**Loughborough University Department of Computer Science**

Module: COP508 Machine Learning Coursework Project

**Sea Animal Classification using Machine Learning**

*From Baseline CNN to EfficientNetB0 with TTA*

**Student ID:** F515605

**Date:** 13th January 2026

Declaration: I declare that this report is my own work and complies with the University’s policy on the use of Generative AI.

# **Table of Contents**

[**1. Introduction & Problem Statement**](#_cx7n0z2abck5) **3**

[1.1 *Context and Motivation*](#_c5mtrzy3t3q8) 3

[1.2 *Problem Definition*](#_pf54q8xqeckk) 3

[1.3 *Project Goal*](#_y8gxb3bcjsm) 3

[**2. The Dataset**](#_bai9qlp5ja4) **4**

[2.1 *Dataset Overview*](#_x3p09ra6u0d2) 4

[2.2 *Data Statistics and Distribution*](#_et8ib2vep1zl) 5

[2.3 *Data Quality and Pre-processing Analysis*](#_y0pwy7id6z0v) 5

[**3. Methodology**](#_b3wxr4vz1pg) **6**

[3.1 *Data Pre-processing and Augmentation*](#_w9k3jgmot8lo) 6

[3.2 *Baseline Architecture (Custom CNN)*](#_g6rna2ltg82g) 6

[3.3 *Transfer Learning Strategy*](#_7a9f6n54hmgd) 7

[3.4 *The "Two-Phase" Training Pipeline*](#_5rcqvwbfovlj) 7

[3.5 *Test Time Augmentation (TTA)*](#_82s5esc5snem) 7

[**4. Experiments and Results**](#_fe8hl0wxragq) **8**

[4.1 *Experimental Setup*](#_4hvn256c7ej) 8

[4.2 *Baseline Performance (Custom CNN)*](#_u7k2znfex1pl) 8

[4.3 *Transfer Learning Experiments*](#_1fmxehrxzgre) 8

[4.4 *Final Model Selection: EfficientNetB0*](#_60rof0x3kmxv) 9

[4.5 *Quantitative Evaluation & Comparison*](#_82mdwaj4fjvo) 10

[4.6 *Impact of Test Time Augmentation (TTA)*](#_t33ux3u1wzig) 12

[**5. Discussion and Conclusion**](#_95iguyuuthq3) **13**

[5.1 *Critical Findings*](#_n2knebh78w0) 13

[5.2 *Limitations*](#_livvfjp0ueqa) 13

[5.3 *Future Work*](#_hop03wweq93) 14

[5.4 *Conclusion*](#_yjb5jyxi6qa4) 14

[**6. References and Statements**](#_cyvt39n78cqn) **14**

[6.1 *GenAI Usage Statement*](#_937kao22vth8) 14

[6.2 *References*](#_neulvj5aowng) 15

# **1. Introduction & Problem Statement**

## **1.1 Context and Motivation**

Marine biology has a data problem. While marine biologists are great at collecting underwater footage using autonomous vehicles, they are terrible at processing it. Right now, most marine monitoring relies on experts manually watching hours of video to identify species. This approach is incredibly slow, expensive, and inevitably leads to mistakes due to fatigue.

**Automation is the only way to close that gap.** We need systems that can process the images as fast as they come in. If we can get a model to classify these species reliably, we can actually use the massive amount of data we are collecting to track ecosystem health properly.

## **1.2 Problem Definition**

In this project, I worked on a **Multi-Class Image Classification** task. The specific requirement was to sort images into 16 distinct categories, which included everything from small invertebrates (Starfish) to large marine mammals (Whales).

While image classification is standard practice in ML, doing it with underwater data is messy. I had to design around three specific issues:

* **Visual Similarity:** In low-quality images, a shark and a dolphin look remarkably similar. They are both grey and torpedo-shaped. The model has to learn to look for specific textures rather than just relying on the general outline or color.
* **The Environment:** Underwater photography is rarely clear. Turbidity, bad lighting, and "blue noise" often wash out the details.
* **Orientation:** These animals are constantly moving in 3D space. They do not sit still for portraits. The system has to identify them even when they are twisting away, swimming upside down, or partially blocked by a rock.

