Median filter, Denoise autoencoder etc. Kumar et al. (2020) explained that the Gaussian filter based on the standard normal function is a linear type of filter while the median filter is a non-linear filter. Because of this difference, the median filter preserves edges while removing noise providing a significant advantage over Gaussian filters. Median filter is used to remove salt and pepper noise. After comparing all three filters, it is concluded that the median filter takes fewer iterations to provide the best results in minimum time.

Therefore, a median filter is employed to remove noise from the images. The most important feature to select while using a median filter is kernel size. While large kernel size reduces the noise, it also leads to loss of details. After experimenting with different kernel sizes, 3x3 size is chosen as a trade-off between noise reduction as well as retaining the details in the image. The image shown below in figure 22 is filtered over a 3x3 kernel. Median filtering produced a lower mean square error over Gaussian filter for most of the data. Figure 23 shows the variation in the distribution of pixels after median filtering has been applied. It also represents the mitigation of outliers in intensity values and more refined distribution. The blue box represents the original image and the red box represents the filtered image. The pixel difference can be easily spotted as the outliers have been removed with the help of a median filter thereby giving a better distribution of pixels.

Figure 22

Sample image after applying median filtering for noise removal

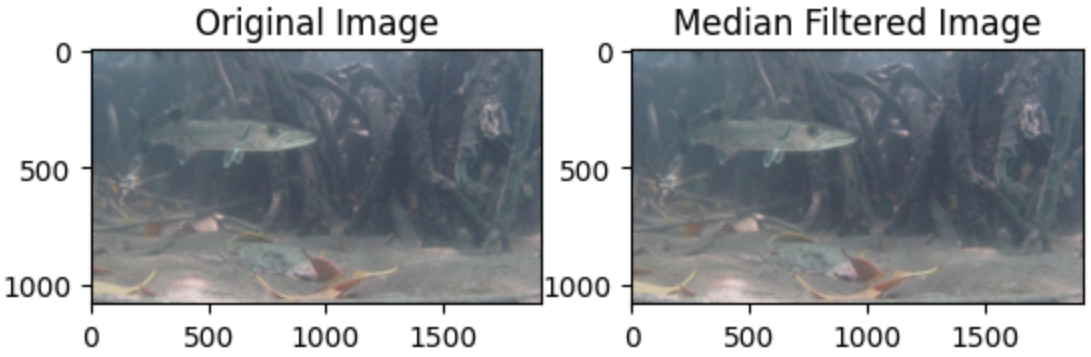
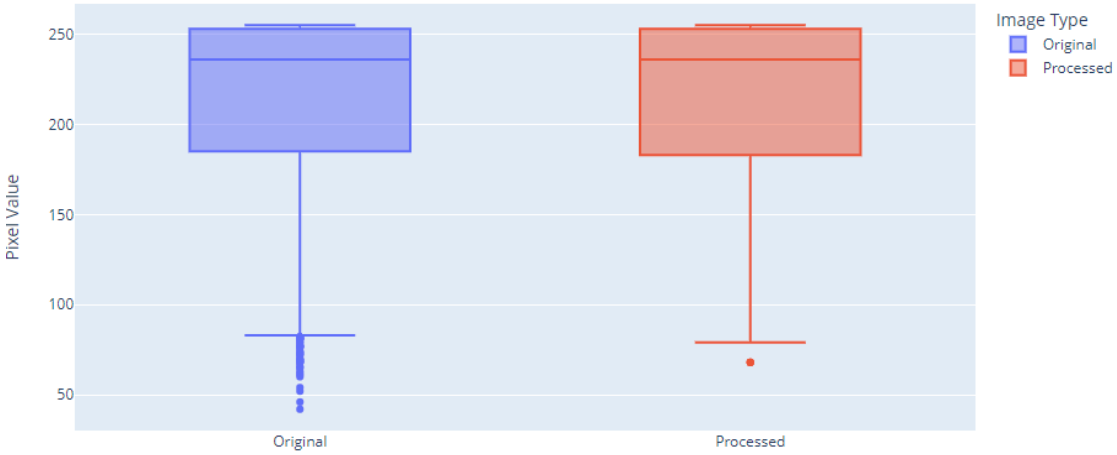


Figure 23

Comparison of pixels after applying median filter



Inconsistent Data

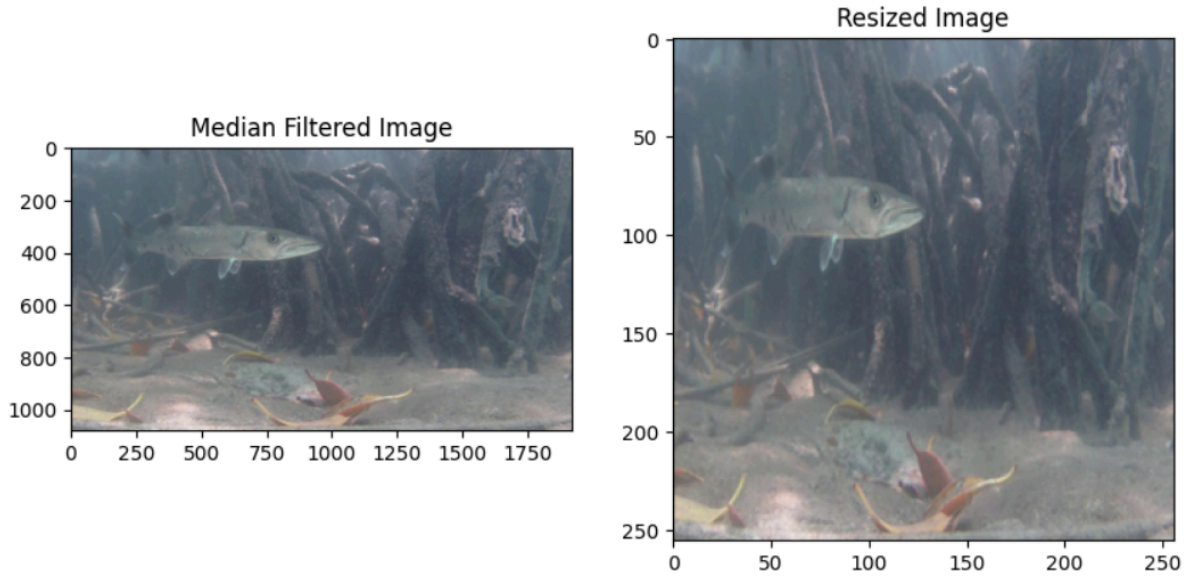
Resizing of images is the next important step in data preprocessing. Since the data is collected from different sources, it is trivial to have images with varying sizes. Images from DeepFish are HD resolution whereas that of Fish4Knowledge are poor quality ones. Input image shapes must be consistent for a machine learning model. One of the effective approaches to resizing is the image downscaling method. Talebi et al. (2021) state the main reasons for resizing, include memory limitations, which prohibit CNN's model from training at high resolutions, and the training model would be time-consuming if there are large images in the dataset. This emphasizes the importance of image resizing in the dataset.

For resizing the images to a uniform size, the image distribution in the dataset is analyzed. From this, it is found that most of the images are between zero and 2000 pixels. Since we aim to reduce size along with image uniformity we choose 256 x 256 pixels which is also the standard size used in most of the pre-trained models. Therefore all the images are resized to the

size of 256 pixels by width and height so that they are processed uniformly by the model. This helps to mitigate issues that arise from varying image resolutions which can lead to better training and generalization. Figure 24 shows the resized version of the image which had dimensions 256x256.

Figure 24

Sample image before and after resizing



Data Transformation

Data transformation is a process converting the data to a more suitable format to enhance the performance of the machine learning and deep learning models. For image dataset, this process involves Data Smoothing, Data Normalization, and Image enhancement.

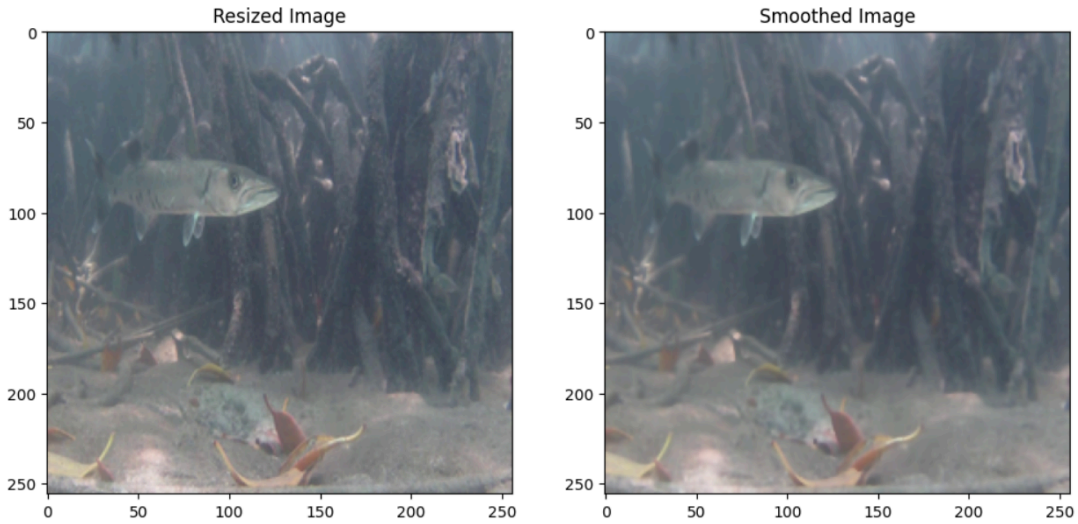
Data Smoothing

Data Smoothing can help in the detection of false edges by applying filters to the image. Tomasi et al. (1998) discovered a new approach to preserve edge information within images by applying a smoothing technique called Bilateral Filtering. It is similar to traditional domain

filtering and the output depends on parameter values. This is used for smoothing the data where it smoothes an image by replacing each pixel of the image with the weighted average of all of its neighbors. It preserves the edges while removing the noise from the image which is a crucial step for accurate interpretation. Figure 25 represents the effect of the smoothing process visually.

Figure 25

Sample resized image after smoothing transformation

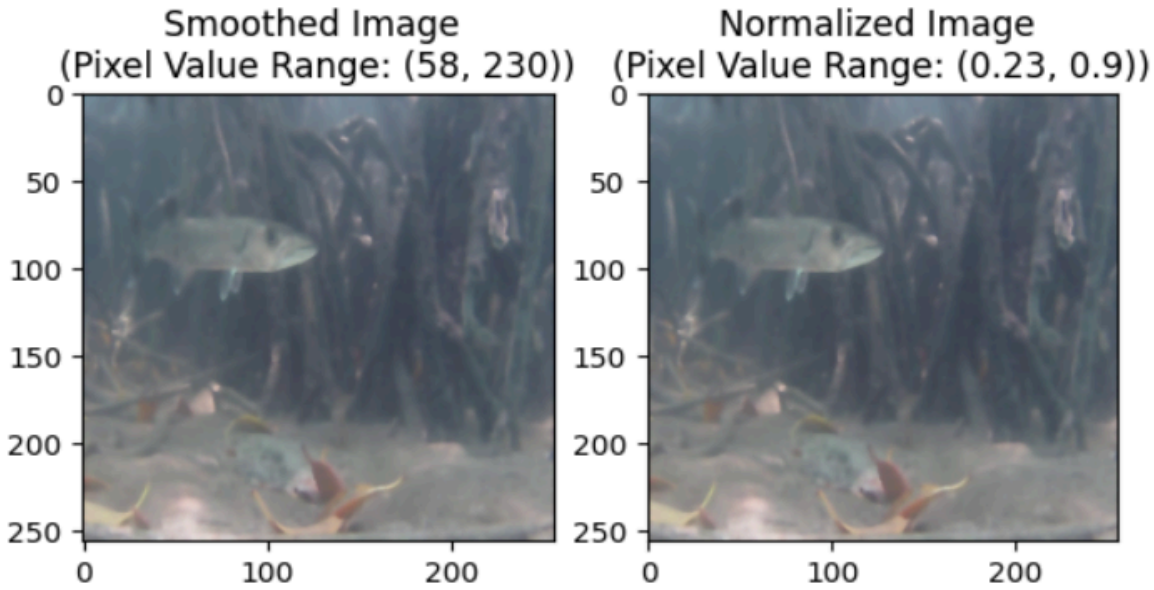


Data Normalization

In image processing, normalization is a process of transforming the pixel values range to a standard scale. The main purpose of normalization is to achieve uniformity without disturbing the differences in the range of values and without losing the information. It helps identify the underlying patterns in the dataset. It ensures that no single feature influences the model's learning. The pixel values of all the images in the fish dataset are ranging from 0 to 255 values. In the normalization process all the pixel values are divided by 255, to transform all the values between 0 to 1. By converting all the values, we ensure that the data is consistent to train the models. Figure 26 represents the original image and normalized version of the image.

Figure 26

Sample image before and after normalization along with pixel values range



Data Enhancement

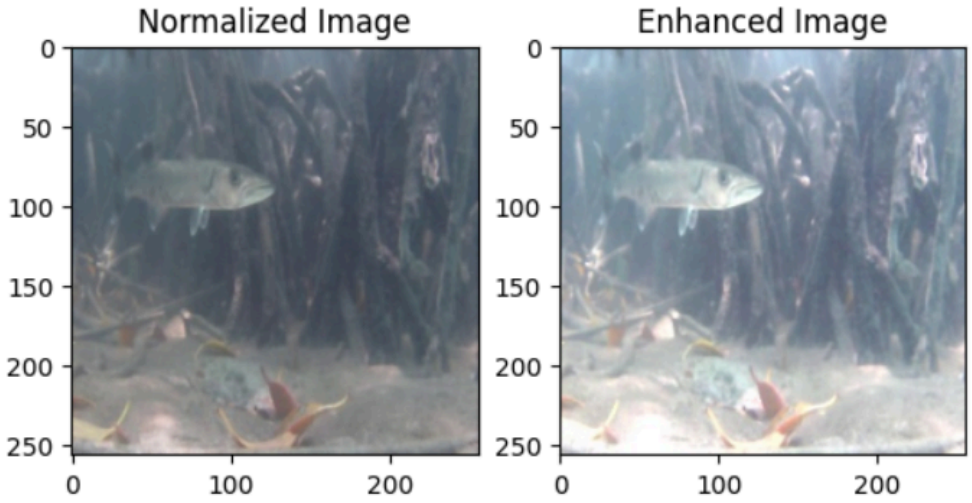
The next step in data processing is enhancing the quality of the images. It is a crucial step especially when dealing with underwater images which are typically captured in low light conditions. The quality of images directly impacts research in areas such as marine life exploration. A study by Raveendran et al. (2021) demonstrated that underwater images often have greenish tint, which leads to decrease in the image visibility. Enhanced images had more image quality which was found from their histograms of distribution of red, blue and green colors.

Image enhancement involves adjusting the brightness and contrast in the images to compensate for the low lighting conditions and distinguishing foreground and background which alter the interpretability of the images. For machine learning and computer vision

applications, image enhancement as a preprocessing step normalizes the dataset, leading to more effective object detection, classification, and segmentation. It also helps in feature extraction and pattern recognition when working with machine learning models. Over-enhancing an image can lead to loss of details because of excessive brightness in the image. Therefore, a trade off value of 1.5 is chosen as a brightness factor for the fish dataset that enhances brightness as well as preserves image's information. Figure 27 shows the original image and the enhanced image.

Figure 27

Comparison of sample image against enhanced image (adjusted brightness & contrast)



Data Reduction with SVD

Data Reduction can be helpful for efficient storage purposes. Tian et al. (2005) describe this technique as basically selecting very few singular values to reconstruct the original matrix. SVD is a data compression technique that represents the image with a smaller set of values, reducing the storage space. This technique preserves crucial features while compressing the image, which is necessary. Figure 28 represents the original image and the image reconstructed

using SVD. The number of components used for reconstruction is 10, which gives an optimal balance for image quality and data reduction.

Figure 28

Sample image reconstructed with top 10 components using SVD technique

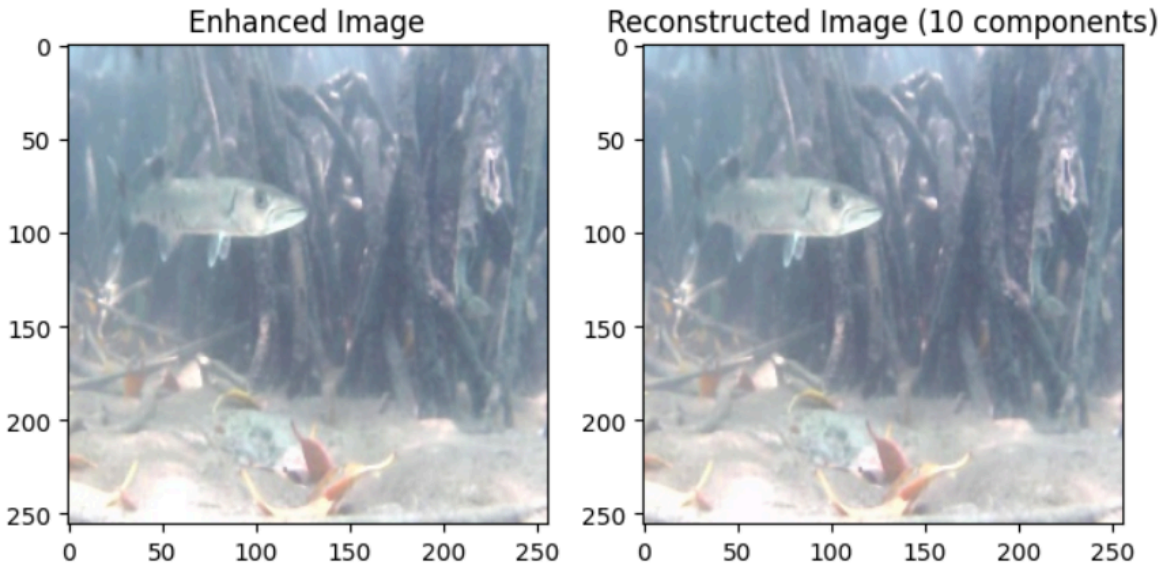


Figure 29 shows the drop-off point i.e the point at which the values carry less energy for red, green, and blue upon performing SVD (scree plot). The point in the plot from where values start to level off is very crucial as it decides the number of components to be considered for modeling. Repetitive experiments with different components have been conducted on a base model to find the optimal components for dimensionality reduction. The experiments resulted that beyond 10 components there has been any further increase in the accuracy of the base model. Figure 30 shows end results of each stage i.e., processing and transformation phase.

