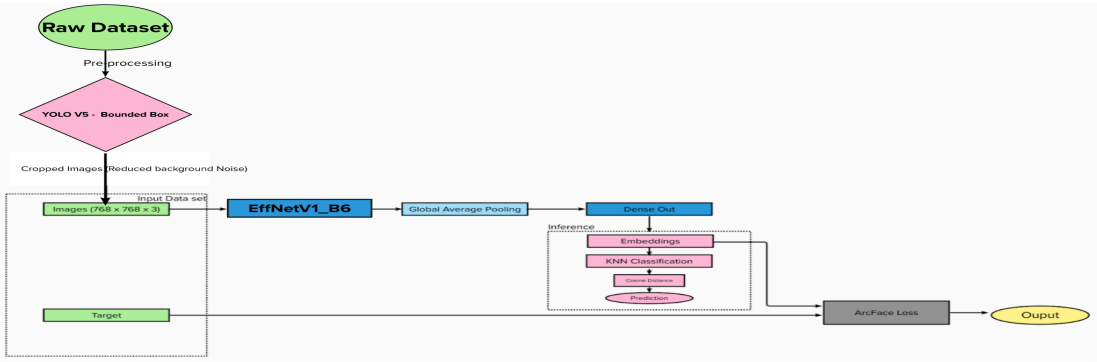
Whale and Dolphin ID: Identification of Whales and Dolphins by using their Unique Characteristics

Uday Kanth Reddy Kakarla
Department of Computer Science
University of Illinois Urbana-Champaign
uk3@illinois.edu

Girija Manoj Kumar Reddy Kalakoti
Department of Computer Science
University of Illinois Urbana-Champaign
gmk6@illinois.edu

Project Summary: Our project focuses on the critical issue of identifying individual whales and dolphins using their unique physical characteristics, an area of study that has significant implications for understanding and preserving marine life. We leveraged deep learning techniques, specifically convolutional neural networks (CNNs), to automate this process, replacing the manual identification traditionally performed by researchers. Using the YoloV5, EfficientNet, combined with the ArcFace loss function, we processed a substantial dataset of over 15,000 unique marine mammal images sourced from multiple research organizations. Our overall system, achieves a mean average precision (mAP) of 72% on the leaderboard. The results also include the potential identification of "new individuals" not previously seen in the training dataset. Our most interesting finding is the efficacy of this automated method, which promises to considerably reduce the time spent on image identification and enable a larger scale of study on marine animals.



Overview of our Approach

Results: Each image has 5 corresponding predictions given by our ensemble model.

Image id	Prediction-1	Prediction-2	Prediction-3	Prediction-4	Prediction-5
59bc2d264c9731.jpg	47a2f9918aa2	beec2e6997c8	91ed5caeb0d3	73c68d52e748	03b002f56368
5fb61ff7e07076.jpg	2e0b381d3467	39fdf5369ae2	5525e545ae81	32d3b30b473f	new_individual
0cfc88bad51a18.jpg	e4a55c745bd9	4f64aee1cda9	5393adf1a8ac	fc0f7c162cc0	d5b28257276e
3c52966f74d2ad.jpg	978520860ceb	d4dfc9c0b367	cfd34e95a962	c917b552fc31	91ed5caeb0d3

1. Introduction

The ability to track and identify individual animals is crucial for understanding the behavior and movements of wildlife populations. While fingerprints and facial recognition are commonly used for human identification, researchers have manually tracked marine life using the shape and markings on their tails, dorsal fins, heads, and other body parts. But this process of manual identification can be time-consuming and labor-intensive. To address this challenge , we explore the use of deep learning techniques, specifically convolutional neural networks (CNNs), to automate the photo-ID process for whale and dolphin identification. By leveraging CNNs to automate the identification of individual animals based on their unique markings and features, we aim to significantly reduce the image identification time and enable a scale of study that was previously unaffordable or impossible. This could lead to more efficient and effective research into the behaviors of marine mammals and other wildlife populations.

2. Background

We went through the existing literature on the identification of whales[2],[3],[4],[5], and came to know about the strategies that have worked out the best in recent times. Out of all the models, the EfficientNet developed at Google would be the best fit for this problem, as it considers depth, width, and resolution scaling, which is crucial for finding the images that are closest with respect to the unique identifiers. Once we finalized the model, we explored the most suitable loss functions for this problem. Currently, in the existing literature, people often use softmax loss, sphere face loss, and ArcFace loss. Among these loss functions, the ArcFace loss function will be more appropriate here as ArcFace loss outperforms the Softmax loss and provides a more clear geometric interpretation due to its exact correspondence to the geodesic distance on the hypersphere. Also exisiting face recognising techniques uses ArcFace loss, which further supported our intuition to select that in our problem of identifying whales on their unique features. Below is a brief overview of our approach.