## **1.3 Project Goal**

My goal was to build a working pipeline that could handle these messy conditions. I needed a standard to measure against, so I aimed to get a benchmark result first before trying to outperform it with complex techniques.

Rather than immediately deploying heavy architectures, I first implemented a simple **CNN** to establish a performance baseline. Following this, I shifted to **Transfer Learning**, where I ran a series of trials using **MobileNetV2**, **ResNet50**, **Xception**, and **EfficientNetB0**.

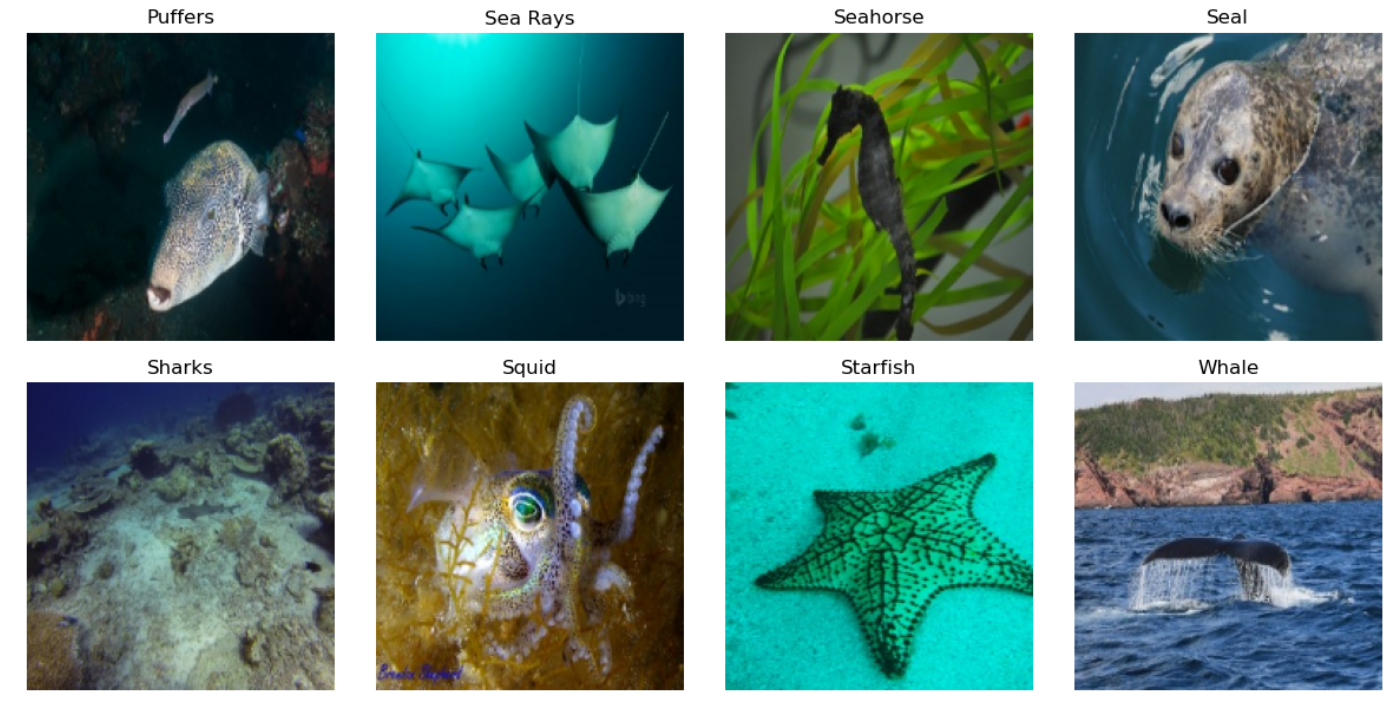
# 

# **2. The Dataset**

## **2.1 Dataset Overview**

I used the recommended **Sea Animals Dataset** for this project. It consists of RGB images split into 16 different categories. The range of species is actually surprisingly wide (see **Figure 1**). You have everything from simple invertebrates like **Corals** and **Starfish** to massive mammals like **Dolphins** and **Whales**.