Figure 29

Scree plot of singular values

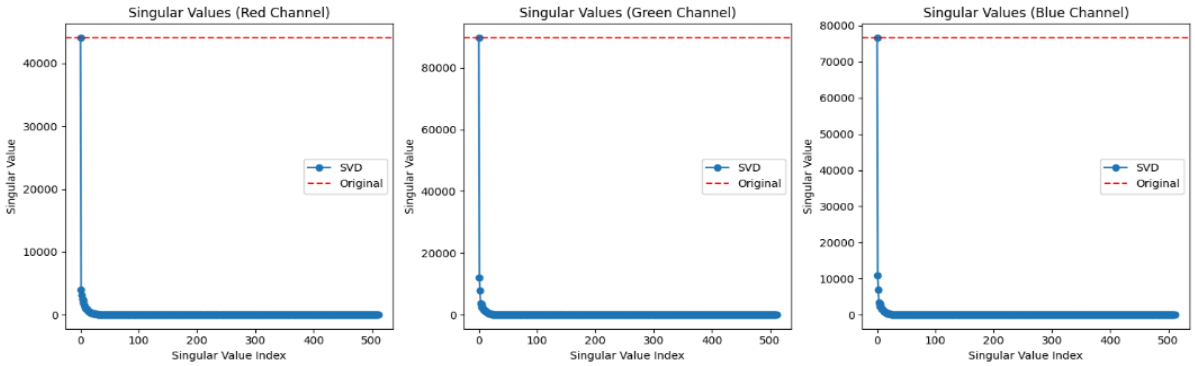
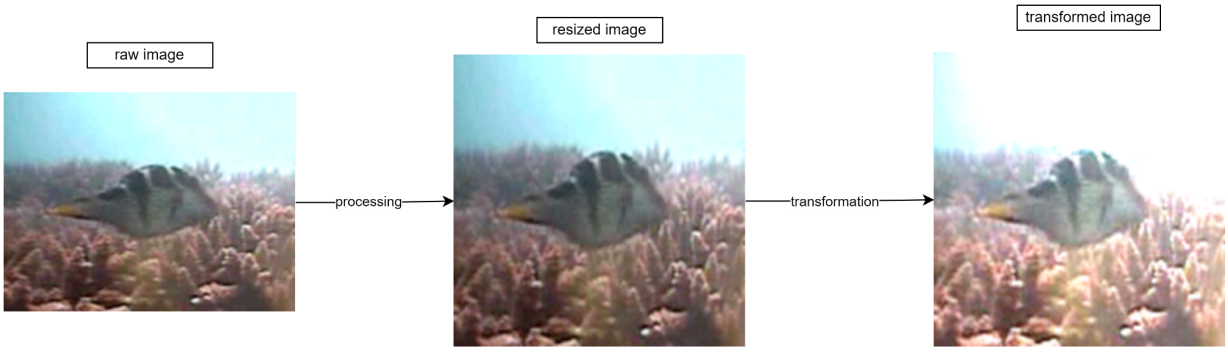


Figure 30

Representation of Data Processing phase



Data Preparation

Once all these preprocessing steps are performed, the dataset is split into unique subsets for training, validation and testing. Nurhopipah and Hasanah (2020) mentions that the performance of the machine learning model depends on the data split strategy. Although they have discussed various techniques like k-Fold Cross Validation, Random Sub-sampling Validation and few other techniques, the common approach followed is to create three subsets of data: training data for training on model, validation split for evaluating the trained model and

generate feedback to the model to adjust model's learning and finally test data to determine the model's accuracy and performance. Training subset captures most of the entire data and in this case, using the hold out approach, 70% of the raw data is attributed towards training. The rest of the data is equally distributed for validation and testing. This makes the overall ratio of train, validation and test datasets in the ratio 70:15:15. Since each class has 241 images, there will be an equal number of images for all classes in each of the subset as shown in table 12.

Table 12

Number of Images within Training, Validation and Testing data subsets for each class

Training	Validation	Testing
168	36	37

Data Statistics

The project was developed using two datasets of images totaling over 6.5 GB of data with over 44,000 valid images. Data exploration and EDA was performed on the raw dataset and key insights were generated. Due to the complexity of dataset size and computation for modeling, the entire dataset is sampled based on median value and just 241 images per class were taken into consideration. Random Over Sampling has been applied to perform under sampling to reduce the images. Data Augmentation has been performed to increase the images in the dataset. Using augmentation multiple samples of the same image have been created with focus on different areas of image, tilt, flip etc. This totals to 8917 images for 37 classes. The image distribution for each class after sampling is shown in figure 31 along with distribution of images in each subfolder for train, test and validation in figure 32.

Figure 31

Distribution of images based on species after Sampling

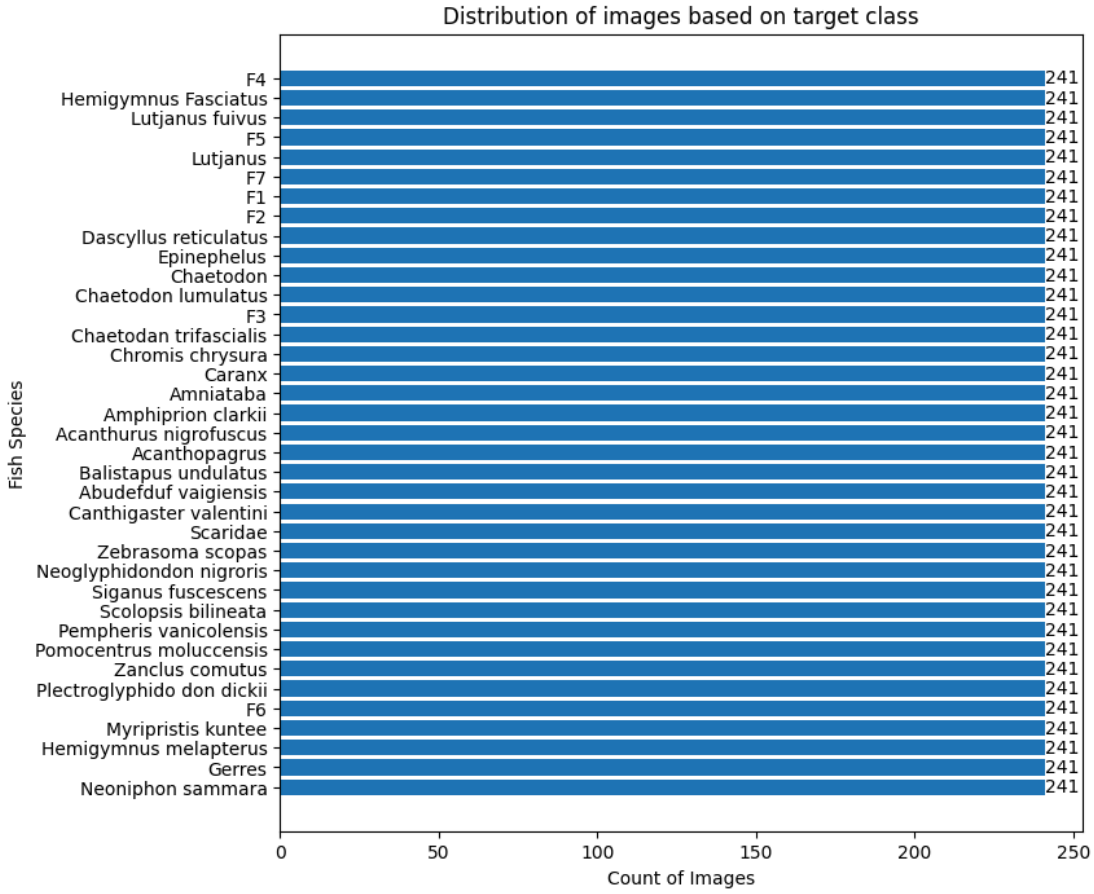


Figure 32

Files in each subfolder of train, test and validation

Folder path: D:/SJSU/Sem 2/DATA 270/project/modeling/train

Subfolder 'Abudefduf vaigiensis': 168 files
Subfolder 'Acanthopagrus': 168 files
Subfolder 'Acanthurus nigrofuscus': 168 files
Subfolder 'Amniataba': 168 files
Subfolder 'Amphiprion clarkii': 168 files
Subfolder 'Balistapus undulatus': 168 files
Subfolder 'Canthigaster valentini': 168 files
Subfolder 'Caranx': 168 files
Subfolder 'Chaetodan trifascialis': 168 files

Folder path: D:/SJSU/Sem 2/DATA 270/project/modeling/test

Subfolder 'Abudefduf vaigiensis': 37 files
Subfolder 'Acanthopagrus': 37 files
Subfolder 'Acanthurus nigrofuscus': 37 files
Subfolder 'Amniataba': 37 files
Subfolder 'Amphiprion clarkii': 37 files
Subfolder 'Balistapus undulatus': 37 files
Subfolder 'Canthigaster valentini': 37 files
Subfolder 'Caranx': 37 files
Subfolder 'Chaetodan trifascialis': 37 files

Folder path: D:/SJSU/Sem 2/DATA 270/project/modeling/val

Subfolder 'Abudefduf vaigiensis': 36 files
Subfolder 'Acanthopagrus': 36 files
Subfolder 'Acanthurus nigrofuscus': 36 files
Subfolder 'Amniataba': 36 files
Subfolder 'Amphiprion clarkii': 36 files
Subfolder 'Balistapus undulatus': 36 files
Subfolder 'Canthigaster valentini': 36 files
Subfolder 'Caranx': 36 files
Subfolder 'Chaetodan trifascialis': 36 files

Applying noise reduction resulted in denoising of images and more uniformed distribution. The distribution of image sizes across height and width dimensions is shown in

figure 33. The plot shows that width and height are varying for different images and is inconsistent. A histogram representing distribution of pixel values across all three color channels is presented in figure 34. It also represents that the graph is normally distributed

Figure 33

Distribution of height and width of raw images from Fish4Knowledge dataset

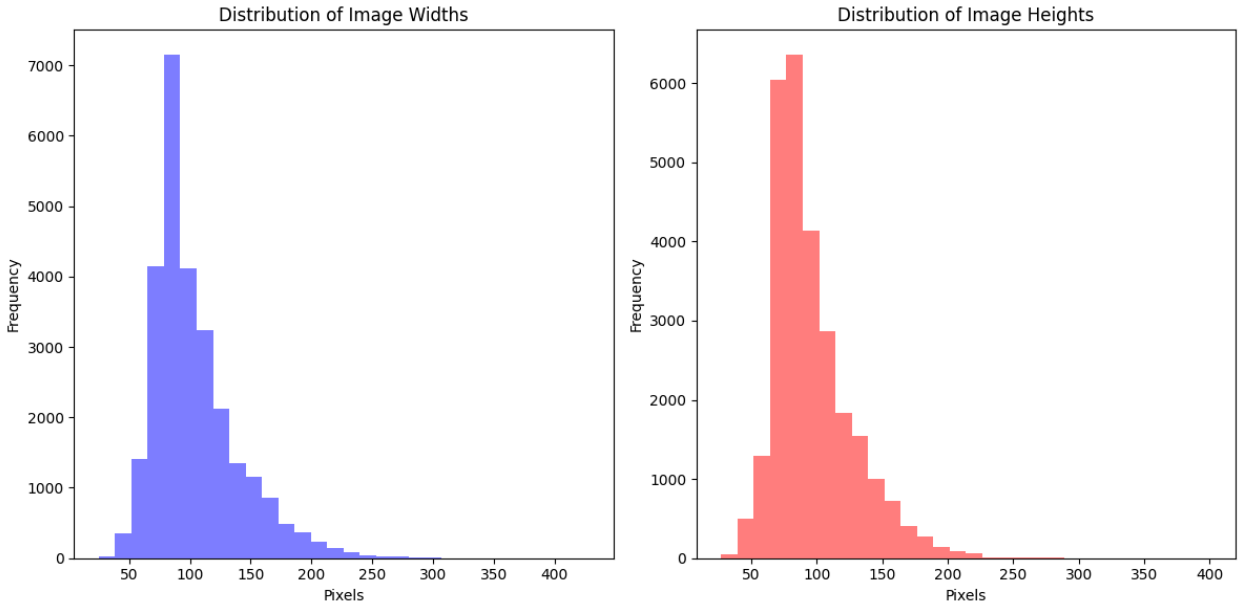
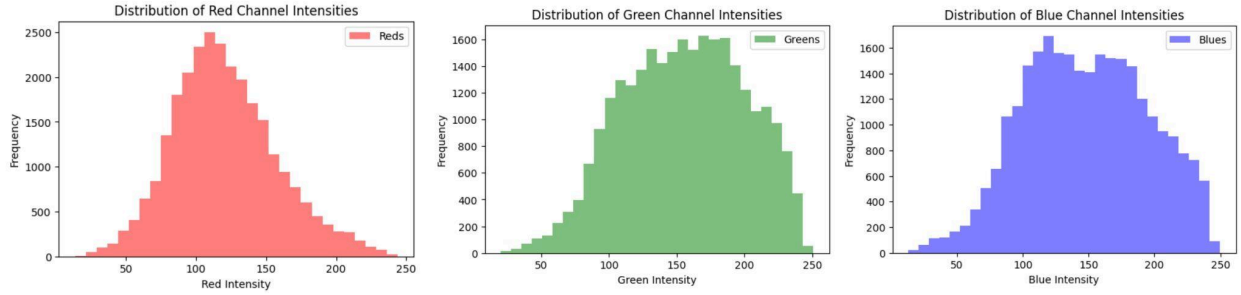


Figure 34

Distribution of pixel values in the red, green and blue channels of images from Fish4Knowledge



Now, all the images are resized to 256 x 256 shape uniformly for further transformations and modeling and bilateral filtering is applied to smooth the images for preserving information at

the edges. Pixel values are normalized to contain values within zero and one. This makes the deep learning models train faster and get more accurate results. This image is enhanced by adjusting brightness and contrast to combat green or bluish concentration that is common in underwater images. Later, these images are reconstructed using SVD technique to reduce and compress the data. Figure 35 shows the raw image at the beginning and end of this entire transformation pipeline.

Figure 35

Sample raw input image and transformed image

Data Analytics Results

Data analysis is the process of finding hidden patterns and insights from the data. All the necessary processing has been performed on the data as discussed in the previous sections. The processed data has more valuable information and is more easily understandable and computable by the computer. Figure 36 and 37 show clear visual enhancements upon transformation of images. Moreover, finer details and low light images are more clearly visible.

Figure 36

Sample images of Fish4Knowledge dataset after transformation steps

Figure 37

Sample images of DeepFish dataset after transformation steps

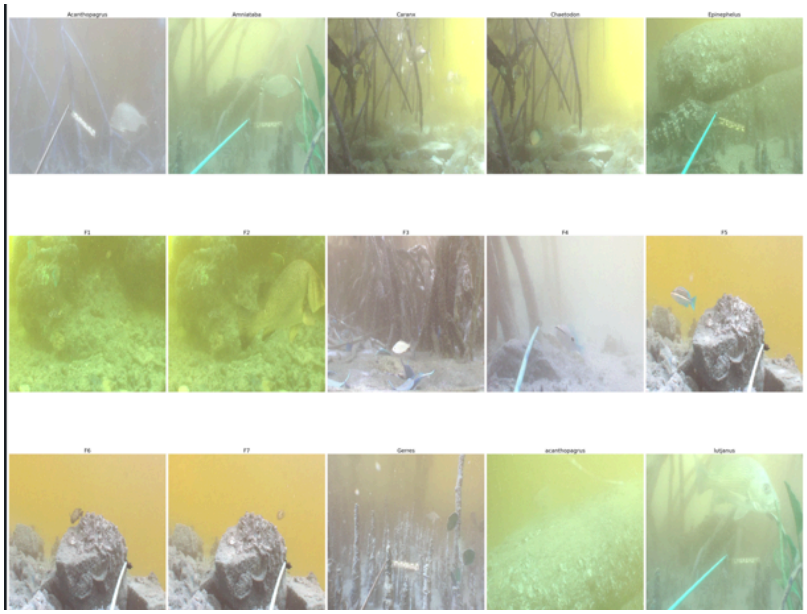


Figure 38 shows the distribution change in pixels after enhancing the images. The enhanced image graph is more shifted to the right indicating the increase in the brightness. Peak variations represent increased dynamic range and balanced contrast distributions relative to the raw data. Initially there was a uniform distribution across images upon applying enhancement the values changed to normal distribution clearly indicating the value changes.

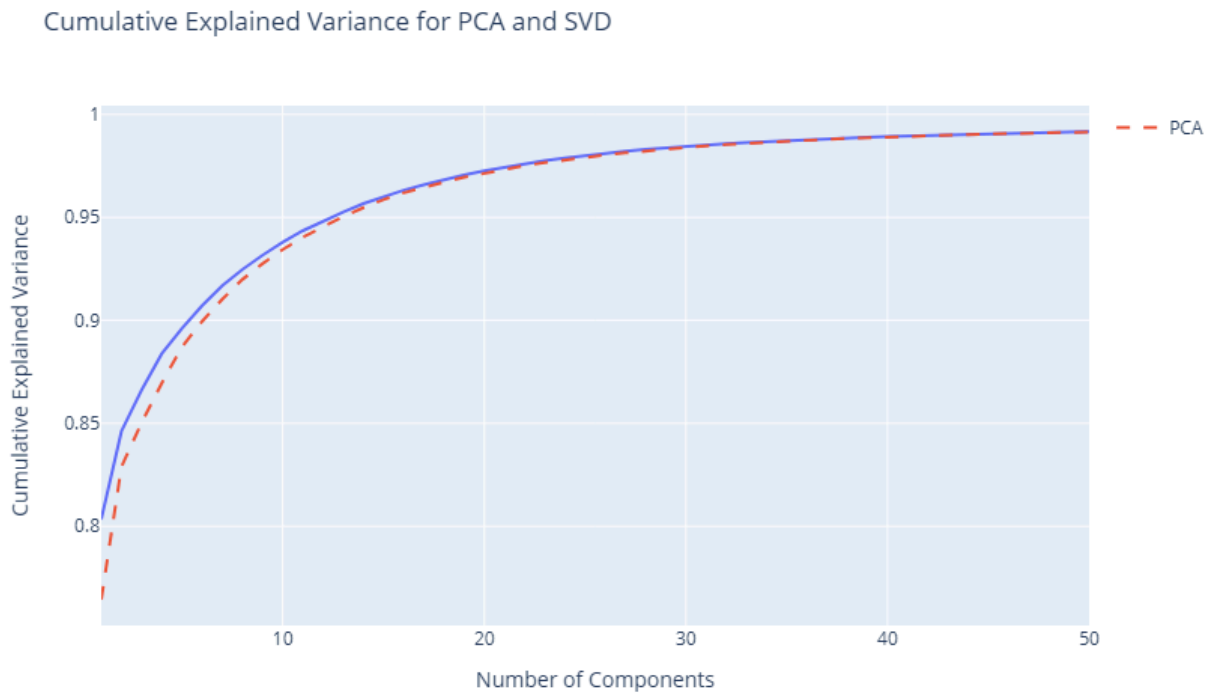
Figure 38

Distribution shift after Enhancement

Figure 39 shows the variance explained by SVD and PCA. PCA initially explained 76% while SVD explained 80% of the variance. SVD was able to explain more for minimal no. of components and the graph has a higher curve when compared with PCA. This gives us information on no, of components to be retained to get high information. Retaining only the dominant components allows machine learning algorithms to operate in reduced yet highly informative feature spaces. This reduces the data down from potentially thousands of pixel features to fewer hundred transformed features that holistically profile the image contents.