3. Approach

a. **Image Preprocessing:** We crop and pseudo label(whether the marine animal in the image is whale or dolphin) the images to obtain portion of the image that is our interest i.e we remove the background water and other irrelevant information from the image. We do this by using Yolov5. We do this image preprocessing process for both our training and validation dataset.
b. **Effnet:** These cropped images along with the corresponding pseudo labels become the input for our Effnet model. When an image is passed through the Effnet model we get the image embeddings for that image.
c. **KNN classification:** After our training, KNN model contains the image embeddings of all the images in the training dataset. For Photo-Identification of an image we check the cosine distance of the image embedding of the image with the the other image embeddings and output the top 5 images that have the highest 1-cosineScore.
d. Here, we only check the cosine distance with image embeddings that are from the same category i.e during testing, the image is first passed through the Yolov5

model to get the cropped image along with the category(Whale or dolphin) of the image. Thus, if the image is a whale we check the cosine distance with only whale images rather than all the images.The below block diagram gives the visualization of our approach.

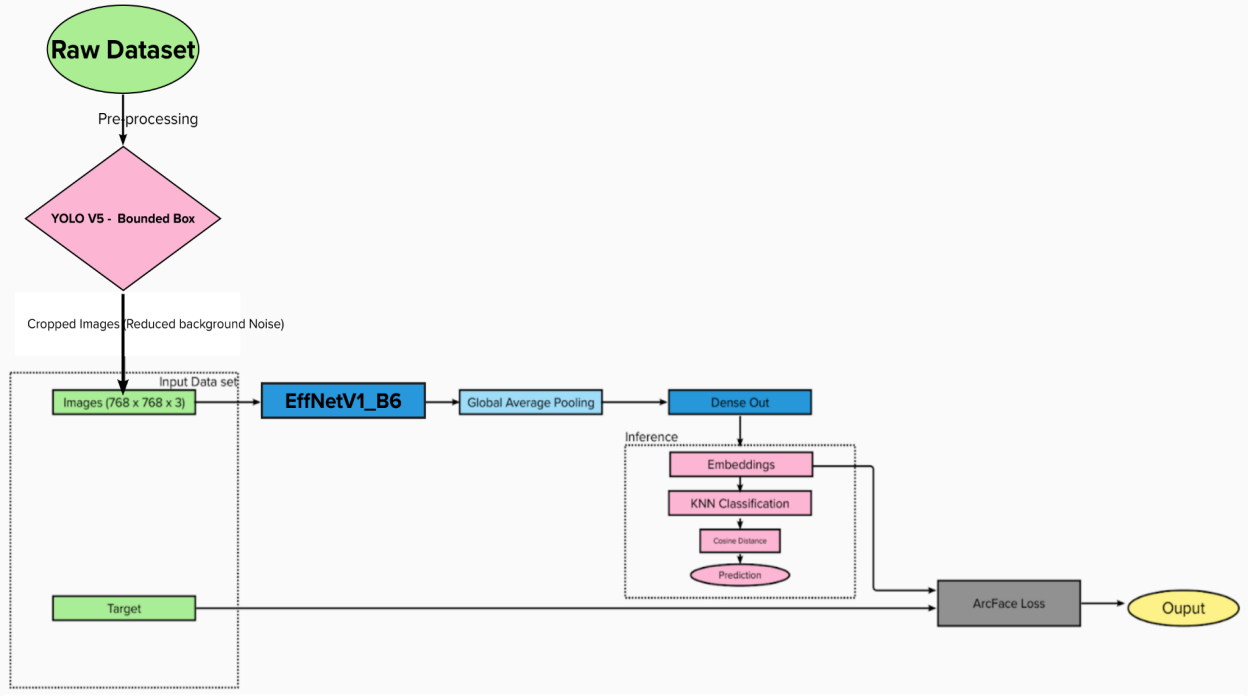


Figure 1: Block diagram of our approach.

4. Details of Our Approach

In this section we will delve deeper into our approach starting from our dataset to the final system we arrived at.

a. Dataset:

i. We have used this Kaggle data set [1] that involves images of over 15,000 unique individual marine mammals from 30 different species collected from 28 different research organizations. Below is a snapshot of a sample of images from the dataset. Also pasted below is a brief description of the dataset indicating the class frequency of whales and dolphins and the species type frequency of whales and dolphins.