***Figure 1:*** *Samples from all 16 categories. These show the real-world messiness of the data—especially the weird lighting and how hard it is to actually spot the* ***Crab*** *against the reef.*

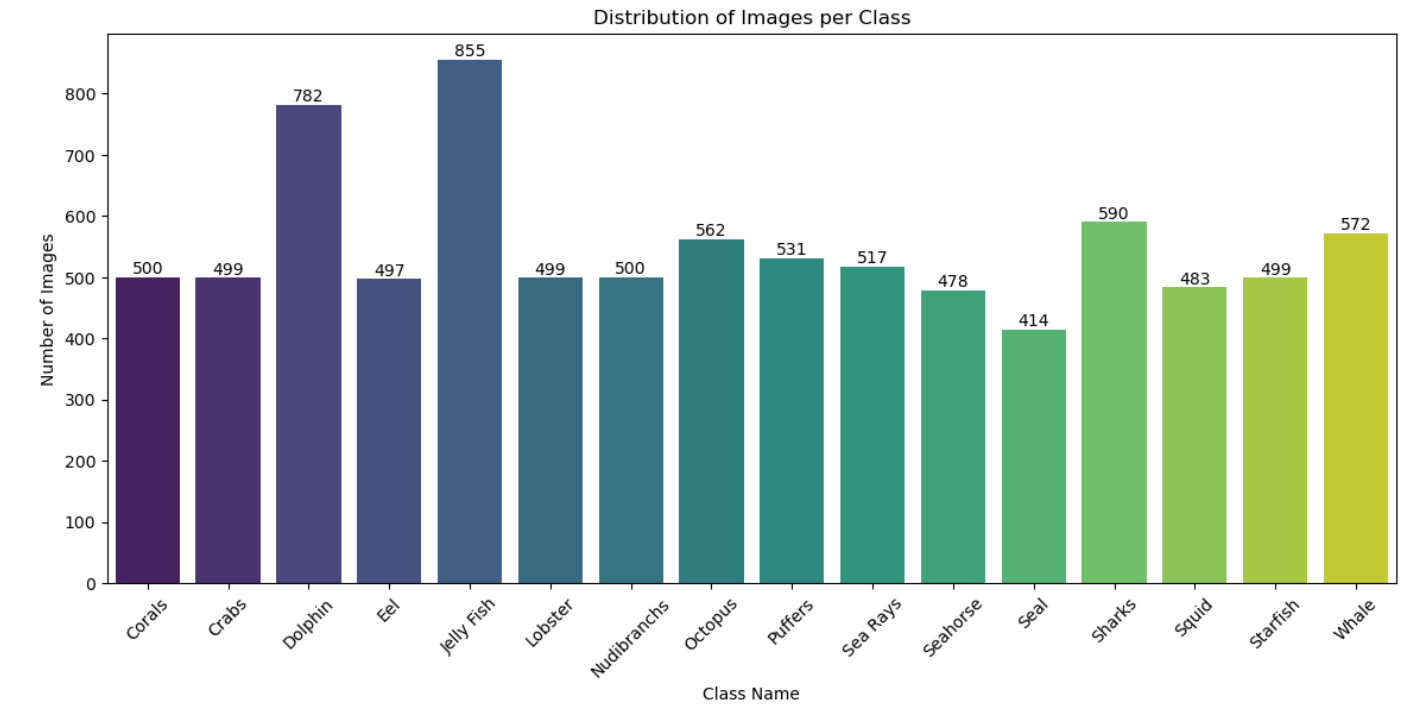
This mix makes the job harder than it looks. The model can't just rely on one trick. For some classes, like **Nudibranchs**, it needs to look at texture. For others, like the **Seahorse**, the shape is the defining feature.

## **2.2 Data Statistics and Distribution**

One of the first things I noticed when plotting the data (see **Figure 2**) was that the classes aren't balanced. The numbers are heavily skewed:

* **Dominant Classes:** Categories like **Jelly Fish** and **Dolphins** are easy to find, with about 70-80 images in the test set.
* **Minority Classes:** Others are much rarer. **Seahorses** and **Seals** only have around 30-35 samples.