Figure 39

Variance explained by PCA and SVD



With the application of these data engineering techniques, the dataset is now ready for the core component of the project which is modeling. This data is used to train and build machine learning models. All the code for modeling stage is shown in Appendix A.

Model Development

Model Proposals

Once the training data is ready, based on the literature review of similar studies, four models: CNN, MobileNet, Inception-ResNet and Vision Transformer were selected. The following section will discuss the model architecture, data flow and the implementation process of these models.

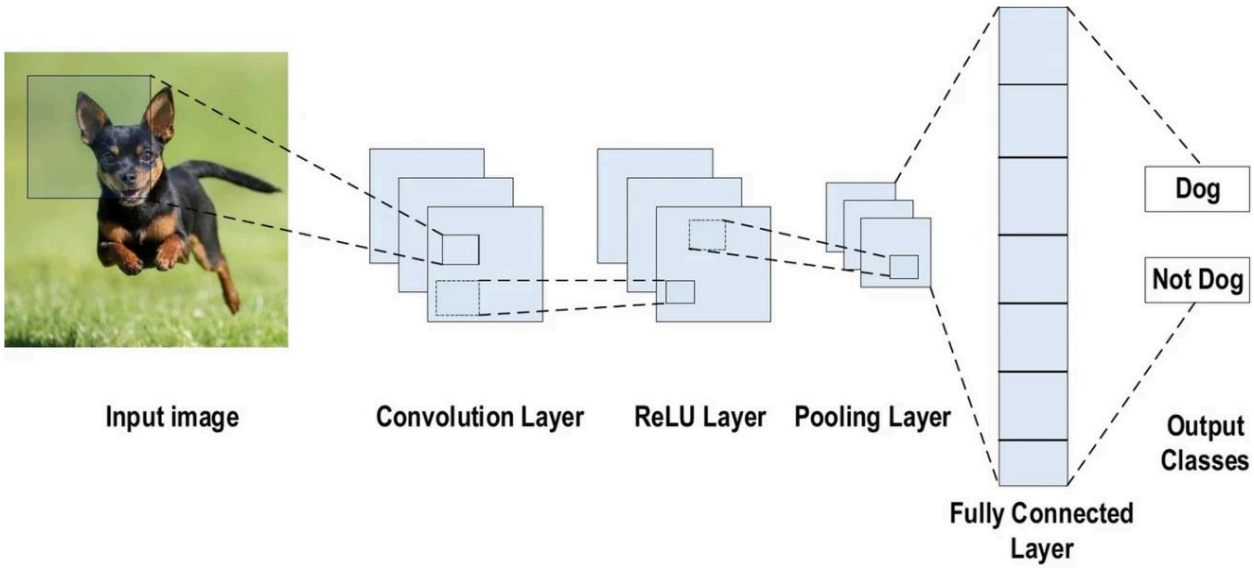
Convolutional Neural Network

The first proposed model is CNN which is a classic and widely used algorithm for image classification. According to Alzubaidi et al. (2021), CNN is inspired by the human visual nervous system, where neurons in the retina extract low-level features while higher-level visual areas refine these features to recognize objects and scenes. This hierarchical organization is mirrored in CNN, where convolutional layers progressively were extracted to refine image features, enabling the network to learn complex patterns and make accurate classifications.

Key features of CNN include the ability to reduce the dimensionality of extensive image data effectively into small image data without compromising the preservation of essential image features. This is in line with the principle of image processing in which data reduction and feature preservation are crucial for efficient and accurate analysis. The figure 40 taken from Alzubaidi et al. (2021) shows a sample architecture of CNN.

Figure 40

Typical CNN Architecture for Image Classification



A typical CNN consists of three parts: convolution layer, pooling layer, and fully connected layer. The convolutional layer is used for extracting local features in the image. The pooling layer is used for dimension reduction, and the fully connected layer is similar to the traditional neural network portion and is used to generate the desired result. As Alzubaidi et al. (2021) and Rathi et al. (2017) demonstrated, activation functions such as ReLU can be incorporated within these layers to introduce non-linearity, enhancing the model's learning capacity.

CNN Algorithms. The proposed CNN algorithm consists of 15 convolutional layers, pooling layers and fully connected layers with their function. The algorithm design is based on the reference of Rathi et al. (2017) and Rauf et al. (2019). Albawi et al. (2017) provide a detailed breakdown of each layer, as summarized below.

Convolutional Layer. This layer is the core component of a CNN. It facilitates the feature extraction from the input image by convolving the input data with a set of filters. Each filter is a small matrix of numbers that is applied to a small region of the image. The output of the convolutional layer is a feature map, which is calculated by performing a dot product between the filter and the corresponding pixels in the image region. The convolutional layer typically has the following parameters:

Number of Filters: The number of filters determines the number of feature maps that are produced.

Filter Size: The filter size determines the size of the receptive field of the filter. The receptive field is the area of the input data that the filter is applied to.

Activation Function: As Alzubaidi et al. (2021) explained, ReLU is the most widely used activation function in CNN. It converts the entire input to positive values. The main advantage of ReLU over other activation functions is its lower computational load. The function can be mathematically expressed as in (1).

$$f(x)ReLU = max(0, x) \quad (1)$$

Pooling Layer. This layer is for reducing the dimensionality of the feature maps output from the convolutional layer. This is done by applying a pooling operation to each feature map. The pooling operation reduces the size of the feature map by selecting a subset of the values in the feature map. Alzubaidi et al. (2021) presented the various pooling methods available for pooling layers, including tree pooling, gated pooling, min pooling, max pooling, global average pooling (GAP), and global max pooling. Max pooling is one of the most frequently used pooling methods and is selected to implement this algorithm. It selects the maximum value from a subset of the values within the feature map. The pooling layer typically has the following parameters:

Pool Size: The pool size determines the size of the subset of values used in the pooling operation.

Stride: The Stride determines the amount by which the pooling operation is shifted across the feature map.

Fully Connected Layer. This layer is for classifying the input image into one of the possible fish species. It is a standard neural network layer that is directly connected to every node in both the previous and the next layer. The fully connected layer has the following parameters:

Number of Neurons: The number of neurons determines the number of classes that the network can classify the input data into.

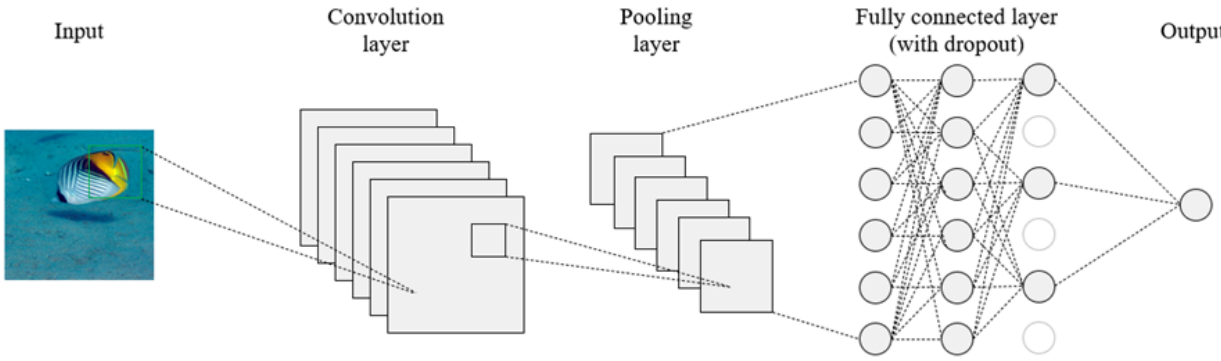
Activation Function: The activation function is applied to the output of the fully connected layer. This function introduces non-linearity into the network, which allows it to learn complex patterns.

Loss Function: As documented by Alzubaidi et al. (2021), when the final output layer generates the final classification result. To assess the predicted error generated during training, various loss functions can be used in the output layer. Common loss functions include SoftMax loss function, Euclidean loss function and hinge loss function. SoftMax loss is chosen because it uses the output layer to transform raw scores into a probability distribution, making it a suitable choice for multiclass classification tasks.

Dropout. Based on Rauf et al. (2019) approach, dropout is applied to the layers to randomly drop out a certain percentage of neurons to further enhance generalization during training, preventing overfitting. Alzubaidi et al. (2021) considered dropout as a common technique for generalization. During each training epoch, neurons are randomly dropped. This distribution of feature selection power across the entire group of neurons forces the model to learn different independent features. During the training process, dropped neurons are excluded from both backpropagation and forward propagation. In contrast, the full-scale network is used for prediction during the testing process. Figure 41 represents the proposed algorithm structure.

Figure 41

Demonstration of the Proposed Algorithm



MobileNet

A second model chosen is MobileNet, a deep learning architecture proposed by Howard et al. (2017) at Google for performing vision-based tasks efficiently in mobile and embedded devices. Originally proposed as a solution to the ILSVRC 2012 challenge (Russakovsky et al., 2015) but has been widely adopted using Transfer learning (Zhuang et al., 2020) which is becoming a popular approach to build deep learning models. This architecture is constructed using depth-wise separable convolutions instead of standard convolution layers. It has 28 layers and can be further shrunk or expanded using width and resolution multipliers based on the use case. There are three versions of MobileNet and each one improves over previous versions in terms of network structure, efficiency, and accuracy.

Sandler et al. (2018) further enhanced the MobileNet by including an inverted residual structure with bottleneck layers. There are three major steps in the algorithm. The traditional convolution operation as shown in equation (2) is replaced by two steps of depthwise and pointwise separable convolutions. This reduces the computational cost of operation from (3) to

(4) by a factor of k^2 where h, w, and d are dimensions of the input image and k is the kernel size.

$$h_i \mathbf{x} w_i \mathbf{x} d_i \ * \ R^{k \mathbf{x} k \mathbf{x} d_i \mathbf{x} d_j} = h_i \mathbf{x} w_i \mathbf{x} d_j \quad (2)$$

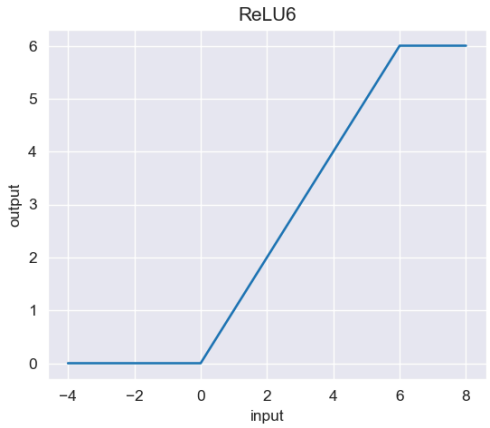
$$h_i . w_i . d_i . d_j . k . k \quad (3)$$

$$h_i . w_i . d_i . (k^2 + d_j) \quad (4)$$

The next stage is the bottleneck layer containing inverted residuals using these two-step convolution blocks. Within the residual network, the ReLU6 activation function is used between the layers to reduce the computational expense by clipping activation function outputs to a maximum of 6 improving the low-precision computation. The graphical output of ReLU6 is shown in Figure 42 and the mathematical representation is shown in equation (5). After a series of bottleneck layers, the final stage has global pooling and fully connected layers and the output layer has softmax activation function as it produces probability distribution for each class type in multi-class classification problems (Nwankpa et al., 2018). It results in a series of values lying between zero and one for each class as per (6) and the predicted class has the highest value.

Figure 42

ReLU6 Function Graph



$$ReLU6(x) = min(max(0, x), 6) \quad (5)$$

$$Softmax(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (6)$$

Pseudo Code. This section contains details about how each neural network block is implemented in the model. Table 13 shows the components involved in each layer of the MobileNet V2 architecture. The convolution block performs depth-wise and point-wise convolutions and the bottleneck layer contains a 1x1 expansion layer followed by this convolution block. Finally, it checks for input and output dimensions to skip the connection if they are identical. The model contains seven such bottleneck layers surrounded by a convolution layer at both ends and a fully connected layer at the output layer to predict the probabilities of each class.

Table 13

Pseudo Code of MobileNet V2 network

Algorithm Pseudocode description of MobileNet V2

Require: set of images passed as input to MobileNet V2 function

Output: set of floats representing the probability of input image belonging to that class

```
1. Convolution Block( ) {
        Depthwise Convolution
        Batch Normalization
        ReLU6 Activation
        Pointwise Convolution
        Batch Normalization
        ReLU6 Activation
   }
```

2. BottleNeck Layer( ) {
    Expansion Layer
    Batch Normalization
    ReLU6 Activation
    Convolution Block( )
    Skip connection in case of identical input-output dimensions
   }
3. MobileNet V2( ) {
    Convolution 2D
    13 x BottleNeck Layer( )
    Convolution 2D
    Average Pooling Layer
    Fully Connected Layer (Softmax)
   }

MobileNet architecture can be customized and fine-tuned based on the task at hand to adjust performance, inference time, computational requirements etc. It has two main parameters: Width multiplier (called alpha) and Resolution multiplier. Width multiplier helps to scale the width of the network which usually lies between 0.35 to 1.4 and impacts model parameters and total operations. Resolution multiplier helps to use this model for images of almost any size. Both these parameters must be tuned carefully to strike the balance between performance and accuracy. Apart from these, a few other variables like the choice of activation function, and dropout rate can have little impact on model training and results.

Inception ResNet

The third model chosen for this study is Inception ResNet. It is one of the significant achievements in the realm of deep learning technology especially in computer vision. This neural network synergizes the strengths of two powerful architectures: Inception network and Residual network. The Inception network effectively processes images at various scales while the residual

networks address the problem of vanishing gradient descent facilitating the training of deeper networks. These strengths of the inception model and ResNet model made a profound impact in the field of deep learning. These advancements helped in the development of the Inception ResNet, integrating inception modules with residual connections. Combining these two architectures is one major advancement blending the efficiency of the inception modules with learning capabilities of ResNet architecture. Consequently, Inception ResNet emerged as a popular choice in image recognition and classification tasks. It is particularly useful in Image classification from simple object recognition to complex images. It is also used to determine the location within the images which is crucial in applications such as autonomous vehicles. In the medical field, this model is used in analyzing X - rays, MRI scans and CT scans. The main layers in the inception ResNet v2 are inception modules, residual connections which define the functionality of the model as mentioned below.

Inception modules. The evolution of integration of ResNet started by the introduction of original inception models known as GoogleNet introduced by Szegedy, a model that automatically adjusts the combination of filters in each layer . The inception module, an integral part of the Inception (Google Net) architecture, a network within a network, was designed to optimize the processing of images by introducing several parallel paths within each module (Szegedy et al., 2015). In each path, they have incorporated different types of filters such as 1x1, 3x3, 5x5 convolution filters and pooling operations such as 3x3 max pooling. This helped to simultaneously filter the same image in multiple ways which captures even the minute details of the image. One of the key features of the inception module is the usage of 1x1 convolutions for dimensionality reduction. This method significantly reduced the computational burden which

enabled training Deep networks without increase in computational costs. The convolution filters used increase as the feature maps increase and mathematical formula for convolution filters as given in Equation (7).

$$G_{(k-i)/s+1,\,(l-j)/s+1,\,n} = \sum_{i,\,j} Ki,\ j,\ m\ *\ C_{k,\,l,\,m} \tag{7}$$

As given in Equation (7) and Equation (8), K representing the convolution kernel, i, j are the sizes of the kernel and m is the number of channels. G represents the output characteristics. C is the input characteristic graph while k, l are the sizes of the characteristic graphs (Peng et al., 2022).

Normalization layer is also added to the inception network. This is because the network keeps on updating a large number of parameters for every iteration. Therefore to ensure stability and large convergence, a normalization layer is added between the convolution layers in the network. (Yao & Huang, 2023). This functionality can be represented with a series of Equation (8), Equation (9). In the Equation (10) , x_i is the input data, m gives the batch size in the input data. μ_B and σ_B represent dataset mean and variance. $\hat{x}_i$ is the normalized data, y_i is the output of the Batch Normalization layer as shown in Equation (11).

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{8}$$

$$\sigma^2_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{9}$$

$$x_i \leftarrow \frac{(\hat{x}_i - \mu_B)}{\sqrt{\sigma^2_B + \epsilon}} \tag{10}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \quad (11)$$

Residual Connections. These residual connections emerged from the ResNet model, an artificial Neural Network that allows the model to skip between the layers . It is highly used in various computer vision tasks. ResNet core is the usage of residual Connections which create a shortcut connection in the network which provides an alternate route for gradient flow during the back propagation. This method helps alleviate the diminishing gradient problem enabling the network to train with increased depth without adding any complexity.

Dense layer also known as a fully connected layer typically found towards the end of the Inception ResNet model is primarily used as a decision making layer to interpret the features which are extracted by the convolution layers. It picks up the largest element from the last feature map.

Inception ResNet is available in versions v1,v2,v3 and v4. Inception ResNet v2 is chosen as one of the models in the project. Inception ResNet v4 and Inception ResNet v2 demonstrate increased complexity compared to v1 and v3. Inception ResNet v4 introduces a more sophisticated structure compared to v2 focusing on optimizing performance. However, the inception of ResNet v2 comes as a tradeoff between computational efficiency and accuracy (Szegedy et al., 2017). Consequently, v2 is utilized to classify the fish species.