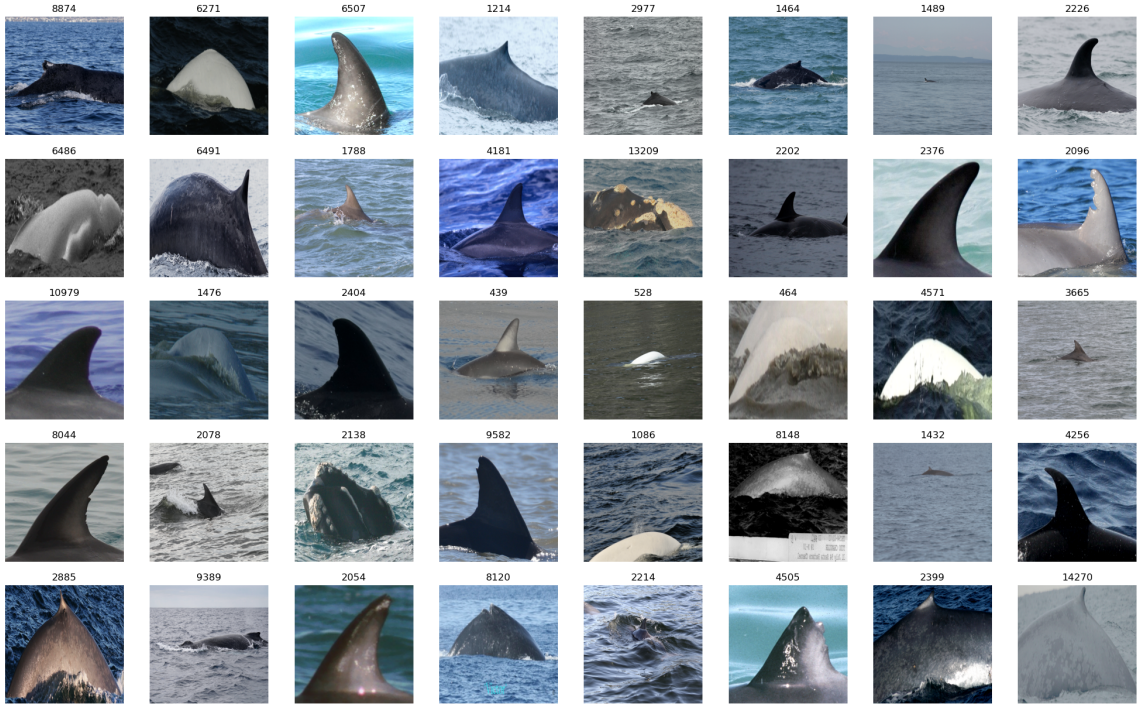


Figure 2: A sample set of images from the dataset.