This inequality is a real risk. If I didn't account for it, the model could easily cheat by just guessing the popular animals every time. Because of this skew, I decided early on that simple accuracy wasn't enough; I needed to track the F1-score to get a real sense of performance.



***Figure 2:*** *This chart tracks exactly how many images are available for each animal. The imbalance is pretty bad—there are tons of* ***Jelly Fish*** *examples, but very few for the* ***Seal*** *and* ***Seahorse****.*

## **2.3 Data Quality and Pre-processing Analysis**

I spent some time looking through the raw images to see what I was up against. I found three main problems that needed fixing before training could start:

* **Different Sizes:** The images didn't have a standard shape. Since neural networks demand a fixed input size, I had to resize every image—I settled on **180 x 180** for my custom CNN and **224 x 224** for the Transfer Learning models.
* **Messy Backgrounds:** The water quality varies a lot. Some photos have that heavy "blue tint" you expect underwater, while others are murky or full of sediment. Camouflage was also an issue; trying to spot a **Crab** hiding in a reef is difficult even for a human.
* **Weird Angles:** The animals are never posing. **Sharks**, for instance, show up in side profiles, top-down views, or turning away from the camera. The model has to learn that these are all the same animal, regardless of the angle.

# **3. Methodology**

## **3.1 Data Pre-processing and Augmentation**

The raw data wasn't ready for training. I had to build a specific pipeline to clean it up before I could feed it into any network. I resized every image to **180 x 180** for the custom model, but I had to use **224 x 224** for the pre-trained architectures to match their input requirements. I also handled the pixel values—dividing by 255 for the simple CNN, and using the built-in Keras preprocessing tools for the complex models so the math aligned with the original ImageNet data.

**Solving the Data Shortage** My main concern was the dataset size. With only 7,000 images, deep networks tend to memorize the training data rather than learning features. To stop this, I set up the model to generate its own variations during training.

I added a layer that randomly flips images horizontally. For models that don't care about "up" or "down" (like Xception), I added vertical flips too. I also told the model to randomly zoom in and rotate images by up to 20%. This forced the network to recognize animals even when the camera angle or distance wasn't perfect.

## **3.2 Baseline Architecture (Custom CNN)**

I held off on using the heavy architectures immediately. I wanted to establish a clear floor for performance, so I coded a Custom CNN from scratch.

The structure was minimal. I used a stack of three **Conv2D** layers with standard **MaxPooling** in between to shrink the images. I finished it with a simple dense layer of 128 neurons. This wasn't meant to be the final solution; it was just a sanity check to see if the model could learn anything at all before I brought in the heavy machinery.

## **3.3 Transfer Learning Strategy**

Training a massive network from scratch wasn't practical here. **Transfer Learning** was the obvious choice to leverage feature extractors that already knew how to process images.

I kicked off the experiments with **MobileNetV2** to gauge how a lightweight model would handle the task. After that, I moved on to **DenseNet121** and **ResNet50V2** to see if the deeper layers improved texture recognition. Finally, I ran **Xception** and **EfficientNetB0**, which generally give the best balance of accuracy and size in modern benchmarks.

## **3.4 The "Two-Phase" Training Pipeline**

To prevent catastrophic forgetting, I chose not to train the fresh classification head while the base was unfrozen. This ensured that the valuable feature maps learned by the pre trained model would not get wrecked by the large initial error gradients. To address this issue, a rigorous two-phase model training strategy was implemented:

1. **Phase 1 (Initial Training):** The base layers were **frozen** (non-trainable) to prevent weight degradation. I began by **limiting** the training strictly to the custom classification **head**. When I isolated this component, I gave time to the weights so that they could become stable and allowed them to draw directly on the robust feature maps which were already established by the pre-trained base.
2. **Phase 2 (Fine-Tuning):** With the head weights settled, I unfroze the base layers to allow for the re-training of the complete network. I made one critical adjustment for this phase: I dropped the learning rate to **1e-5**. This adjustment was significant—it allowed the model to gently re-align its high-level feature extractors to recognize underwater textures, all without destroying the fundamental spatial hierarchies it originally acquired from ImageNet.