Model Optimization. Optimization is a crucial step in achieving accurate predictions from machine learning models. There are various optimizers , and one of the most effective ones is Adam. Adam (Adaptive Moment Estimation) is an optimization algorithm used in machine learning which updates network weights iteratively based on the training data as shown in Equation (12) and Equation (13). It utilizes the concept of first-order and second order moment

estimation gradients which continuously keep adjusting the learning rate of every parameter. The advantage of this process is that after bias correction , the learning rate comes within the specified range as mentioned in Equation (14) (Peng et al., 2022). This ensures greater stability in the parameters. It is computationally efficient as it requires less memory compared to other optimizers.

$$m_t = \beta_1 . m_{t-1} + (1 - \beta_1) . g_t, \quad (12)$$

$$v_t = \beta_2 . v_{t-1} + (1 - \beta_2) . g^2_t \quad (13)$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2} \quad (14)$$

$$\Delta\theta_t = -\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} . \eta \quad (15)$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1} \quad (16)$$

m_t, v_t represent the first and second order estimation of the gradient $\hat{m}_t$, $\hat{v}_t$ is the correction of m_t, v_t. β_1, β_2 are the learning rate of first and second order estimation, ε is the deviation, η depicts the learning rate of θ update as shown in equation (15) and (16).

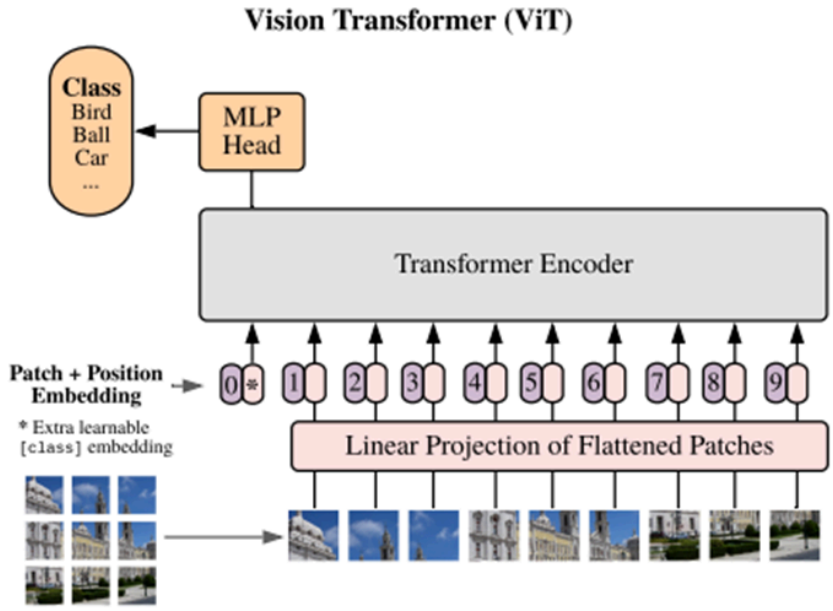
Vision Transformers

The fourth model selected for this study is the Vision Transformers (ViT). This innovative architecture, proposed by Dosovitskiy et al. (2021) leverages the transformer model's capabilities for image processing tasks. It initially split the input image of resolution (H, W) into N flattened patches of size (P x P pixels), where N = HW/P^2. Each patch is then linearly projected to a D-dimensional vector called a token. Positional embeddings are added to the

patches to retain spatial relationships between patches . Together, the sequence of N patch tokens and positional codes create the input representation for the transformer model. The figure 43 provided by Dosovitskiy et al. (2021) shows the entire architecture of the Vision Transformer model.

Figure 43

Architecture of ViT



The embeddings are sent to multi head attention in the form of Query (Q), Keys (K), and Values (V) by linearly projecting them along with token embeddings. Multi-head attention is a combination of multiple self-attention layers where each layer focuses weights at different parts of the image. These images with different weights are called heads which encapsulate different relationships in the image. The figure 44 provided by Khan et al. (2022) denotes the equation for multi-head attention where W^0, W_i^Q, W_i^K, W_i^V are weights for linear projection.

Figure 44

Equations representing Multi-head attention mechanism

$$MultiHead(Q,K,V) = Concat(head_1,\ldots,head_h)\,W^0$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Note. Equation denotes the representation of multi head attention mechanism

A single self-attention layer allows the model to incorporate global context by enabling each token to attend to all other tokens Vaswani et al. (2017). Attention scores are computed between queries and keys via a scaled dot product in this layer as shown in figure 45 was provided from Vaswani et al. (2017) where d_k is dimension of key vectors:

Figure 45

Self Attention mechanism formula

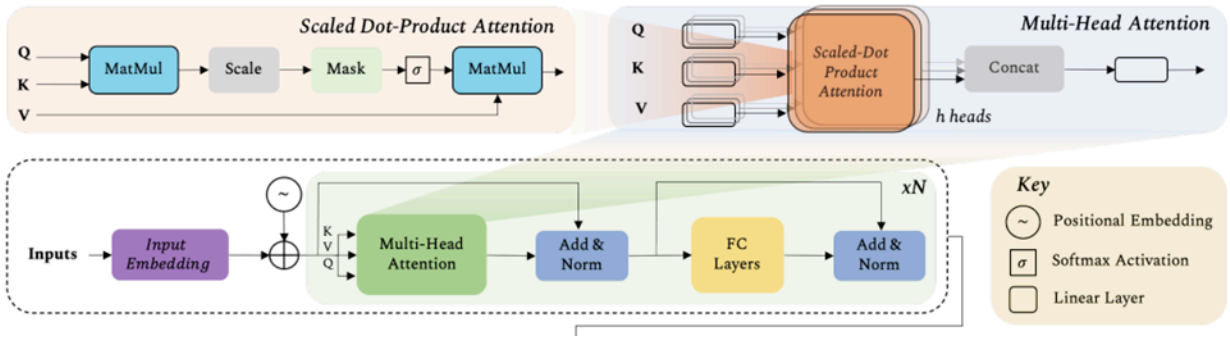
$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Note. Equation denotes the representation of self attention mechanism

The division by $\sqrt{d_k}$ allows smooth gradient flow during training. The softmax normalization then turns the scores into attention weights over the values. The model can effectively understand details and context to inform image classification. Multiple heads allow exponentially more relationships to be modeled. Figure 46 provided by Khan et al. (2022).shows the full architecture of the transformer encoder, multi head attention and self attention mechanism.

Figure 46

Transformer Encoder Architecture



Note. Detailed view layers in transformer encoder

The output of the Multi head attention layer is passed on to a Feed Forward network which comprises a simple 2-layer network with ReLU activation. This layer captures complex patterns and relationships and enriches patch-level representations individually Then layer normalization is applied and labels are predicted based on values. The figure 47 provided by Vaswani et al. (2017) denotes the equation for Feed Forward Network where W_1 *and* W_2 are linear transformations with dimensions, b_1 *and* b_2 are bias terms.

Figure 47

Equation of Multi layer Perceptron

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Note. Equation denotes the representation of feed forward network

The token embedding stores all the information that is related to the image. It gathers all details the model has extracted and stores it. The final layer of MLP uses this token embedding and creates a probability for all the classes. The class with the highest probability is assigned to the image.

Optimization. Vision transformers consist of parameters that can be optimized such that the model can be tuned so as to meet the requirement of the project. These parameters include patch length, learning rate, loss function and optimizer used.

Using ViT as a baseline, Ibrahimovic (2023) has carried out experiments in which different hyperparameters were tuned to see how they affect some selected metrics or dimensions. The CIFAR-100 dataset comprises 60000 color 32×32 images spanning over one hundred classes; they formed the basis of the experiments. The three main factors examined were the layers of the transformer, and the size of the image patches being considered as inputs. Increasing the number of transformer layers resulted in marked increase in accuracy for a given patch size at the expense of very long training time because of increased complexity. However, the improvement reduced after the parameter's threshold saturation. It was due to more granular representation that reducing the patch size provided benefits in terms of accuracy and convergence rate; however, those improvements ceased to improve at a resolution smaller than 8×8. Generally, in this instance the study showed that tweaking core ViT parameter settings increases accuracy, time, and customisation of a model suitable for image classification for an application context. These insights will help create optimized ViT architecture by providing ways for balancing parameter tuning and practical deployment of a model.

Based on literature review attention has to be given to hyperparameters to gauge the importance of model performance. The goal is to identify configurations that yield optimal results while minimizing training time-a critical factor in deploying efficient machine learning models. This iterative process enables a comprehensive understanding of how the model

responds to different hyperparameter settings and facilitates the discovery of a well-balanced configuration that aligns with the desired metrics and time constraints.

Model Supports

Hardware and Software Requirements

The project requires a basic setup of hardware and software for its implementation. For this purpose an i7 Intel processor with 256 GB internal storage is required to store the entire dataset as local backup and 16 GB RAM is desired to download datasets and perform basic processing. Along with these, an NVIDIA Tesla A100 GPU is required to accelerate machine learning processes. A detailed explanation of hardware used for research has been provided in Table 14.

Table 14

Hardware Requirements and Usage

Hardware	Configuration	Usage
CPU	Intel i7	High performance processor,ensures rapid execution of algorithms.
Storage	256GB	Ample space to store large datasets
RAM	16GB	Rapid data access and large datasets can be loaded into memory without disk swapping
Network Bandwidth	4G/5G Bandwidth	Provides rapid data transfer
GPU	NVIDIA Tesla A100	Faster execution of machine learning models

The list of softwares mentioned in table 15 have been used for proper execution and streamlining the project. A licensed Windows 11 OS is required to operate the machine. Sprint calls and team meetings were conducted on Zoom so that the team can stay up to date with tasks

in the project. A web browser like Chrome is used for researching and accessing online resources. Anaconda Distribution and Python are installed to perform data processing, cleaning and modeling. Few products from Microsoft Office such as MS Word, MS Powerpoint were used to document and prepare project reports.

Table 15

Softwares and versions used and their usage

Software	Specifications	Usage
Operating System	Windows 11	To provide GUI for hardware and manage resources for tasks
Zoom	5.16.6	Sprint meetings to provide updates and ensure alignment on assigned tasks
Chrome/Safari	Latest Version	Data research and working with tools
Python	3.10.0 or above	Programming language for data processing, cleaning and preparation
Anaconda Distribution	Anaconda 3	Platform for data processing and model building
MS Office	MS Office 2019	For documentation and reports

Tools and Libraries

Various python scripts were deployed in Jupyter notebook and open source python IDE These scripts utilized inbuilt libraries in python programming language aiming to process the data, build model and test in it various metrics collectively aiming to solve the research problem. Table 16 gives an in detail explanation of different libraries used in the project. Each library is chosen for its specific capability in handling image data. The ‘os’ and ‘shutil’ libraries facilitate path operations and file movement. For random sampling of images which is crucial for

unbiased training, a “random” library is utilized. “Scikit -Learn” offers various evaluation metrics while ‘’Matplotlib’ and “seaborn” are essential to visualize the data. The keras framework under TensorFlow has many layers and models which help in building and training deep learning architecture. ‘NumPy’’ and “PIL” support various numerical operations and image preprocessing techniques. Every tool is crucial for efficient data processing, model development and evaluation in the project.

Table 16

Libraries used in the project

Library		**Method**	**Usage**
os	os.path	join	Creating the path to a resource/file
		exists	Check whether a particular path exists or not
	os	listdir	List all the items within the input directory
		mkdir	Creates a new directory
shutil	shutil	copyfile	Move image files from source to destination
random	random	sample	Random Sampling of images
Scikit-Learn	sklearn.metrics	Confusion_matrix	Evaluation metrics
		classification_report	
Matplotlib	pyplot	show, figure, plot, legend	Plot results and evaluation graphs
Seaborn	seaborn	heatmap	Generate heatmap for confusion matrix for evaluation

Library		Method	Usage
Keras (tensorflow)	keras.layers	Dense	Used in deep learning network layers
		GlobalAveragePooling2D	
		Dropout	
	keras.models	Model	Build deep learning model
	keras.preprocessing.image	ImageDataGenerator	Generate augmented images for sampling
	keras.applications	MobileNet	Base model for deep learning model
		InceptionResNet	Employed pre-trained for Inception ResNet
	keras.layers	Conv2D	Pre-built layers for constructing CNN
		GlobalMaxPooling2D	
		Flatten	
		Dense	
		Dropout	
	loaded_model.predict		Making predictions on input data using a pre-trained model
	keras.optimizers	Adam	Optimizer to perform learning rate gradients
Transformers	ViTForImageClassification		For image classification tasks
	TrainingArguments		Define arguments such epochs, learning rate parameters for the model

Library		Method	Usage
	Trainer		Define train, test and validation datasets for learning
	ViTImageProcessor	from_pretrained	Loads a pre-trained Vision Transformer model
Torch	nn, stack, tensor		Neural network module, tensor values
Datasets	load_dataset	with_transform	Loads processed datasets
	load_metric	Accuracy, recall, F1 score, Precision	Metrics for evaluation
Numpy	numpy	argmax	Predicting class name label from output
PIL	ImageFilter	MedianFilter	Preprocessing on images
cv2 (opencv)	cv2	imread, imshow	Read and Display images

Model Architecture and Data Flow

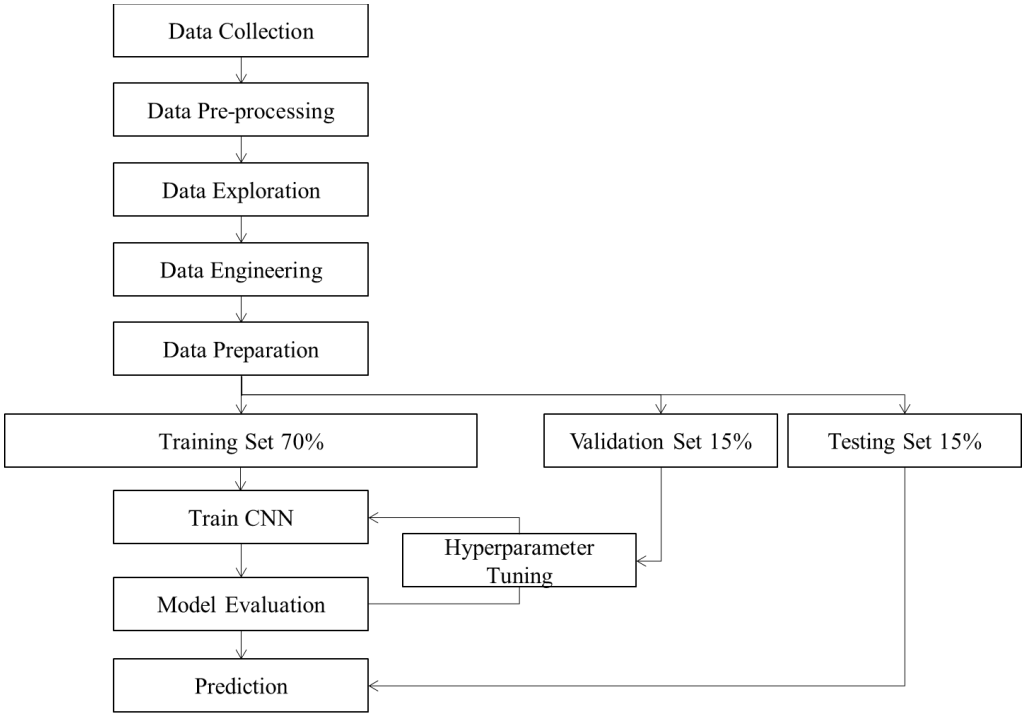
Convolution Neural Networks (CNN)

The image dataset was first collected, pre-processed, and split into training, validation, and testing sets. Figure 48 illustrates the process of training a CNN model for fish species classification. The training set served as input for training the CNN model, while the validation set evaluated its performance during training. Finally, the testing set assessed the final model performance independently. An iterative training, evaluation, and hyperparameter tuning process

ensured the CNN model to achieve optimal performance. Once the optimal model is determined, it can be used to predict new data.

Figure 48

Data Flow Overview



Alzubaidi et al. (2021) concluded that the performance of CNN is highly sensitive to the hyperparameter selection. Any small changes to hyperparameter values may have a significant impact on overall CNN performance. Therefore, careful hyperparameter selection is crucial during the optimization scheme development. After an iterative process of testing different numbers of layers and hyperparameters, the final model was determined.

Model Architecture

Building upon the previous algorithm, the final CNN model consists of 15 layers, including four convolutional layers, four max pooling layers followed by one flatten layer, and

four dense layers and two dropout layers. A clear architecture of CNN has been provided in Figure 49.