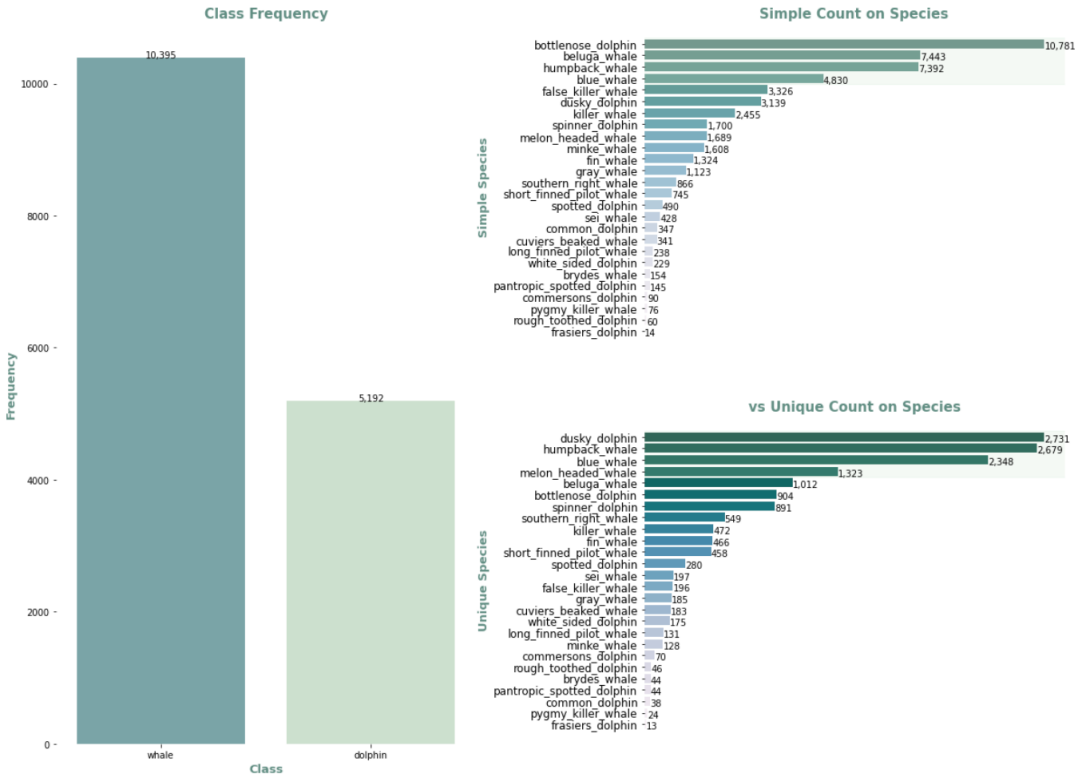


Figure 3: Brief Summary of Data

5. Image Preprocessing:

Initially, we tried using the images from the training dataset without any preprocessing in order to photo-ID the whales and dolphins. But this didnt work out well and resulted in low mAP scores. It seemed like the model wasn't able to accurately ID them due to noise or irrelevant background information in the images.

a. To improve the accuracy, we decided to crop the images to only the relevant part by using the **bounding box** technique from **YOLO V5**, which involves defining a rectangular region around the object of interest in an image. By cropping the image to the size of the bounding box, we were able to remove the unnecessary water background that could have interfered with assigning ID to the image. Also we trained our Yolo V5 to predict the category to which this particular image belongs to. This helped in creating pseudo labels such as whether the marine animal in the image is that of dolphin or whale. This information was further used when trying to identify the top5 images that are close to this image, such as discarding images that are of completely different class while calculating image embeddings for K-NN classification. Link to our Cropped dataset.

b. Using YOLO V5 and the bounded boxing technique helped us to improve the accuracy of the model, as we observed an increase in mAP scores. This approach allowed the model to focus on the relevant features while ignoring the irrelevant information in the images. You can see the before and after of processed images from the training dataset in the below Fig 3.

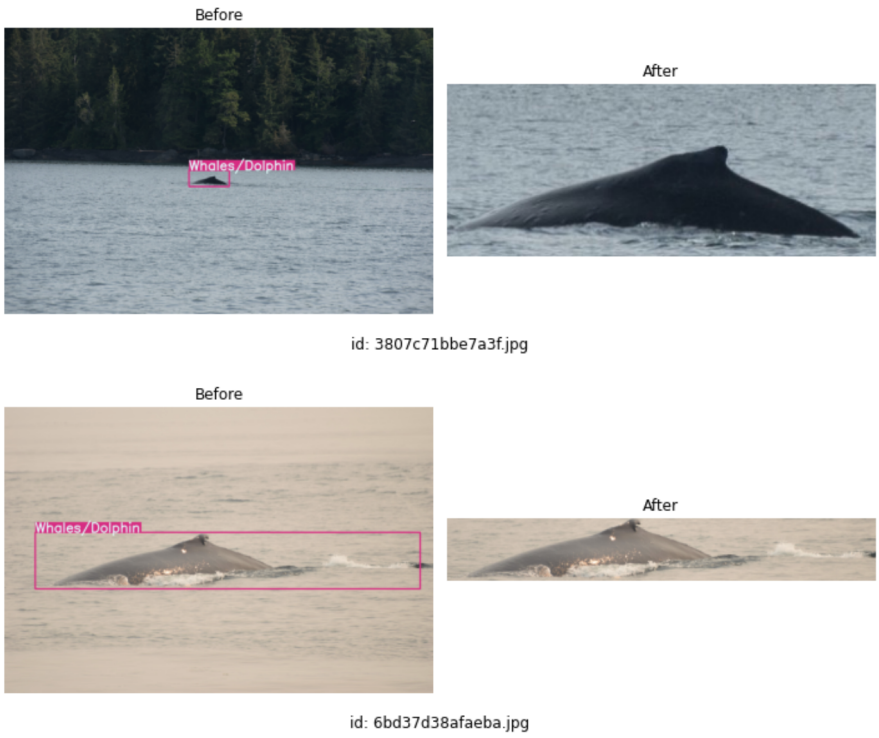


Figure 4: Before and After Preprocessing the images with Yolov5

6. Explanation of Design Choices and Key Terms:

The following section explains the key terms in our system. This gives a background idea regarding the components used and provides a foundation for understanding the implementation details of our approach.

a. Effnet as our backbone:

i. Effnet is a convolutional neural network architecture designed for efficient use of computational resources and optimized for image classification tasks. It uses a combination of convolutional layers with different kernel sizes and scaling factors to extract features from images at multiple scales, and then combines these features in a way that balances accuracy and efficiency.

ii. In the context of whale identification, the unique markings and characteristics of individual whales can be quite subtle and require high levels of accuracy and detail to detect. Effnet, with its ability to extract and combine features at multiple scales, may be better suited for this task than traditional CNNs or Vision Transformers.

iii. Additionally, Effnet can be trained with relatively small datasets and can perform well even with limited training data which is the case with our dataset where there are only few instances related to a particular animal. Hence we proceeded with EffNet.