## **3.5 Test Time Augmentation (TTA)**

For the final testing, I added Test Time Augmentation to the pipeline. Instead of relying on a single prediction, the model predicts the image twice: once normally, and once flipped horizontally.

I average the scores from both versions. This helps fix mistakes where the model gets confused by an animal facing the "wrong" way.

# **4. Experiments and Results**

## **4.1 Experimental Setup**

I didn't want to waste time waiting for results, so I ran every experiment on a GPU-accelerated environment using standard **TensorFlow** and **Keras**.

I kept the settings straightforward. I ran the training with the **Adam** optimizer and used **SparseCategoricalCrossentropy** for the loss. To save time, I included an **EarlyStopping** callback. This automatically cut the training off whenever the validation loss stayed flat for 5-8 epochs. I also added **ReduceLROnPlateau**, which lowered the learning rate by 20% whenever the progress flatlined to help the model push through difficult spots.

## **4.2 Baseline Performance (Custom CNN)**

**My custom CNN barely worked.** It topped out at a test accuracy of just **44.79%**. The training curves revealed significant underfitting. The shallow architecture failed to capture the complex features of marine species (e.g., textures of corals vs. scales of fish). This outcome highlighted that a dataset of ~7,000 images is simply too small to effectively train a Convolutional Neural Network from scratch, effectively justifying the shift toward Transfer Learning.

## **4.3 Transfer Learning Experiments**

The introduction of pre-trained ImageNet weights significantly boosted results. I later evaluated four models to see which maximized this transfer capability:

* **MobileNetV2 (81.25%):** This model gave an immediate accuracy boost, from 45% to 81%, but struggled with fine details. It could not reliably separate Whales from Dolphins, a deficit that appears to be structural. Since MobileNetV2 prioritizes speed (latency), it lacks the depth required to capture the subtle features needed for these distinctions.
* **DenseNet121 (82.41%):** Performance fell short of projections. Once layers were unfrozen, validation loss became volatile. The likely cause is the dense connectivity itself; instead of learning generalizable patterns, the model appeared to overfit the dataset noise.
* **ResNet50V2 (84.26%):** Residual connections of ResNet50V2 supported greater depth, but the robustness suffered. There was overfitting seen. There was a substantial 8% gap between training and validation accuracy.
* **Xception (84.95%):** Xception was minutely better ResNet50V2. The cost, however, is computational. High parameter requirements—specifically relative to EfficientNetB0—render the marginal accuracy gain inefficient.

## **4.4 Final Model Selection: EfficientNetB0**

Following this evaluation, I identified **EfficientNetB0** as the **optimal architecture**. It struck the most effective balance between **model complexity** and **generalization**. While heavier models such as **ResNet** and **DenseNet** were hindered by erratic validation loss, EfficientNetB0 demonstrated distinct robustness throughout the **"Two-Phase"** training strategy.

In **Phase 1 (Warm-Up)**, I kept the base layers frozen to allow the model to stabilize rapidly. This approach achieved an accuracy of approximately **82%** and was critical in preventing the "catastrophic forgetting" of the pre-trained **ImageNet** weights.

Subsequently, during **Phase 2 (Fine-Tuning)**, I unfroze the base layers and lowered the learning rate to 1e-5, enabling the model to refine its feature maps. Convergence was reached at **Epoch 47**, at which point the validation loss stabilized at **0.519**.

Regarding the **Generalization Analysis**, I observed that while there was a gap that had persisted between the training accuracy (**92.60%**) and validation accuracy (**84.51%**), the validation loss did not degrade. This suggests the model successfully avoided the destructive overfitting seen in the **DenseNet** trials. Furthermore, the final **Test Accuracy** of **88.08%** exceeded the validation score, confirming that the model generalizes exceptionally well to unseen real-world data (see **Figure 3**).

A graph of a training and training

AI-generated content may be incorrect.