Figure 49

Final Model Architecture

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_5 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_6 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_6 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_7 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_7 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 conv2d_8 (Conv2D)           (None, 24, 24, 256)       295168

 max_pooling2d_8 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 flatten_1 (Flatten)         (None, 36864)             0

 dense_3 (Dense)             (None, 512)               18874880

 dropout (Dropout)           (None, 512)               0

 dense_4 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 128)               32896

 dense_6 (Dense)             (None, 37)                4773

=================================================================
Total params: 19,432,293
Trainable params: 19,432,293
Non-trainable params: 0
_________________________________________________________________
```

The four convolutional layers were used to extract features from the input image. Each convolutional layer employs ReLU activation for non-linearity. These layers have specific parameters customized for effective feature extraction. Conv2d_5 layer uses 32 filters with a 5x5 kernel size to capture initial features from the input image. Conv2d_6 layer uses 64 filters and a 3x3 kernel size; this layer builds upon the previous features by extracting more complex details.

Conv2d_7 layer uses 128 filters and a 3x3 kernel size, this layer further refines the extracted features, capturing increasingly intricate information. Conv2d_8 layer uses 256 filters with a 3x3 kernel size to perform the final stage of feature extraction, capturing the most valuable information for accurate classification.

Following each convolutional layer, a max pooling layer reduces the spatial dimensions of the feature maps. As described in the algorithm section, there are four max pooling layers down-sampling uses a 2×2 pool size and a stride of 2, reducing the size of the feature maps by half.

A single flatten layer was implemented to convert the feature maps into a single vector. Four dense layers, also known as the fully connected layers, are implemented to handle the classification task. Each layer has customized for optimal performance. Dense_3 layer consists of 512 neurons and employs ReLU as the activation function to introduce non-linearity and enhance the model's ability to learn complex decision boundaries. Dense_4 layer uses 256 neurons with ReLU activation to further refine the extracted features and prepare them for the final classification step. Dense_5 layer further reduces the dimensionality of the feature space by utilizing 128 neurons and ReLU activation. Dense_6 as the final layer, comprises 37 neurons, corresponding to the number of fish species the model is designed to identify. It employs Softmax activation to ensure the output is a probability distribution across all 37 classes.

Two dropout layers with a 0.5 dropout rate are implemented to prevent overfitting, as explained in the algorithm section. These randomly drop out neurons during training, reducing the network's reliance on any single feature.

MobileNet

MobileNet V2. This section describes the MobileNet architecture. MobileNet belongs to a class of efficient neural network family particularly designed for mobile devices and utilizes low computation and provides results in low latency. The typical architecture contains around 53 layers but can vary depending on parameters like resolution and width multipliers. After initial data processing and preparation, training and validation datasets are used for building the model. Unlike traditional convolutional architecture, MobileNet uses a unique approach during training on architecture, the basic convolution operation is replaced by two layers. These layers perform depth-wise convolution by using a single convolution for each channel followed by point-wise convolution which is a 1x1 convolution that creates new linear features. The activation tensors of the network's layers are treated as a group of three dimensional pixels. The use of ReLU causes complexity in reducing the dimensionality of activation space due to its non-linear transformations (Sandler et al., 2018). But if the input manifold is located in lower dimensional space, ReLU can preserve complete information. So, the non-linearity is eliminated from the residual block by introducing inverted residual blocks. A pictorial comparison of both blocks is shown in Figure 50. Each bottleneck layer has three major steps, expansion layer to increase channel depth thereby allowing for more intricate feature transformations, depth-wise convolution layer to split input to groups and extract spatial information, and finally projection layer to bottleneck the channels by reducing high dimensional space to lower dimensions. The input image is resized to a 224x224 shape that is accepted by the input layer of MobileNet by default, and flows through the entire architecture as outlined in Figure 51. The output is slightly modified to include an average pooling layer followed by fully connected layers and a dropout layer. The final layer is a fully connected layer with 37 nodes one for each class with softmax

activation function. When evaluating the model using a test dataset, the input image passes through all these layers and the softmax layer outputs probabilities for each of 37 classes. The class with the highest probability is the predicted class value.

Figure 50

Image showing difference between residual block and inverted residual block

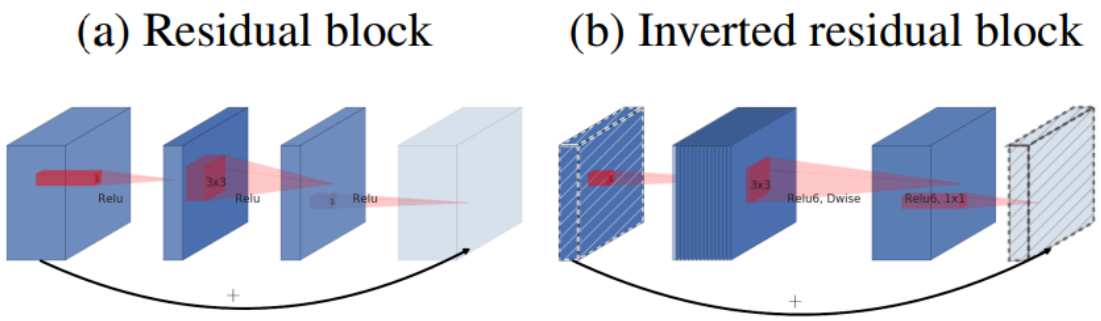
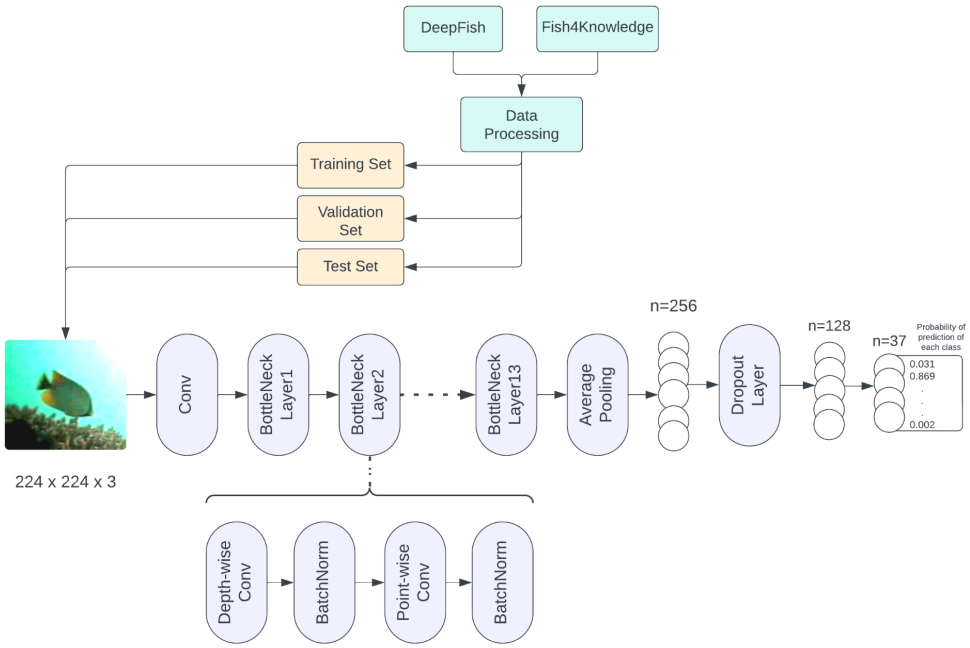


Figure 51

Architecture of MobileNet V2

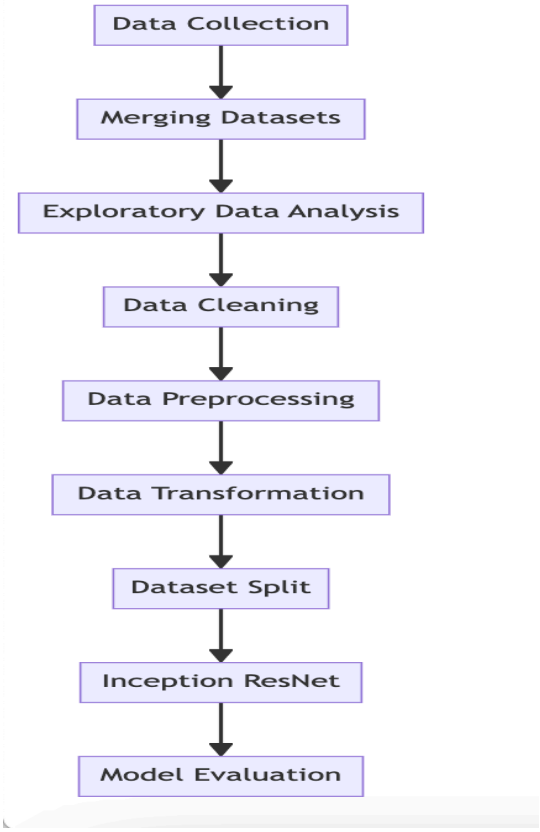


InceptionResNet

Figure 52 illustrates the data flow of the proposed inception ResNet model. The initial stage involves preprocessing the dataset and dividing it into training, testing and validation sets with a 70:15:15 ratio. Then this dataset is given as input to the Inception ResNet model. The Inception ResNet v2 model with pre-trained ImageNet weights is employed which leverages transfer learning to effectively utilize insights gained from one task to enhance learning in another task.

Figure 52

Data Flow of Inception ResNet Model

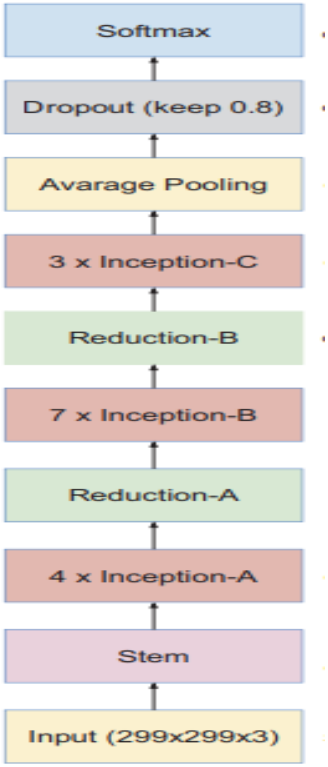


Note. The Above Figure Gives A Detailed Information Of Data Flow.

The model architecture is shown in Figure 53 for classification of fish species. The Inception ResNet network takes the input size of 299x299x3. The first part of the network, known as the "stem" involves multiple convolutions and pooling which helps in reducing dimensionality and featuring mapping. The next module in the Inception ResNet combines the features at multiple scales using parallel convolutions. As shown in figure 53, there are three main types in the network: Inception ResNet A , Inception ResNet B and Inception ResNet C (Szegedy et al., 2017).

Figure 53

Detailed Model Architecture Diagram



Note. The Above Figure Gives A Detailed Information Of Model Architecture

Inception ResNet A designed for 35x35 grid modules consisting of 1x1, 3x3, 5x5 convolutions. The output is given as an input to Reduction A which reduces the grid size to 17x17 while increasing depth. Each inception module includes residual connections which bypass convolution branches that help gradient flow through the network which takes care of the gradient descent problem. In Inception ResNet B consisting of 1x1, 7x7 convolutions combined with reduction B, the size is reduced to further to 8x8 .Similarly, Inception ResNet C consists of 8x8 grid modules featuring 1x1, 3x1 and 1x3 convolutions.

In each inception-ResNet and reduction combination, there will be reduction in spatial dimensions and the network undergoes deeper training to extract complex features. The final layers contain average pooling layers, dropout layers which help in regularization and a SoftMax layer for classification. Each module in the architecture is designed to ensure effective learning on challenging computer vision tasks. The architecture is designed to promote feature reuse which helps in learning complex patterns without again relearning representatives.

Building upon the pre-trained model, custom layers are implemented to fine tune the model for fish classification. A GlobalAveragePooling2D layer is the first layer on the base model which reduces each feature map by averaging out the spatial information. This step allows to reduce the total number of parameters in the model thereby preventing overfitting. Following the pooling layer, the next layer is the fully connected Dense layer with 1024 neurons and a ReLU activation function. This is a critical layer that helps models to learn complex patterns from the high level features extracted by the InceptionResNetV2 layers. The final Dense layer is in conjunction with the total number of classes in the datasets. This layer utilizes SoftMax activation function which outputs a probability distribution across all classes, a standard method

in multi classification tasks. This model is then compiled with Adam optimizer for faster convergence during training.

After the model is trained, the model is evaluated using the validation dataset and key performance metrics are extracted. These are compared against other proposed models.

Vision Transformers

For Vision Transformer, prior to modeling, the images are saved into another separate folder for further use. The flow in Figure 54 represents a sophisticated structure of data flow for image classification which was used in this research with an emphasis on fish species' identification. The process begins by dividing the images into small patches, each consisting of 16x16 pixels. These patches are shown visually with positional embeddings maintaining important spatial relations between patches for visualizing their understanding. Token embedding is also created along with positional embedding to store the image information.

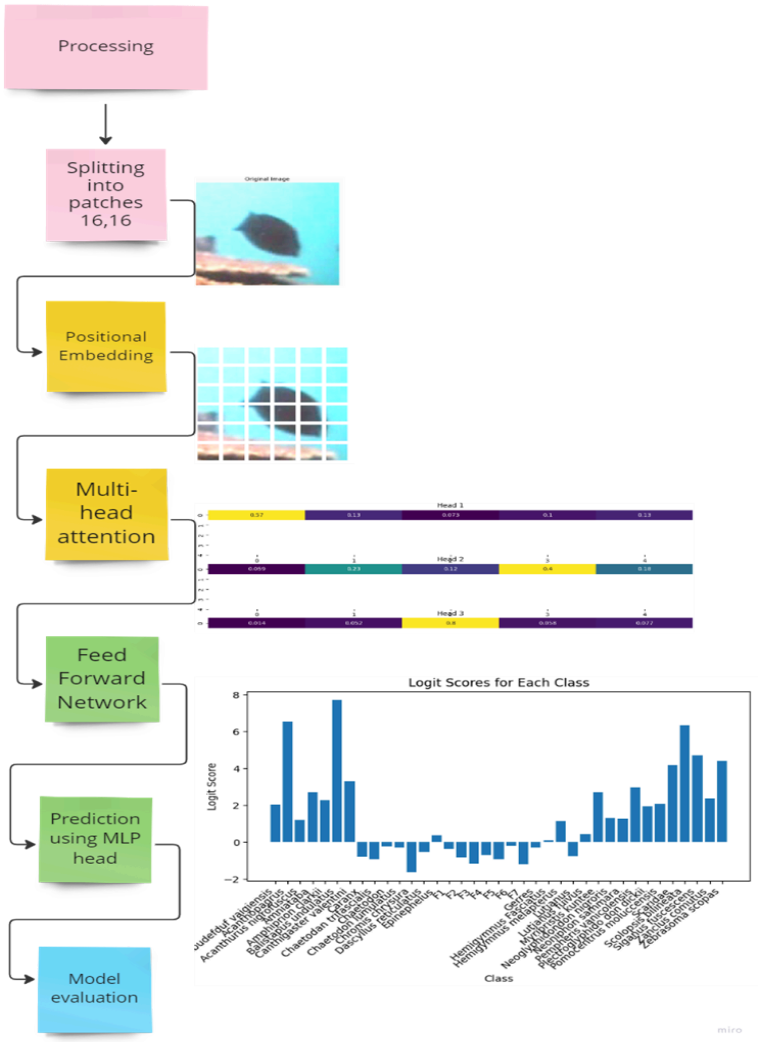
These patches are taken as input by the multi head attention where weights are applied to different parts of the image to understand the image and find in depth details. Multi-head attention is at the core of this network. Multi head attention is a combination of multiple self-attention mechanisms where attention is applied as a serial order Using the self-attention mechanism, these patches are redistributed across multiple heads. Attention maps arising from these images demonstrate differences in focused attention which is manifested in vivid yellows for particular areas of interest. Thus minute details that are critical for correct identification, and among various fishes species are captured by varying attention on patches.

As soon as the process of self-attention is complete, the pictures are fed forward into the network. This is when the model uses a MLP to predict and merge the features extracted by the

Self-attention mechanism. This is where non-linearity takes place so that it can discover complex shapes and spatial relationships between visual elements. Through a synergetic approach where self-attention is coupled with the feedforward network operations, the models learn strong and dissimilar representations. The MLP layer has the information of the image as a whole not only as patches thereby allowing it to understand all the information collected in the previous step.

Figure 54

Data flow of Model



After integrating information from the token embedding and MLP layer, the model makes a final prediction on the class of the input image. This last prediction step is illustrated in Figure 54, which shows the output probabilities for several fish species.

Finally, the model is applied on the testing data set and evaluated based on accuracy, recall, precision, confusion matrix to assess the efficiency of the model. By passing through every phase, the acquired and computed information enables the model to forecast correctly as it demonstrates the capability of vision transformers in classifying images with different kinds of fishes. All the code for modeling stage is shown in Appendix B.

Model Comparison and Justification

The purpose of the model comparison is to find the model most suited for fish species identification tasks. As acknowledged by Kelleher et al. (2015), no model will ever be perfect. Many more factors may have to be considered than just simply measuring model performance. Therefore, a comprehensive evaluation is conducted including criterias like model architecture, data type compatibility, performance on small data, model complexity, limitations and the overall trade-off between these considerations.

Based on the literature review and input from other researchers, Table xx compares four machine learning models implemented in this project. Among these models, Vision Transformers excel at capturing global dependencies and are transferable across various tasks. However, they require large datasets for optimal performance, struggle with small data, and are computationally expensive and time-consuming to train. Inception ResNet is efficient with smaller datasets and robust against overfitting, but it may not capture long-range dependencies well. It is resource efficient and relatively fast to train. MobileNet is lightweight and performs well with small data.

However, it is less efficient for complex patterns. Finally for CNN, this is widely used and adaptable, it requires moderate-sized to large dataset to learn well and may not capture complex patterns as effectively. It requires moderate training times and resource requirements. In depth comparison of models is listed in Table 17.

Table 17

Comparison of Models

Characteristic	**Vision Transformers**	**InceptionResNet**	**MobileNet**	**CNN Model**
Basic Architecture	Transformer-based architecture with self-attention mechanism	Hybrid architecture combining Inception blocks and ResNet	Supports parallel processing	Supports parallel processing
Data Types	Primarily designed for image data	Image data	Designed for image data	Can handle image, audio, text, time series data
Performance on Small Data	Generally requires large datasets for optimal performance	Efficient on smaller datasets, especially with transfer learning	Can perform well with limited training data	Requires moderate-sized to large datasets to learn well
Overfitting/Under fitting	Susceptible to overfitting, especially with limited data	Robust against overfitting, especially with residual connections	Usually not prone to underfitting	Improper parameters can lead to underfitting
Complexity	Moderate preprocessing, requires image patching and positional encoding	Moderate preprocessing, multi-level feature extraction	Configured to run with lower complexity	Simple model with less than 15 layers

Characteristic	Vision Transformers	InceptionResNet	MobileNet	CNN Model
Space Complexity	Requires more memory due to the self-attention mechanism	Moderate memory requirements	Trained model is around 40 MB in size	Trained model size is approximately 220 MB
Computational Complexity	Computationally intensive, benefits from GPU acceleration	Efficient GPU utilization, faster inference	Accelerates training on special hardware like GPUs	Requires GPUs to train faster
Strengths	Effective at capturing global dependencies in images, Transferable across diverse tasks	Excellent feature extraction with multiple pathways	Quick to get started and running	Can be customized and built from scratch
Limitations	Resource-intensive, especially for large images, Interpretability challenges	May not capture long-range dependencies well	Difficult to capture complex patterns	Finding the right combination of parameters
Optimizer	Adam	Adam	Adam	Adam
Parameters	4.73 M	5.7 M	3.5 M	19.4 M
Training Time	120 min	30 min	63 min	45 min
Inference Time	Decent speed in predicting new data points	Faster prediction time	Low latency and predicts fastest	Depends on the width of the network but typically slower

Model Evaluation Methods

Evaluation of implemented models gives an overview of their performance. The performance also gives details on model understanding of the data and its limitations. The metric

values during evaluation indicate the model understanding on given data. As the research problem is classification evaluation metrics such as accuray, precision, recall and F1-score have been used. There are other metrics which such as ROC curve and Cross validation for classification problems but being a large data set it would require huge computational resources to recompute the metrics.

Accuracy

It is one of the most common metrics used in classification tasks. It is calculated as the ratio of correct predictions to the total number of predictions and is given by formula (17). It represents the total performance of the model and lies between zero and one where zero is the worst and one is the highest. Usually converted to percentages for better notation.

$$\text{Accuracy} = \frac{(\,True\ Positives + True\ Negatives\,)}{(\,True\ Positives + False\ Positives + True\ Negatives + False\ Negatives\,)} \tag{17}$$

Precision

Precision is the ratio of correct positive predictions to the total number of predictions predicted as positive class. It is given by the formula (18).

$$\text{Precision} = \frac{(\,True\ Positives\ \,)}{(\,True\ Positives + False\ Positives\,)} \tag{18}$$

Recall

Recall or Sensitivity is calculated as a fraction of true positive predictions over total positive examples. It is given by the formula (19). It quantifies the model's capability to make accurate predictions on positive class.

$$\text{Recall} = \frac{(\,True\ Positives\,)}{(\,True\ Positives + False\ Negatives\,)} \tag{19}$$

F1-score

For use cases where both precision and recall are essential, this metric can be used to strike a balance. F1 score is the harmonic mean of precision and recall as given in formula (20). It is suitable for measuring performance while trained on imbalanced datasets.

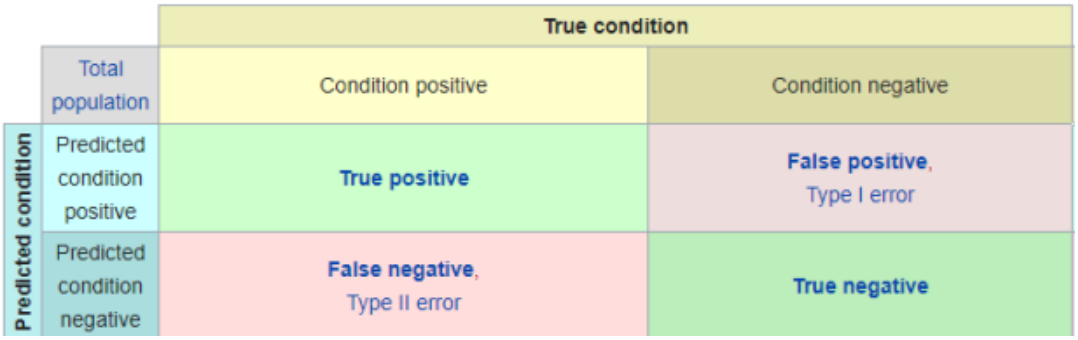
$$\text{F1 Score} = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \quad (20)$$

Confusion matrix

It is a type of square matrix that shows the count of correct and incorrect predictions for each class within the dataset. The shape of the matrix depends on the number of classes the model is trained on. In this context, the shape of the confusion matrix is 37 x 37 as the model is capable of classifying input images to one of the 37 classes. Considering a binary classification task, True Positives represent the number of positive examples predicted as positive and similarly True Negatives represent the number of negative examples predicted as negative. False Positives are counts of negative class instances predicted as positive and False Negatives are vice-versa. A sample confusion matrix for a binary classification problem is shown in Figure 55 (E. Chan, personal communication, November 20, 2023).

Figure 55

Confusion Matrix for a binary classification task



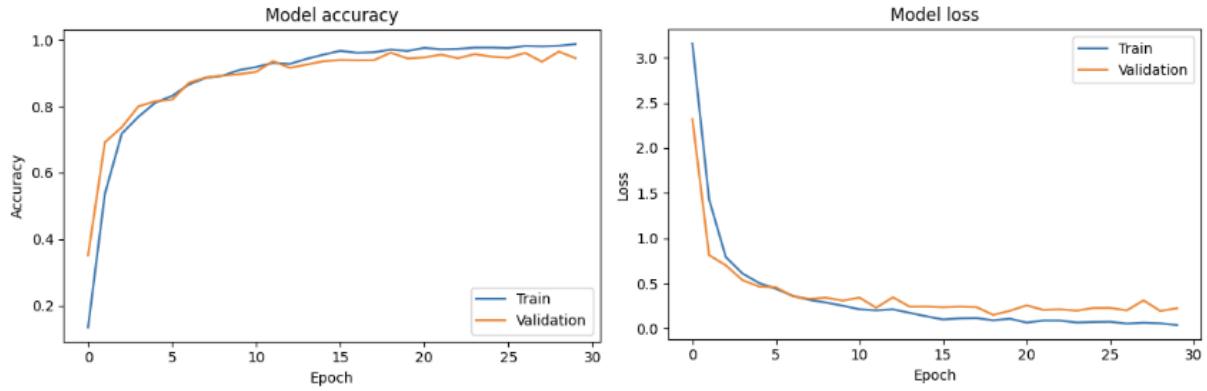
Model Validation and Evaluation

CNN

The model was trained for 30 epochs with a learning rate of 0.001 and the optimizer used was Adam. Training time was 25 minutes. This model provided the best overall results of 98% of recall, precision and F1-score. The inference time to calculate predictions in the test dataset is higher at 48.2 seconds. Figure 56 shows the accuracy and loss curves for training and validation data for this model. The confusion matrix of the model is shown in Appendix C1.

Figure 56

Training and Validation accuracy and loss curves for CNN model

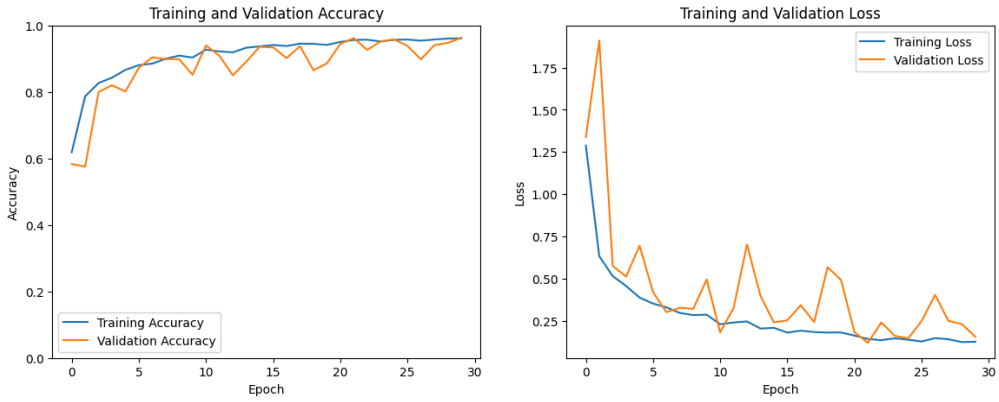


MobileNet

The model has the second best accuracy of 95.9% on test data, 96% precision, 95.6% recall and 95.8% F1 score after training for 30 epochs with a learning rate of 0.01 with Adam optimizer which took 65 minutes. This model has shown the best inference time of 9.3 seconds. Figure 57 shows the accuracy, loss curves of train and validation data.

Figure 57

Training and Validation accuracy and loss curves for MobileNet model



InceptionResNet