We started with the EfficientNet-V1-B0 architecture and kept checking the performance as we increased the B value. We have reached the EfficientNet-B6 model, and we observed that as we kept increasing the value of B, the mAP kept increasing. With EfficientNet-B6 we are obtaining a 72% mAP on the public score. With higher variations of B like B6, we train and infer from the images of higher resolution.

b. Brief EfficientNet Version 1 B0 Architecture:

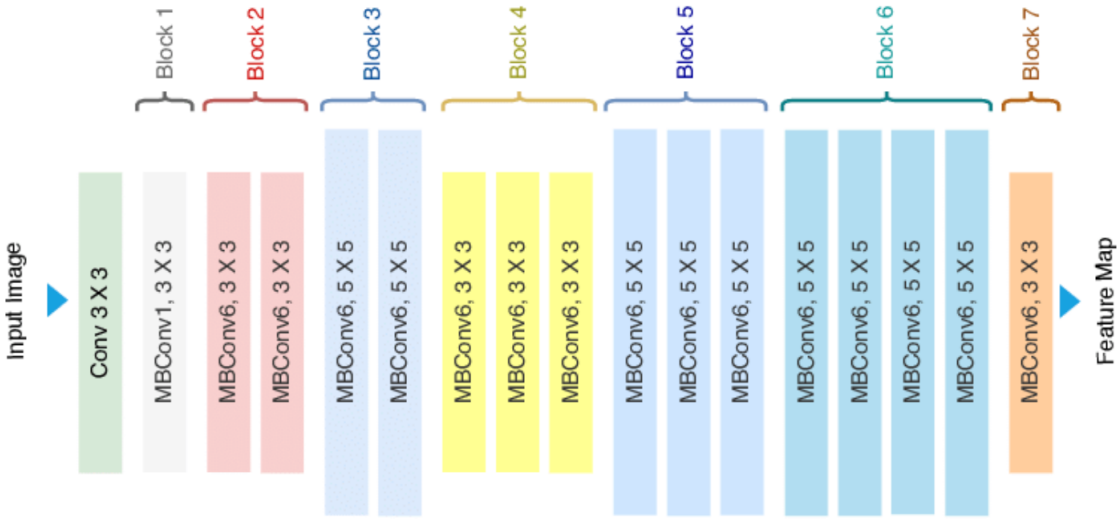


Figure 5: EffNet Base Architecture [Image source]

EfficientNet is designed using Neural Architecture Search(NAS) [12]. The idea is to use MBConv6 blocks from MobileNetV3 and stack these blocks repeatedly just like VGG or ResNet. However, the parameters for each block are derived using the idea of NAS. In this, we have a controller, such as a RNN, to sample block parameters from a probability-determined search space.

We then initialize our Effnet MBConv6 block using these architecture parameters (number of filters, filter height, filter width, stride height, and stride width) and train and test this on a test dataset to get accuracy 'R'. This accuracy along with the gradient of the probability is fed to our controller RNN which acts as an agent. Thus, we sample architecture parameters from our controller,train and test our network to get the accuracy 'R' from this architecture and feed this reward back to the agent in order to get better architectures. In order to make the architectures efficient and faster, EfficientNet also uses 'Floating point operations per second (FLOPS)' also as reward to the controller. This is the basis for the derivation of base EffNet architecture.

The architectures of B5, B6 and B7 will also be similar to the above base model EffNet B0, but the difference being the resolution of input images, the number of channels, and the number of layers of MBConv6. As discussed before MBConv6 is a variation of the MobileNetV3 block that uses 6 layers of depthwise separable convolutions, which are a combination of depthwise convolution and pointwise convolution.

c. **Effnet B6:**

As discussed previously, to improve the mAP value, we tried using Effnet B6 [11]. EfficientNet-B6 is a scaled-up version of the base model, EfficientNet-B0. It is larger than its predecessors (B0 to B5) in terms of depth, width, and resolution. Specifically, it has wider layers, and higher input image resolution.

As a result, it's more powerful and capable of achieving higher accuracy, but it also requires more computational resources to train and deploy as compared to EfficientNet-B0. In terms of performance, EfficientNet-B6 (and its larger counterpart, B7) achieve state-of-the-art accuracy on ImageNet while at the same time being more efficient than the previous models that achieve the same accuracy. Thus, by scaling up the depth, width, and resolution in a balanced way, Effnet B6 achieves excellent performance while being efficient as compared to the traditional Convolutional Models (CNNs) of same size.

d. **Arc Face Loss(Additive Angular Margin Loss for Deep Face Recognition)[10]**

Arc Face - Additive Angular Margin Loss is the similarity function, also known as the loss function, that we deal with. It suggests an additive angular margin penalty to improve the discriminative power of the model between inter-classes in comparison to its predecessors, which are Centre Loss, Softmax Loss, and Sphere Face loss.