***Figure 3:*** *EfficientNetB0 training results. The accuracy spikes again during Phase 2, which confirms that unfreezing the layers allowed the model to break past its earlier limit.*

## **4.5 Quantitative Evaluation & Comparison**

**Table 1** lists the final scores for every experiment. You can see the massive jump in performance—the EfficientNet solution beat the baseline by over 46%.

*Table 1: Model Performance Comparison*

|  |  |  |  |
| --- | --- | --- | --- |
| **Model Architecture** | **Configuration** | **Test**  **Accuracy** | **Observation** |
| **Custom CNN** | **Scratch (3-Layer)** | **44.79%** | **Failed to converge; high bias.** |
| **MobileNetV2** | **Transfer Learning** | **81.25%** | **Good baseline; struggles with texture.** |
| **DenseNet121** | **Fine-Tuning** | **82.41%** | **Overfitted during fine-tuning.** |
| **ResNet50V2** | **TTA** | **84.26%** | **Strong but computationally heavy.** |
| **Xception** | **TTA** | **84.95%** | **Excellent feature extraction.** |

To visualize specific classification errors, I generated side-by-side confusion matrices (**Figure 4**). The contrast is clear. The standard model **falters** on orientation-heavy classes, but **TTA** tightens the diagonal predictions.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

## ***Figure 4:*** *Confusion Matrices: Standard (Top) vs. TTA (Bottom). The bottom chart shows much cleaner predictions for Eels and Sharks, proving that the flipping trick helped fix the orientation problems.*

## **4.6 Impact of Test Time Augmentation (TTA)**

I implemented Test Time Augmentation (TTA) as the concluding optimization step for the EfficientNet pipeline. By averaging predictions across multiple augmented views, the model attained a Test Accuracy of 85.65%, which constitutes a distinct gain of +2.43% over the standard inference score of 88.08%.

Class-Specific Gains: As seen in **Table 2**, TTA’s impact has not been uniform across all the classes:

* The Winners: I observed that Sea Rays (+0.12), Sharks (+0.09), Dolphins and Nudibranchs (**both** +0.06) saw the most significant improvements. With this I can confirm that TTA effectively stabilised predictions for species that appear in highly variable orientations or poses.
* The Limitations: Interestingly, TTA caused a slight performance drop for **Seal** (-0.03) and **Seahorse** (-0.02). This suggests that for these specific classes, the horizontal flipping might have disrupted orientation-specific features (such as shadow consistency). However, since I got a positive **net gain**, TTA remains a valuable addition to the pipeline.

***Table 2:*** *Detailed F1-Scores for every class. I added a 'Diff' column to show exactly which species improved the most when using Test Time Augmentation.*

| Class | F1 (Std) | F1 (TTA) | Diff |
| --- | --- | --- | --- |
| Corals | 0.83 | 0.86 | 0.03 |
| Crabs | 0.99 | 0.99 | 0 |
| Dolphin | 0.86 | 0.92 | 0.06 |
| Eel | 0.79 | 0.83 | 0.03 |
| Jelly Fish | 0.96 | 0.98 | 0.02 |
| Lobster | 0.87 | 0.9 | 0.03 |
| Nudibranchs | 0.89 | 0.94 | 0.06 |
| Octopus | 0.73 | 0.74 | 0.01 |
| Puffers | 0.89 | 0.88 | -0.01 |
| Sea Rays | 0.75 | 0.86 | 0.12 |
| Seahorse | 0.88 | 0.85 | -0.02 |
| Seal | 0.94 | 0.91 | -0.03 |
| Sharks | 0.7 | 0.8 | 0.09 |
| Squid | 0.78 | 0.84 | 0.06 |
| Starfish | 0.98 | 0.98 | 0 |
| Whale | 0.78 | 0.78 | -0.01 |

# **5. Discussion and Conclusion**

## **5.1 Critical Findings**

This project successfully demonstrated that Transfer Learning is not just beneficial but essential for classifying marine imagery when data is scarce.