The model was trained for 25 epochs with a learning rate of 0.001 and optimizer used was Adam. Training time was 30 minutes. This model has shown decent accuracy of 93.6% on test data. The average precision and recall values lie between 94-95%. It has a good inference time of 18.4 seconds. Figure 58 shows the accuracy and loss curves for training and validation data for this model. The confusion matrix of the model is shown in Appendix C3.

Figure 58

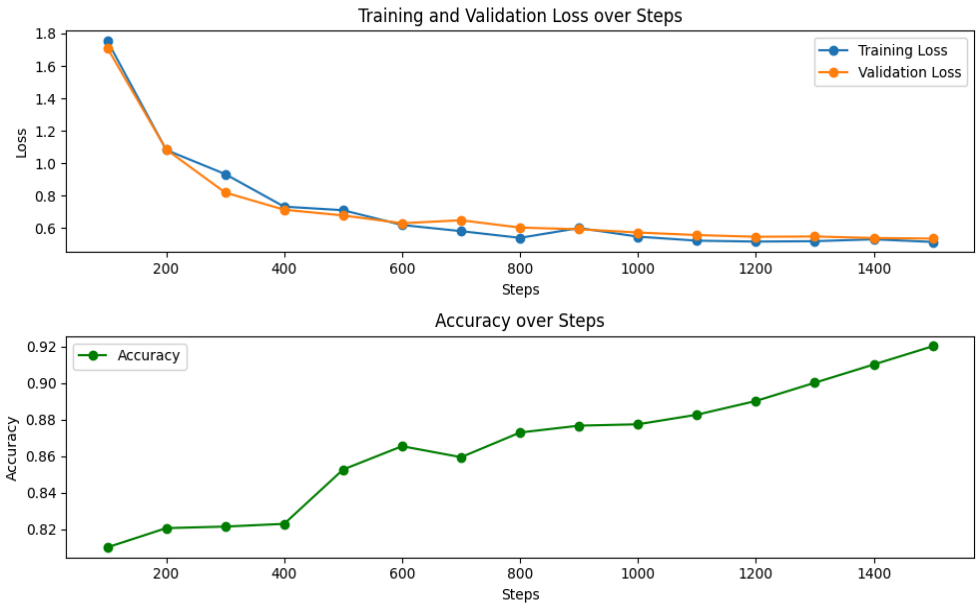
Training and Validation accuracy and loss curves for InceptionResNet model

Vision Transformers

Despite the lesser number of epochs, the model has a decent accuracy of 92% with 21 minutes of training. Figure 59 shows the loss of training and validation datasets and accuracy on test dataset for the ViT model.

Figure 59

Training and Validation loss and testing accuracy curves for Vision Transformers model



In comparison of proposed machine learning models, their performance was assessed using various key metrics. MobileNet is the top performer, having the highest accuracy of 0.9598. This highlights the algorithm proficiency in making accurate predictions and its future potential in real world scenarios. Complementing this accuracy, it also has admirable precision score of 0.9608 and recall score of 0.9567 illustrating the models capability in capturing actual positive cases. Following MobileNet, CNN model has highest accuracy of 0.9489. Its precision score of 0.98 which is highest among all which testifies its capacity to minimize false positives. It has a recall score of 0.98 indicating well rounded performance of the model. Inception ResNet

has modest accuracy of 0.936 but has lower precision and recall value of 0.95 and 0.94 respectively compared to CNN and MobileNet. Vision Transformers exhibits performance that lags behind all other models, demonstrating an accuracy 0.92. Its recall and precision value is 0.92 which is also least in comparison to other models suggesting vision transformers may generate more false positives. Table 18 depicts the overall comparison of machine learning models used in the project.

Table 18

Comparison of evaluation of all the models

Metric / Model	CNN	MobileNet	Inception-ResNet	Vision Transformer
Accuracy	0.9489	0.9598	0.936	0.92
Precision	0.98	0.9608	0.95	0.92
Recall	0.98	0.9567	0.941	0.92
F1-Score	0.9808	0.9587	0.941	0.91

Note. All four models performance across various evaluation metrics.

Conclusion

Based on the research, implementation and evaluation it can be concluded that the ViT model was able to classify the images appropriately but required a lot of time for the model to get adapted to the data. InceptionResNet, a combination of Inception and ResNet models, classified the images with an accuracy of 93.6%. The MobileNet model outperformed all the models with an accuracy of 95.9% making it a reliable choice for real time predictions due to low latency and efficient computation. CNN model provided an accuracy 94.8% but its

precision, recall and F1 score exceeded other models with 98%. The study underscored the potential of the models and their handling of new unseen data. MobileNet has efficiently classified new images correctly indicating that of all the models it was able to find the discern patterns lying within the images.

Limitations

The main limitations of all the models were computational demands. All the models required GPU resources for faster execution and took at least greater than 30 minutes to completely understand the images. ViT model required a lot of computation time when compared with all the other models, rolling it out of option for a real time analysis. Although CNN had better metrics except for accuracy as they are prone to overfitting impacting the performance of the model. InceptionResNet has high latency of all the models and is sensitive to outliers and overfitting of unseen data for large datasets. The limitations underscore the computational resources as a major drawback of the model indicating the necessity for enhancement in model architecture to balance the accuracy and resources.

Future Scope

For future the project intends to create a pool of unlabeled data such that they can be corrected and the model can be retrained making the model more efficient to unseen data. The combination of models is another prospect to be looked into through which one model balances the drawbacks of the other thereby enhancing the performance. This combination also helps in creating lightweight architecture that can be deployed on edge computing devices for real time analysis and faster results. This data when combined with weather near the captured regions gives an analysis of migration patterns of fishes.

References

Agarwal, A. K., Kiran, V., Jindal, R. K., Chaudhary, D., & Tiwari, R. G. (2022). Optimized Transfer Learning for Dog Breed Classification. *International Journal of Intelligent Systems and Applications in Engineeri*, *10*(1s), 18–22

Akbuğday, B. (2019). Classification of breast cancer data using machine learning algorithms. *2019 Medical Technologies Congress (TIPTEKNO)*. https://doi.org/10.1109/tiptekno.2019.8895222

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *IEEE. 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey.* https://doi.org/10.1109/icengtechnol.2017.8308186

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A. Q., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00444-8

Ananda, A., Ngan, K. H., Karabağ, C., Ter-Sarkisov, A., Alonso, E., & Reyes-Aldasoro, C. C. (2021). Classification and Visualisation of Normal and Abnormal Radiographs; A Comparison between Eleven Convolutional Neural Network Architectures. *Sensors*, *21*(16), 5381. https://doi.org/10.3390/s21165381

Bardoliwala, D. N., Desai, M. V., & Shanbhag, A. D. (2023). The Impact of Missing Data Handling Techniques on the Accuracy of COVID-19 Mortality Prediction. *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)* 1-6. https://doi.org/10.1109/ICECCME57830.2023.10253044

Chen, J., Shan, S., He, C., Zhao, G., Pietikäinen, M., Chen, X., & Gao, W. (2009). WLD: A robust local image descriptor. IEEE transactions on pattern analysis and machine intelligence, 32(9), 1705-1720. https://doi.org/10.1109/TPAMI.2009.155

Dablain, D. A., & Chawla, N. V. (2023). Towards Understanding How Data Augmentation Works with Imbalanced Data. *arXiv preprint arXiv:2304.05895*. https://doi.org/10.48550/arXiv.2304.05895

Das, C., Sahoo, A. K., & Pradhan, C. (2022). Multicriteria recommender system using different approaches. In *Elsevier eBooks*, 259–277. https://doi.org/10.1016/b978-0-323-85117-6.00011-x

Deely, J., Hynes, S., & Cawley, M. (2022). Overseas visitor demand for marine and coastal tourism. *Marine Policy*, *143*, 105176. https://doi.org/10.1016/j.marpol.2022.105176

Dorfman, E. (2022, March 25). *How Much Data Is Required for Machine Learning?* Postindustria. https://postindustria.com/how-much-data-is-required-for-machine-learning/

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations*. https://openreview.net/pdf?id=YicbFdNTTy

Ferreira, C., Melo, T., Sousa, P., Meyer, M. I., Shakib Pour, E., Costa, P., & Campilho, A. (2018). Classification of breast cancer histology images through transfer learning using a pre-trained inception ResNet V2. In *Lecture Notes in Computer Science* (pp. 763–770). https://doi.org/10.1007/978-3-319-93000-8_86

Fisher, R. B., Shao, K. T., & Chen-Burger, Y. (2016). Overview of the Fish4Knowledge project. In *Intelligent systems reference library*,1–17. https://doi.org/10.1007/978-3-319-30208-9_1

Fisher, R. B., Shao, K. T., & Chen-Burger, Y. H. (2016). Overview of the fish4knowledge project. *Fish4Knowledge: collecting and analyzing massive coral reef fish video data*, 1-17. https://doi.org/10.1007/978-3-319-30208-9_1

Foster, N. (2022, September 23). The Evolution of the GPU: How It Became the Heart of AI and ML. Ace Cloud. https://www.acecloudhosting.com/blog/evolution-of-gpu/

Geng, Z., & Wang, Y. (2020). Automated design of a convolutional neural network with multi-scale filters for cost-efficient seismic data classification. *Nature Communications*, *11*(1). https://doi.org/10.1038/s41467-020-17123-6

Ghosh, S., Das, N., Das, I., & Maulik, U. (2019). Understanding deep learning techniques for image segmentation. *ACM Computing Surveys*, *52*(4), 1–35. https://doi.org/10.1145/3329784

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for Multi-Class Classification: an Overview. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2008.05756

Gupta, S., & Gupta, A. (2019). Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, *161*, 466-474. https://doi.org/10.1016/j.procs.2019.11.146

Han, G., Long, X., Zhu, C., Guizani, M., & Zhang, W. (2019). A high-availability data collection scheme based on multi-AUVs for underwater sensor networks. IEEE Transactions on Mobile Computing, 19(5), 1010-1022. https://doi.org/10.1109/TMC.2019.2907854

Highsmith, J. (2004). *Agile Project Management: creating innovative products*. http://cds.cern.ch/record/1518321

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: efficient convolutional neural networks for mobile vision applications. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1704.04861

Ibrahimovic, E. (2023). Optimizing Vision Transformer Performance with Customizable Parameters. *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*. https://doi.org/10.23919/mipro57284.2023.10159761

Iqbal, M., Wang, Z., Ali, Z. A., & Riaz, S. (2019). Automatic fish species classification using deep convolutional neural networks. *Wireless Personal Communications*, *116*(2), 1043–1053. https://doi.org/10.1007/s11277-019-06634-1

Iqbal, Z., Khan, M. A., Sharif, M., Shah, J. H., Rehman, M. H. U., & Javed, K. (2018). An automated detection and classification of citrus plant diseases using image processing techniques: A review. *Computers and Electronics in Agriculture*, *153*, 12–32. https://doi.org/10.1016/j.compag.2018.07.032

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., & Shah, M. (2022). Transformers in Vision: a survey. *ACM Computing Surveys*, *54*(10s), 1–41. https://doi.org/10.1145/3505244

Knausgård, K. M., Wiklund, A., Sørdalen, T. K., Halvorsen, K. T., Kleiven, A. R., Jiao, L., & Goodwin, M. (2021). Temperate fish detection and classification: a deep learning based approach. *Applied Intelligence*, *52*(6), 6988–7001. https://doi.org/10.1007/s10489-020-02154-9

Kohavi, R. (03 2001). *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. *14*. https://roboticsStanfordedu/"ronnyk

Kühn, M., & Johnson, K. (2013). Applied Predictive Modeling. In *Springer eBooks*. https://doi.org/10.1007/978-1-4614-6849-3

Kumar, A., & Sodhi, S. S. (2020). Comparative analysis of gaussian filter, median filter and denoise autoenocoder. *International Conference on Computing for Sustainable Global Development (INDIACom)* 45-51. https://doi.org/10.23919/INDIACom49435.2020.9083712

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In *Lecture Notes in Computer Science* (pp. 21–37). https://doi.org/10.1007/978-3-319-46448-0_2

Log in with Atlassian account. (n.d.). https://tiffany-li.atlassian.net/jira/software/projects/D2G5/apps/787143b2-23f5-41c2-ba67-91994c652ab7/47e35f85-18b2-4af3-a1ba-88ce6e019565

Manivannan, S., & Venkateswaran, N. (2023). Dog Breed Classification using Inception-ResNet-V2. *International Conference for Advancement in Technology (ICONAT) (Pp. 1–5). IEEE*. https://doi.org/10.1109/iconat57137.2023.10080065

Methods in Ecology and Evolution, 9(11), 2216–2225.

Nichol, A., Achiam, J., & Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*. https://doi.org/10.48550/arXiv.1803.02999

Noda, J. J., Travieso, C. M., & Sánchez-Rodríguez, D. (2016). Automatic taxonomic classification of fish based on their acoustic signals. Applied Sciences, 6(12), 443. https://doi.org/10.3390/app6120443

Nurhopipah, A., & Hasanah, U. (2020). Dataset splitting techniques comparison for face classification on CCTV images. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, *14*(4), 341-352. https://doi.org/10.22146/ijccs.58092

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*. https://doi.org/10.48550/arXiv.1811.03378

Obaid, H. S., Dheyab, S. A., & Sabry, S. S. (2019). The impact of data pre-processing techniques and dimensionality reduction on the accuracy of machine learning. *Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, 279-283. https://doi.org/10.1109/IEMECONX.2019.8877011

Pan, H., Pang, Z., Wang, Y., Wang, Y., & Chen, L. (2020). A new image recognition and classification method combining transfer Learning algorithm and MobileNet model for welding defects. *IEEE Access*, *8*, 119951–119960. https://doi.org/10.1109/access.2020.3005450

Peña, J. M., Gutiérrez, P. A., Hervás‐Martínez, C., Six, J., Plant, R. E., & López‐Granados, F. (2014). Object-Based Image Classification of Summer Crops with Machine Learning Methods. *Remote Sensing*, *6*(6), 5019–5041. https://doi.org/10.3390/rs6065019

Peng, C., Liu, Y., Yuan, X., & Chen, Q. (2022). Research of image recognition method based on enhanced inception-ResNet-V2. *Multimedia Tools and Applications*, *81*(24), 34345–34365. https://doi.org/10.1007/s11042-022-12387-0

Perronnin, F., Sánchez, J., & Mensink, T. (2010). Improving the Fisher Kernel for Large-Scale Image Classification. *Computer Vision – ECCV 2010*, 143–156. https://doi.org/10.1007/978-3-642-15561-1_11

Prasetyo, E., Suciati, N., & Fatichah, C. (2022). Multi-level residual network VGGNet for fish species classification. *Journal of King Saud University - Computer and Information Sciences*, *34*(8), 5286–5295. https://doi.org/10.1016/j.jksuci.2021.05.015

Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., & Dosovitskiy, A. (2021). Do vision transformers see like convolutional neural networks? *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2108.08810

Rathi, D., Jain, S., & Indu, S. (2017). Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning. *2017 Ninth International Conference on Advances in Pattern Recognition (ICAPR)*. https://doi.org/10.1109/icapr.2017.8593044

Rauf, H. T., Lali, M. I. U., Zahoor, S., Shah, S. Z. H., Rehman, A. U., & Bukhari, S. (2019). Visual features based automated identification of fish species using deep convolutional neural networks. *Computers and Electronics in Agriculture*, *167*, 105075. https://doi.org/10.1016/j.compag.2019.105075

Raveendran, S., Patil, M. D., & Birajdar, G. K. (2021). Underwater image enhancement: a comprehensive review, recent trends, challenges and applications. *Artificial Intelligence Review*, *54*, 5413-5467. https://doi.org/10.1007/s10462-021-10025-z

Rodrawangpai, B., & Daungjaiboon, W. (2022). Improving text classification with transformers and layer normalization. *Machine Learning With Applications*, *10*, 100403. https://doi.org/10.1016/j.mlwa.2022.100403

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*, 211-252. https://doi.org/10.48550/arXiv.1409.0575

Saleh, A., Laradji, I. H., Konovalov, D. A., Bradley, M., Vazquez, D., & Sheaves, M. (2020). A realistic fish-habitat dataset to evaluate algorithms for underwater visual analysis. *Scientific Reports*, *10*(1), 14671. https://doi.org/10.1038/s41598-020-71639-x

Saleh, A., Laradji, I. H., Konovalov, D. A., Bradley, M., Vazquez, D., & Sheaves, M. (2020). A realistic fish-habitat dataset to evaluate algorithms for underwater visual analysis. *Scientific Reports*, *10*(1), 14671. https://doi.org/10.1038/s41598-020-71639-x

Saleh, A., Laradji, I. H., Konovalov, D. A., Bradley, M., Vazquez, D., & Sheaves, M. (2020). A realistic fish-habitat dataset to evaluate algorithms for underwater visual analysis. *Scientific Reports*, *10*(1), 14671. https://doi.org/10.1038/s41598-020-71639-x

Salman, A., Jalal, A., Shafait, F., Mian, A., Shortis, M., Seager, J., & Harvey, E. (2016). Fish species classification in unconstrained underwater environments based on deep learning. Limnology and Oceanography: Methods, 14(9), 570-585. https://doi.org/10.1002/lom3.10113

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510-4520. https://doi.org/10.48550/arXiv.1801.04381

Serin, G., Sener, B., Ozbayoglu, A. M., & Unver, H. O. (2020). Review of tool condition monitoring in machining and opportunities for deep learning. *The International Journal of Advanced Manufacturing Technology*, *109*, 953-974.

Siami-Irdemoosa, E., Dindarloo, S. R., & Sharifzadeh, M. (2015). Work breakdown structure (WBS) development for underground construction. *Automation in Construction*, *58*, 85–94. https://doi.org/10.1016/j.autcon.2015.07.016

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). Inception-V4, Inception-ResNet and the impact of residual connections on learning. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1602.07261

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-V4, Inception-ResNet and the impact of residual connections on learning. *Proceedings of the . . . AAAI Conference on Artificial Intelligence*, *31*(1). https://doi.org/10.1609/aaai.v31i1.11231

Talebi, H., & Milanfar, P. (2021). Learning to resize images for computer vision tasks. In *Proceedings of the IEEE/CVF international conference on computer vision*. 497-506. https://doi.org/10.48550/arXiv.2103.09950

Tharwat, A., Hemedan, A. A., Hassanien, A. E., & Gabel, T. (2018). A biometric-based model for fish species classification. Fisheries research, 204, 324-336. https://doi.org/10.1016/j.fishres.2018.03.008

Tian, M., Luo, S. W., & Liao, L. Z. (2005). An investigation into using singular value decomposition as a method of image compression. *International Conference on Machine Learning and Cybernetics*, 8, 5200-5204. https://doi.org/10.1109/ICMLC.2005.1527861

Tomasi, C., & Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Sixth international conference on computer vision*, 839-846. https://doi.org/10.1109/ICCV.1998.710815

Vasudevan, H., Kottur, V. K. N., & Raina, A. A. (2020). *Proceedings of International Conference on Intelligent Manufacturing and Automation*. Springer Nature. http://books.google.ie/books?id=2JjuDwAAQBAJ&printsec=frontcover&dq=Proceedings+of+International+Conference+on+Intelligent+Manufacturing+and+Automation.+(2020).&hl=&cd=1&source=gbs_api

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. *arXiv (Cornell University)*, *30*, 5998–6008. https://arxiv.org/pdf/1706.03762v5

Villon, S., Iovan, C., Mangeas, M., Claverie, T., Mouillot, D., Villéger, S., & Vigliola, L. (2021). Automatic underwater fish species classification with limited data using few-shot learning. *Ecological Informatics*, *63*, 101320. https://doi.org/10.1016/j.ecoinf.2021.101320

Wäldchen, J., & Mäder, P. (2018). Machine learning for image based species identification. *Methods in Ecology and Evolution*, *9*(11), 2216–2225. https://doi.org/10.1111/2041-210x.13075

Wang, D., He, X., Wei, Z., & Yu, H. (2009). A method of aircraft image target recognition based on modified PCA features and SVM. 2009 9th International Conference on Electronic Measurement & Instruments, 4-177-4–181. https://doi.org/10.1109/icemi.2009.5274100

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, *1*, 29–39.

Yao, G., & Huang, R. (2023). An image matting algorithm based on Inception-ResNet-V2 network. In *Smart innovation, systems and technologies* (pp. 323–334). https://doi.org/10.1007/978-981-99-3416-4_26

Yuan, H., Cheng, J., Wu, Y., & Zeng, Z. (2022). Low-res MobileNet: An efficient lightweight network for low-resolution image classification in resource-constrained scenarios. Multimedia Tools and Applications, 81(27), 38513–38530. https://doi.org/10.1007/s11042-022-13157-8

Zhang, H., Berg, A. A., Maire, M., & Malik, J. (2006). SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition. *Computer Vision and Pattern Recognition*, *2*, 2126-2136. https://doi.org/10.1109/cvpr.2006.301

Zhang, J., Shao, K., & Luo, X. (2018). Small sample image recognition using improved Convolutional Neural Network. *Journal of Visual Communication and Image Representation*, *55*, 640–647. https://doi.org/10.1016/j.jvcir.2018.07.011

Zhu, K., Hui, H., & Huang, H. (2018). Underwater object Images Classification Based on Convolutional Neural Network. *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*. https://doi.org/10.1109/siprocess.2018.8600472

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... & He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, *109*(1), 43-76. https://doi.org/10.1109/JPROC.2020.3004555

Appendix A

Data Engineering

Figure A1

Code for Data Pre-processing