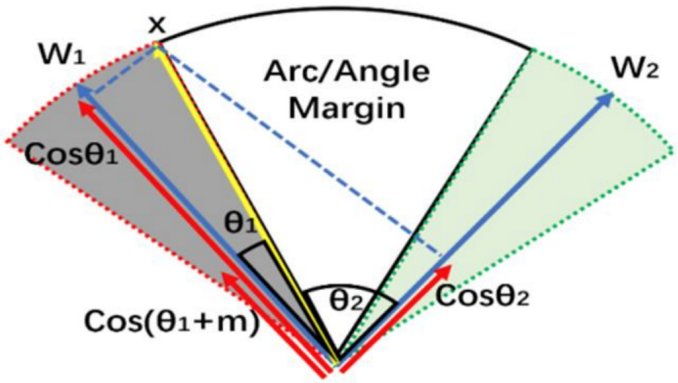


Figure 6: Arc Face loss depiction of two classes

In the context of our system, this is how utilize Arc Face loss. After we normalize every image embedding, all the embeddings reside on a unit sphere in D-dimensional space where D is the size of the embedding. We would like similar images to have embeddings close to each other and dissimilar images with different features to be far from each other. ArcFace adds more loss to the training procedure to encourage similar image embeddings to be close and dissimilar image embeddings to be far from each other.

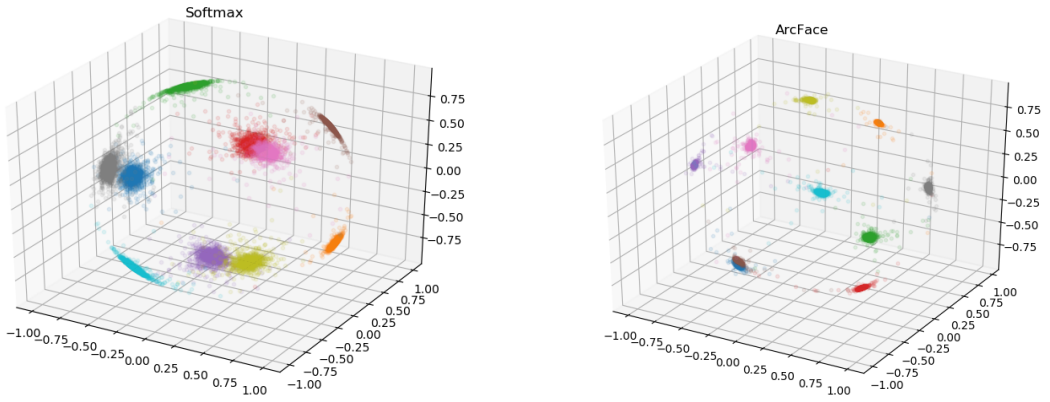


Figure 7:Image source

As you can see in the above images, ArcFace outperforms the softmax loss (i.e softmax activation followed by cross entropy loss) function by separating the dissimilar embeddings in a much better way. This is the reason we chose Arc Face loss for fine-tuning our EffNet-B6.

7. Implementation Details of our Approach:

In this section we will describe and give an overview of the implementation details of our approach.

a. Preprocessing:

Step 1: Creating the Pseudo Labels using YOLO V5 and also storing the confidences corresponding to them. We also store the bounding box co-ordinates

using image width and height so that we can use this data to crop the image correctly.

```
#Pseudo labeling using YOLO
    with open(row.label_path, "w") as f:
        if num_bbox < 1:
            annot = ""
            f.write(annot)
            cnt += 1
            continue
        bboxes_voc = clip_bbox(bboxes_voc, image_height,
image_width)
        bboxes_yolo = voc2yolo(bboxes_voc, image_height,
image_width).astype(str)
        all_bboxes.extend(bboxes_yolo.astype(float))
        bboxes_info.extend([[row.image_id]] * len(bboxes_yolo))
        annots = np.concatenate([labels, bboxes_yolo], axis=1)
        string = annot2str(annots)
        f.write(string)
```