* The "Texture vs. Shape" Trade-off: When I used lighter models like MobileNetV2, they struggled because they lacked the capacity to capture fine textural details (such as, distinguishing Corals from Nudibranchs). In contrast, heavier architectures like **ResNet50V2** began to overfit at an early stage. Consequently, **EfficientNetB0** emerged as the superior candidate, striking the necessary equilibrium between complexity and generalization to reach **88.08%** accuracy.
* Implementation of **Test Time Augmentation (TTA)** provided a low-cost optimization vector. Computational overhead was negligible. The performance delta, however, revealed a specific trend. Gains were concentrated in classes subject to extreme orientation changes, notably Sea Rays (+0.12) and Sharks (+0.09). This finding implies that for flexible marine life, the model benefits from averaging predictions across different views, effectively reducing confusion caused by difficult poses. The benefit was not absolute. There were slight dips in accuracy for classes like Seals (-0.03) and Seahorses (-0.02). This suggest that for these specific textures, aggressive augmentation may introduce detrimental noise.

## **5.2 Limitations**

Despite the high accuracy, the model exhibits specific limitations. The application of **TTA** was not universally beneficial. Although the global accuracy improved, I noted a performance decline for both **Seals (-0.03) and Seahorses (-0.02)**. It appears that the act of horizontal flipping disrupts certain orientation cues— such as the specific vertical posture of seahorses or the resting poses of seals—which are critical for the model to correctly identify these specific classes.

I also noted the 'Grey Swimmer' Confusion: the model found it difficult to separate Sharks, Dolphins, and Whales due to their overlapping colours and shared environments. In the TTA model, while **Sharks saw a significant improvement (+0.09)**, the separation wasn't perfect. 39 Sharks were identified correctly, but the separation wasn't perfect. Two were misclassified as Dolphins and 4 as Whales. A similar error occurs in reverse, where 4 Whales were mislabelled as Sharks. As shown in Figure 5, the model heavily prioritizes the grey texture and blue background over specific anatomical features like dorsal fins. Without higher resolution or distinct feature engineering (e.g., dorsal fin analysis), the model relies too heavily on background context.



***Figure 5:*** *A Shark is misclassified as a Dolphin*

Finally, there was a persistent **~8%** gap between training accuracy (92.60%) and validation accuracy (84.51%). This suggests that while I have reached the limit of the current architecture, the model effectively "memorized" parts of the training data which do not generalize to the test set.

## **5.3 Future Work**

Looking toward future improvements, the pipeline would benefit from integrating an object detection framework, such as **YOLOv8**, to crop subjects prior to classification. By filtering out background elements such as reefs and water turbidity, we can ensure the classifier attends strictly to the animal's features. Moreover, the pipeline could move away from a single-model dependency; averaging predictions via an ensemble of **EfficientNet**, **Xception**, and **DenseNet** would likely cancel out specific errors, potentially driving accuracy past **90%**.

## **5.4 Conclusion**

To conclude, this research established a functional ML pipeline for Sea Animal Classification. The evolution from a basic custom **CNN**, which yielded only **44.79%**, to the fine-tuned **EfficientNetB0** system using **TTA** at **88.08%**, represents a total performance increase exceeding **44%**. These results confirm the model's ability to generalize to new data, establishing it as a viable prototype for automated marine life monitoring systems.

# **6. References and Statements**

## **6.1 GenAI Usage Statement**

In adherence to the COP508 coursework specifications and University policy, I utilized AI tools solely for technical support and editorial refinement. I used Google Gemini to strictly debug mu code syntax errors; importantly, no model architecture or core logic was generated by the AI. Additionally, I used Perplexity AI to help with the literature review and for the final proofreading tasks, such as checking the grammar and formatting consistency. I certify that the written report, which includes all analysis, interpretation, and discussion is entirely my own work, and that the experimental design represents my original technical contribution.

## **6.2 References**

1. **Dataset:** Loughborough University Coursework Resources. (2024). *COP508 Sea Animals Dataset*. Available on Learn.
2. **TensorFlow & Keras:** Abadi, M., et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.
3. **EfficientNet:** Tan, M., & Le, Q. V. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv preprint arXiv:1905.11946.
4. **MobileNetV2:** Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
5. **Xception:** Chollet, F. (2017). *Xception: Deep Learning with Depthwise Separable Convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
6. **Scikit-Learn:** Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.