```
def balance_dataset(input_folder, output_folder, target_count):
    # Create output folder if not exists
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    # Get the list of class folders
    class_folders = sorted(os.listdir(input_folder))
    # Loop through each class folder
    for class_folder in class_folders:
        class_path = os.path.join(input_folder, class_folder)
        # Get the list of images in the current class folder
        images = os.listdir(class_path)
        # Calculate the number of images to generate to reach the target count
        current_count = len(images)
        images_to_generate = target_count - current_count
        # Perform undersampling if needed
        if current_count > target_count:
            sampled_images = random.sample(images, target_count)
        else:
            sampled_images = images
        # Create an ImageDataGenerator for data augmentation
        datagen = ImageDataGenerator(rotation_range=20, width_shift_range=0.2,
            height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True,
            fill_mode='nearest'        )
       # Loop through sampled images and generate augmented images
        for image_name in sampled_images:
            image_path = os.path.join(class_path, image_name)
            img = load_and_preprocess_image(image_path)  # You need to implement this function
            img = img.reshape((1,) + img.shape)  # Reshape for flow method
            # Generate augmented images
            i = 0
            for batch in datagen.flow(img, batch_size=1, save_to_dir=output_folder, save_prefix=class_folder, save_format='jpg'):
                i += 1
                if i >= images_to_generate:
                    break
```

Figure A2

Code for Data Transformation

```
def preprocess_image(image_path, output_folder, target_size=(256, 256)):
    # Read the image
    img = cv2.imread(image_path)

    # Resize the image to the target size
    img_resized = cv2.resize(img, target_size)

    # Apply bilateral filter
    img_filtered = cv2.bilateralFilter(img_resized, d=9, sigmaColor=75, sigmaSpace=75)

    # Normalize pixel values to the range [0, 1]
    img_normalized = img_filtered / 255.0

    # Apply histogram equalization for image enhancement
    img_enhanced = exposure.equalize_adapthist(img_normalized)

    # Perform Singular Value Decomposition (SVD) with 10 components
    svd = TruncatedSVD(n_components=10, random_state=42)
    img_svd = svd.fit_transform(img_enhanced.reshape(-1, 3))

    # Reshape back to the original image shape
    img_svd_restored = svd.inverse_transform(img_svd)
    img_svd_restored = img_svd_restored.reshape(img_enhanced.shape)

    # Save the preprocessed image
    output_path = os.path.join(output_folder, os.path.basename(image_path))
    cv2.imwrite(output_path, (img_svd_restored * 255).astype(np.uint8))
```

Figure A3

Splitting processed data to train, test, and validation datasets

```
DATA_DIR = '/content/drive/Shareddrives/DATA 270 - GWAR/dataset/processed_data'

os.mkdir('modeling')
classes = os.listdir(DATA_DIR)
train_dir = os.path.join('modeling', 'train')
val_dir = os.path.join('modeling', 'val')
test_dir = os.path.join('modeling', 'test')

if not os.path.exists(train_dir):
    os.mkdir(train_dir)
if not os.path.exists(val_dir):
    os.mkdir(val_dir)
if not os.path.exists(test_dir):
    os.mkdir(test_dir)

for class_name in classes:
    class_dir = os.path.join(DATA_DIR, class_name)
    images = os.listdir(class_dir)
    random.shuffle(images)

    # Split into training, validation and testing
    train_split = int(0.7 * len(images))
    test_split = int(0.15 * len(images))
    train_images = images[:train_split]
    val_images = images[train_split:train_split+test_split]
    test_images = images[train_split+test_split:]

    # Create class directories and copy images
    os.mkdir(os.path.join(train_dir, class_name))
    os.mkdir(os.path.join(val_dir, class_name))
    os.mkdir(os.path.join(test_dir, class_name))
    for img in train_images:
        shutil.copyfile(os.path.join(class_dir, img), os.path.join(train_dir, class_name, img))
    for img in val_images:
        shutil.copyfile(os.path.join(class_dir, img), os.path.join(val_dir, class_name, img))
    for img in test_images:
        shutil.copyfile(os.path.join(class_dir, img), os.path.join(test_dir, class_name, img))
```

Appendix B

Modeling

Figure B1

Importing required packages

```
import os
import shutil
import random
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
```

Figure B2

Loading train, test and validation datasets

```
# Define hyper-parameters
img_size = 224
batch_size = 32

train_dir = "/content/drive/Shareddrives/DATA 270 - GWAR/dataset/processed_data/modeling/train"
val_dir = "/content/drive/Shareddrives/DATA 270 - GWAR/dataset/processed_data/modeling/val"
test_dir = "/content/drive/Shareddrives/DATA 270 - GWAR/dataset/processed_data/modeling/test"

# Create ImageDataGenerators for train, validation, and test sets
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

Figure B3

Building model architecture appropriate parameters

```
# Defining the custom CNN architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dense(37, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Callbacks for model checkpoints and early stopping
checkpoint_path = "/content/cnn_model_checkpoint.h5"
checkpoint = ModelCheckpoint(checkpoint_path, save_best_only=True, verbose=1)
```

Figure B4

Training the CNN model

```
import time
train_start = time.time()

# Train the model
history = model.fit(train_data, validation_data=val_data, epochs=30, callbacks=[checkpoint])

train_end = time.time()
training_time = train_end - train_start

print(f"\nTraining Time: {training_time:.2f} seconds")
```

Figure B5

Plotting accuracy and loss curves for training and validation data

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.tight_layout()
plt.show()
```

Figure B6

Evaluating accuracy on test dataset

```
# Evaluate on test data

test_loss, test_accuracy = model.evaluate(test_data)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
41/41 [==============================] - 48s 154ms/step - loss: 0.2796 - accuracy: 0.9489
Test Loss: 0.2796
Test Accuracy: 0.9489
```

Figure B7

Generating classification report

```
import numpy as np
from sklearn.metrics import classification_report

# Generate predictions
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Get true labels
true_classes = test_generator.classes

# Get class indices and labels
class_indices = test_generator.class_indices
class_labels = list(class_indices.keys())

# Generate classification report
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print('Classification Report')
print(report)
```

Figure B8

Generating confusion matrix

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(true_classes, predicted_classes)
print('\nConfusion matrix')

plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True)
plt.xticks(ticks=np.arange(37), labels=labels, rotation=90)
plt.yticks(ticks=np.arange(37), labels=labels, rotation=0)

plt.show()
```

Figure B9

Importing packages for MobileNet model

```
import os
import shutil
import random
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
drive.mount('/content/drive')
```

Figure B10

Split data to training, val and test data

```
DATA_DIR = '/content/drive/Shareddrives/DATA 270 - GWAR/dataset/processed_data'

os.mkdir('mobilenet-modeling')
classes = os.listdir(DATA_DIR)
train_dir = os.path.join('mobilenet-modeling', 'train')
val_dir = os.path.join('mobilenet-modeling', 'val')
test_dir = os.path.join('mobilenet-modeling', 'test')

if not os.path.exists(train_dir):
    os.mkdir(train_dir)
if not os.path.exists(val_dir):
    os.mkdir(val_dir)
if not os.path.exists(test_dir):
    os.mkdir(test_dir)

for class_name in classes:
    class_dir = os.path.join(DATA_DIR, class_name)
    images = os.listdir(class_dir)
    random.shuffle(images)

    # Split into training, validation and testing
    train_split = int(0.7 * len(images))
    test_split = int(0.15 * len(images))
    train_images = images[:train_split]
    val_images = images[train_split:train_split+test_split]
    test_images = images[train_split+test_split:]

    # Create class directories and copy images
    os.mkdir(os.path.join(train_dir, class_name))
    os.mkdir(os.path.join(val_dir, class_name))
    os.mkdir(os.path.join(test_dir, class_name))
    for img in train_images:
        shutil.copyfile(os.path.join(class_dir, img), os.path.join(train_dir, class_name, img))
    for img in val_images:
        shutil.copyfile(os.path.join(class_dir, img), os.path.join(val_dir, class_name, img))
    for img in test_images:
        shutil.copyfile(os.path.join(class_dir, img), os.path.join(test_dir, class_name, img))
```

Figure B11

Load datasets as generator objects

```
# Define hyper-parameters
img_size = 224
batch_size = 32

# Create ImageDataGenerators for train, validation, and test sets
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

Figure B12

Building and training MobileNet model

```
# Load MobileNet model without including the top classification layers
base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))