Step 2: Using the sizes of the bounded-boxes, and the pictures that cross a certain threshold value for confidence we crop them. We set a threshold of 60% and crop those images which has more confidence than 0.6. Below is the code snippet, illustrating how cropping is performed.

```
def crop_image(row):
    image_path = row['image_path']
    if 'train' in image_path:
        save_dir = '/tmp/train_images'
    else:
        save_dir = '/tmp/test_images'
    img = load_image(image_path)
    if len(row['bbox']):
        bbox = row['bbox'][0]
        conf = row['conf'][0]
        if conf>=CONF: # don't crop for poor confident bboxes
            xmin, ymin, xmax, ymax = bbox
            img = img[ymin:ymax, xmin:xmax]
    img = cv2.resize(img[...,::-1], dsize=IMG_SIZE,
interpolation=cv2.INTER_AREA)
    cv2.imwrite(f'{save_dir}/{row.image_id}', img)
    return
```

b. Partitioning data into 10 Folds and Serializing the Images:

i. We performed 10-fold cross-validation on both Training Dataset and Test Datasets obtained from previous preprocessing step. We did this 10-Fold cross validation with the intention as it provides a more reliable estimate of the model's performance by using all of the data for both training and evaluation, and averaging the results over multiple runs.

ii. We also serialise each image and write it to a TFRecord file because the TFRecord is a binary format that can store large amounts of data in a compact format. Further, serializing each image and storing it in a TFRecord file can reduce the storage space required for the dataset and make it easier to manage and also reading data from a TFRecord file is faster than reading from other formats like jpg or png files.

iii. One other important reason for TFRecord is that, it is a native format for TensorFlow, which makes it easy to integrate the dataset with TensorFlow code. Once the dataset is in TFRecord format, it can be easily read and preprocessed using TensorFlow's built-in functions and APIs.Following is the code snippet for serializing the images and link to our TFrecords.

```
# Define a function to load and preprocess each image
def load_and_preprocess_image(filename):
    image = Image.open(filename)
    image = image.resize((768, 768)) # Resizing the image for
EffNetB6
    image = np.asarray(image)
    return image
# Loop over each fold
for fold_idx in range(num_folds):
    # Define the output TFRecord file for this fold
    tfrec_file = os.path.join(tfrec_dir,
f"happywhale-2022-test-{fold_idx+1}.tfrec")
    # Serialize each image in the fold and write it to the TFRecord
file
    with tf.io.TFRecordWriter(tfrec_file) as writer:
        for png_file in image_folds[fold_idx]:
            image = load_and_preprocess_image(png_file)
            serialized_image = tf.io.serialize_tensor(image)
            feature = {'image':
tf.train.Feature(bytes_list=tf.train.BytesList(value=[serialized_imag
e.numpy()]))}
            example =
tf.train.Example(features=tf.train.Features(feature=feature))
            writer.write(example.SerializeToString())
```

These serialised images are then fed into the ensemble model whose backbone is either EffNet B5 and B6, You can see the the overall model configurations with both EffNet B5 and EffNet B6 in the below images.

Layer (type)	Output Shape	Param #	Connected to
inp1 (InputLayer)	[(None, 768, 768, 3)]	0	[]
efficientnet-b5 (Functional)	(None, None, None, 2048)	28513520	['inp1[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	['efficientnet-b5[0][0]']
dropout (Dropout)	(None, 2048)	0	['global_average_pooling2d[0][0]']
dense (Dense)	(None, 512)	1049088	['dropout[0][0]']
inp2 (InputLayer)	[(None,)]	0	[]
head/arcface (ArcMarginProduct)	(None, 15587)	7980544	['dense[0][0]', 'inp2[0][0]']
softmax (Softmax)	(None, 15587)	0	['head/arcface[0][0]']

Total params: 37,543,152
Trainable params: 37,370,416
Non-trainable params: 172,736

Layer (type)	Output Shape	Param #	Connected to
inp1 (InputLayer)	[(None, 768, 768, 3)]	0	[]
efficientnet-b6 (Functional)	(None, None, None, 2304)	40960136	['inp1[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2304)	0	['efficientnet-b6[0][0]']
dropout (Dropout)	(None, 2304)	0	['global_average_pooling2d[0][0]']
dense (Dense)	(None, 512)	1180160	['dropout[0][0]']
inp2 (InputLayer)	[(None,)]	0	[]
head/arcface (ArcMarginProduct)	(None, 15587)	7980544	['dense[0][0]', 'inp2[0][0]']
softmax (Softmax)	(None, 15587)	0	['head/arcface[0][0]']

Total params: 50,120,840
Trainable params: 49,896,408
Non-trainable params: 224,432

Figure 8: Ensemble Model with EffNet B5 **Ensemble Model with EffNet B6**

c. EffNet B5 vs EffNet B6:

i. We observed higher mAP values when we used EffNet B6 as backbone in our ensemble model than having EffNet B5 as our backbone. This is consistent with our understanding of their architectures as B6 has more layers and more filters than B5.