# Add custom classification layers on top of MobileNet
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)

# Combine base model with custom classification layers
model = Model(inputs=base_model.input, outputs=predictions)

# Unfreeze the last 10 layers to adapt to fish images
for layer in base_model.layers[-15:]:
    layer.trainable = True

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
num_train = train_generator.samples
num_val = val_generator.samples
epochs = 30

history = model.fit(
    train_generator,
    steps_per_epoch=num_train // batch_size,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=num_val // batch_size
)
```

Figure B12

Plot accuracy and loss curves

```
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

Figure B14

Evaluation metrics generated on test data

```
import seaborn as sns
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_generator)
print(f'Test accuracy: {test_acc}')
print(f'Test Loss: {test_loss}')

# Generate predictions
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)

# Get true labels
true_classes = test_generator.classes

# Get class indices and labels
class_indices = test_generator.class_indices
class_labels = list(class_indices.keys())

# Generate classification report
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print('Classification Report')
print(report)
labels = [k for k in train_generator.class_indices.keys()]
cm = confusion_matrix(true_classes, predicted_classes)
print('\nConfusion matrix')
plt.figure(figsize=(15, 15))
sns.heatmap(cm, annot=True)
plt.xticks(ticks=np.arange(37), labels=labels, rotation=90)
plt.yticks(ticks=np.arange(37), labels=labels, rotation=0)
plt.show()
```

Figure B15

Importing packages for Vision Transformers

```
from transformers import ViTForImageClassification
from transformers import ViTImageProcessor
from transformers import Trainer
import torch
import numpy as np
from datasets import load_metric
from transformers import TrainingArguments
```

Figure B16

Training and testing ViT model

```
model_name_or_path = 'google/vit-base-patch16-224-in21k'
processor = ViTImageProcessor.from_pretrained(model_name_or_path)
ds = load_dataset('/content/drive/Shareddrives/project/Vit/split_data')

model = ViTForImageClassification.from_pretrained(model_name_or_path, num_labels=len(labels),
    id2label={str(i): c for i, c in enumerate(labels)}, label2id={c: str(i) for i, c in enumerate(labels)}
)

training_args = TrainingArguments(output_dir="./vit-base-beans", per_device_train_batch_size=32, evaluation_strategy="steps",
    num_train_epochs=8, fp16=True, save_steps=100, eval_steps=100, logging_steps=10, learning_rate=2e-4, save_total_limit=2,
    remove_unused_columns=False, push_to_hub=False, report_to='tensorboard', load_best_model_at_end=True, logging_dir="./logs",
)

trainer = Trainer(model=model, args=training_args, data_collator=collate_fn, compute_metrics=compute_metrics,
    train_dataset=prepared_ds["train"], eval_dataset=prepared_ds["validation"], tokenizer=processor,
)

train_results = trainer.train()
trainer.save_model()
trainer.log_metrics("train", train_results.metrics)
trainer.save_metrics("train", train_results.metrics)
trainer.save_state()

metrics = trainer.evaluate(prepared_ds['test'])
trainer.log_metrics("eval", metrics)
trainer.save_metrics("eval", metrics)
```

Figure B17

Building InceptionResNet model

```
[ ]  from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2
     from tensorflow.keras import layers, models

     # Loading the pre-trained Inception ResNet V2 model
     base_model = InceptionResNetV2(include_top=False, weights='imagenet', input_shape=(299, 299, 3))
     base_model.trainable = False  # Freeze the base model
```

```
# Defining the number of classes
num_classes = 37  # Update this with the number of your classes

# Adding custom layers on top of the base model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.Dense(num_classes, activation='softmax')  # Classification layer
])
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure B18

Evaluation of InseptionResNet model on test data

```
[ ]  test_dataset = datagen.flow_from_directory(test_dir, target_size=(299, 299), class_mode='categorical')

     testing_loss, testing_accuracy = model.evaluate(test_dataset)
     print(f"Test Dataset accuracy: {testing_accuracy * 100:.2f}%")
```

Appendix C

Evaluation Results

Figure C1

Confusion Matrix for CNN Model

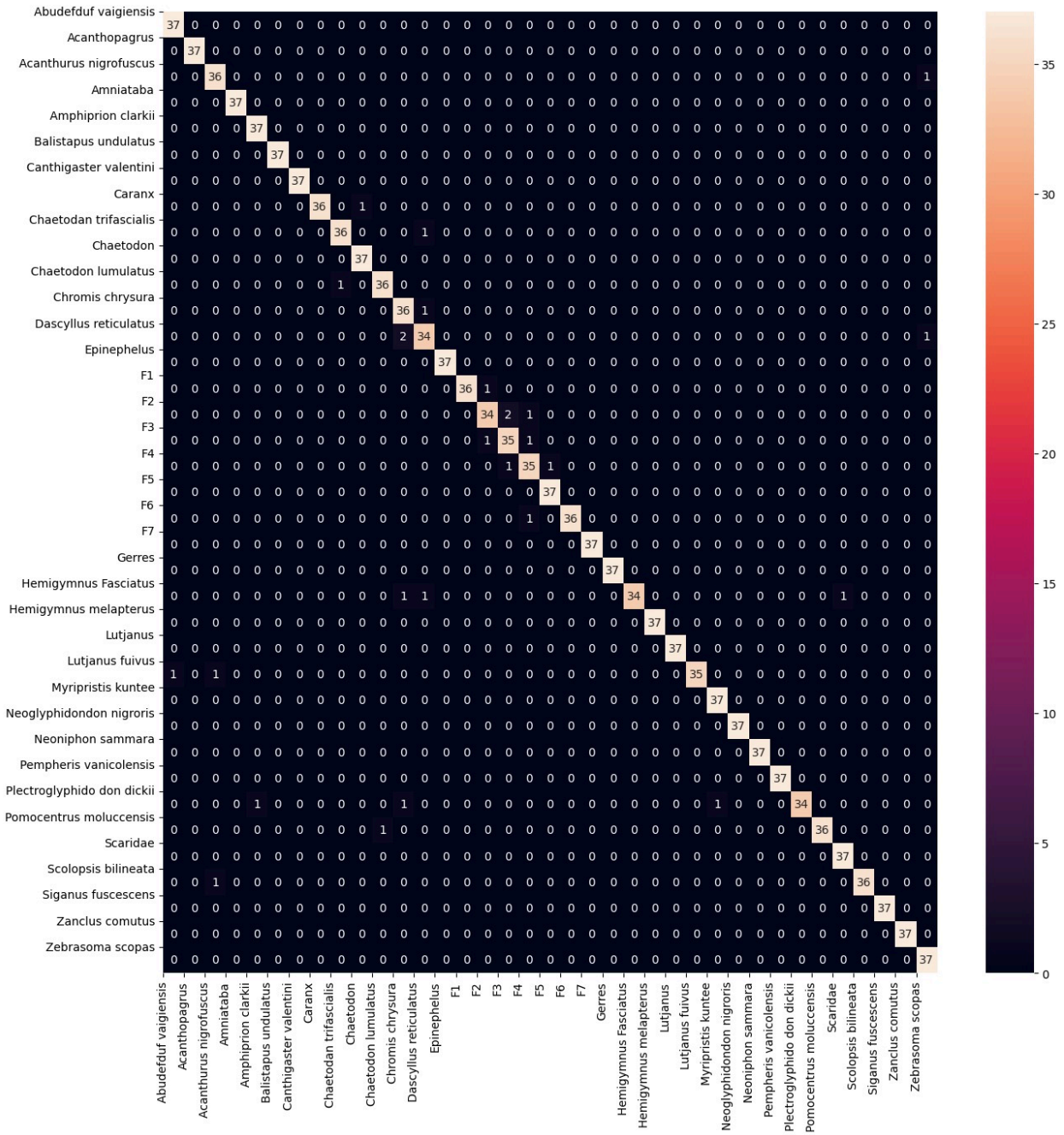


Figure C2

Confusion Matrix for MobileNet Model

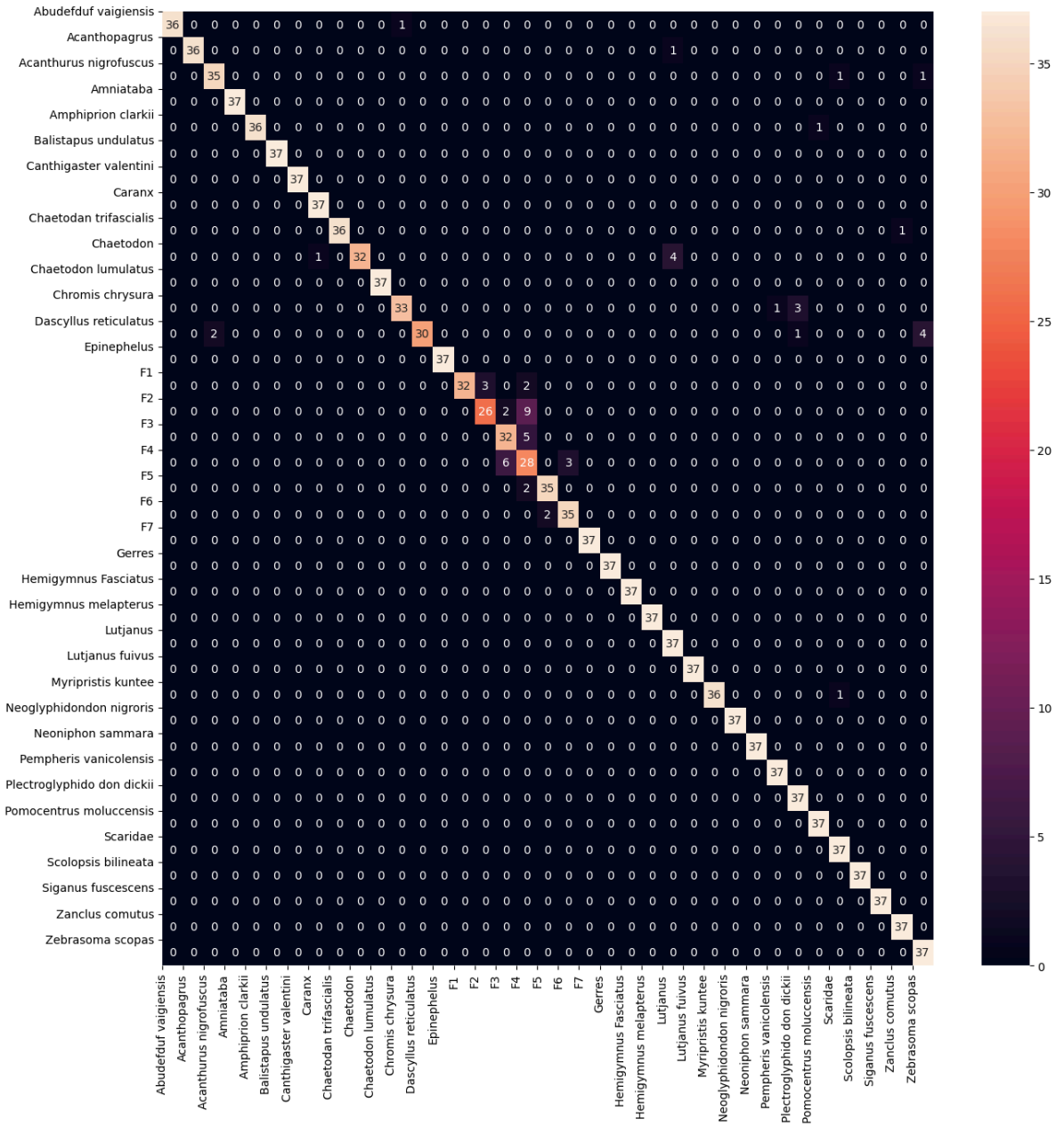


Figure C3

Confusion Matrix for InceptionResNet Model

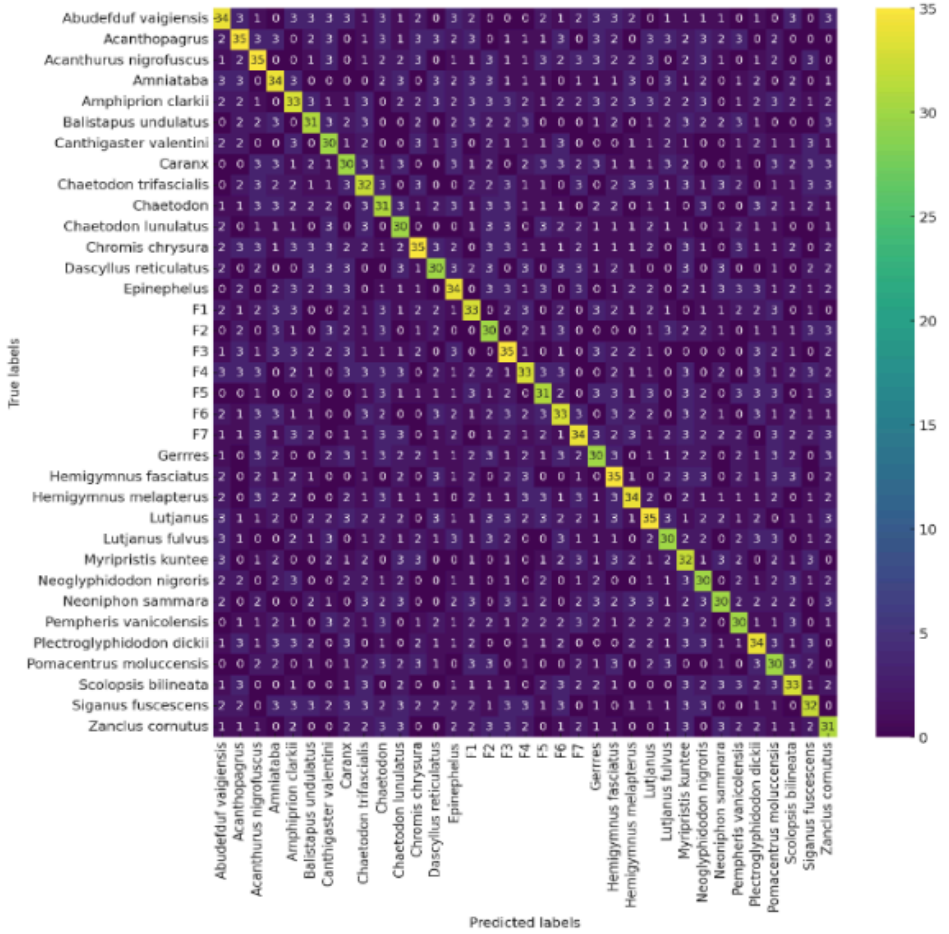
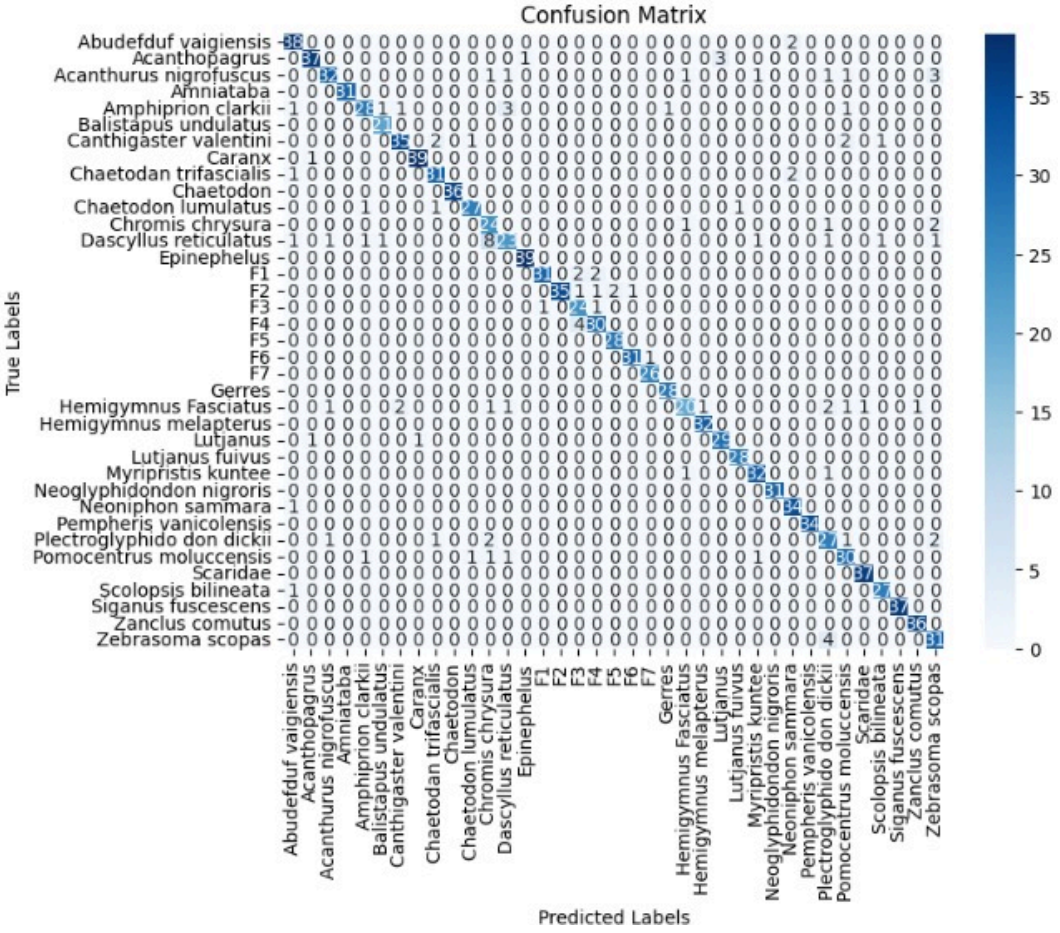


Figure C4

Confusion Matrix for Vision Transformers Model


Confusion Matrix
True Labels
Predicted Labels
Abudefduf vaigiensis
Acanthopagrus
Acanthurus nigrofuscus
Amniataba
Amphiprion clarkii
Balistapus undulatus
Canthigaster valentini
Caranx
Chaetodan trifascialis
Chaetodon
Chaetodon lumulatus
Chromis chrysura
Dascyllus reticulatus
Epinephelus
F1
F2
F3
F4
F5
F6
F7
Gerres
Hemigymnus Fasciatus
Hemigymnus melapterus
Lutjanus
Lutjanus fuivus
Myripristis kuntee
Neoglyphidondon nigroris
Neoniphon sammara
Pempheris vanicolensis
Plectroglyphido don dickii
Pomocentrus moluccensis
Scaridae
Scolopsis bilineata
Siganus fuscescens
Zanclus comutus
Zebrasoma scopas
35
30
25
20
15
10
5
0