ii. Specifically, EfficientNet B6 has 28M parameters and uses 456MB of memory, while EfficientNet B5 has 40M parameters and uses 557MB of memory.

iii. We observed the mAP values in the proximity of 0.65 when we used B5 but with B6 we got mAP around 0.81 which is quite significant but one limitation that we noticed is the increasing computation cost of EffNet B6, makes the process slower when compared to B5.

d. Global Average Pooling 2D:

i. The output of EffNet is connected to Global Average Pooling 2D layer. GlobalAveragePooling2D is a layer in the TensorFlow Keras API that performs global average pooling operation on the spatial dimensions of a 2D tensor.

ii. This layer is used to reduce the dimensionality of the output feature maps before the final classification because it reduces the dimensionality of the input feature maps while retaining important information about the spatial structure of the input.

e. ArcFace Loss:

i. As explained in the previous sections, using ArcFace in the last layer helps to improve the separability of features learned by the individual models, making the ensemble more robust and accurate.

ii. In general ArcFace is used as a head in ensemble models which are used in Face recognition. Given that our task is also analogous to face recognition we proceeded with Arc Face Loss with the difference being to find the related fin patterns across the whale/dolphin images. Below is the code for ArcFace Loss.

```
class ArcMarginProduct(tf.keras.layers.Layer):
    def __init__(self, n_classes, s=30, m=0.50, easy_margin=False,
                 ls_eps=0.0, **kwargs):

        super(ArcMarginProduct, self).__init__(**kwargs)

        self.n_classes = n_classes
        self.s = s
        self.m = m
        self.ls_eps = ls_eps
        self.easy_margin = easy_margin
        self.cos_m = tf.math.cos(m)
        self.sin_m = tf.math.sin(m)
        self.th = tf.math.cos(math.pi - m)
        self.mm = tf.math.sin(math.pi - m) * m

    def get_config(self):

        config = super().get_config().copy()
        config.update({
            'n_classes': self.n_classes,
            's': self.s,
            'm': self.m,
            'ls_eps': self.ls_eps,
            'easy_margin': self.easy_margin,
        })
        return config

    def build(self, input_shape):
        super(ArcMarginProduct, self).build(input_shape[0])

        self.W = self.add_weight(
            name='W',
            shape=(int(input_shape[0][-1]), self.n_classes),
            initializer='glorot_uniform',
            dtype='float32',
            trainable=True,
            regularizer=None)
```

```
    def call(self, inputs):
        X, y = inputs
        y = tf.cast(y, dtype=tf.int32)
        cosine = tf.matmul(
            tf.math.l2_normalize(X, axis=1),
            tf.math.l2_normalize(self.W, axis=0)
        )
        sine = tf.math.sqrt(1.0 - tf.math.pow(cosine, 2))
        phi = cosine * self.cos_m - sine * self.sin_m
        if self.easy_margin:
            phi = tf.where(cosine > 0, phi, cosine)
        else:
            phi = tf.where(cosine > self.th, phi, cosine - self.mm)
        one_hot = tf.cast(
            tf.one_hot(y, depth=self.n_classes),
            dtype=cosine.dtype
        )
        if self.ls_eps > 0:
            one_hot = (1 - self.ls_eps) * one_hot + self.ls_eps /
self.n_classes

        output = (one_hot * phi) + ((1.0 - one_hot) * cosine)
        output *= self.s
        return output
```

The ArcMarginProduct layer is parameterized by several hyperparameters, including the number of classes ‘n_classes’, a scaling factor ‘s’, a margin ‘m’, and an optional label smoothing factor ls_eps’. These hyperparameters control the behavior of the ArcFace margin method and can be adjusted to optimize the performance of the model on a given task.

The build method of the ArcMarginProduct layer initializes the weight matrix W as a trainable parameter with a Glorot uniform initializer. The call method computes the cosine similarity between the normalized input X and the normalized weight matrix W, and applies the ArcFace margin formula to the resulting similarity scores. The output of the layer is the final similarity scores scaled by the factors.

f. Inference Phase :

i. We get the image embeddings as vectors after the image is processed by EffNet and we collect these image embeddings for all the images. Among these image embeddings, we use the K nearest neighbors classification

algorithm, and the classification is done by using the cosine distance metric.

ii. Thus, for each image embedding, we get all the image embeddings that cross a certain threshold and then sort them by the confidence values (1-cosine distance). Further, for each image, the top 5 image embeddings from this sorted order are our predictions for this particular image.

iii. If we get fewer than 5 image embeddings that cross the threshold, we add a prediction saying that this image may belong to a new individual. "new individual" here refers to a totally new whale image that we have so far not seen in the training dataset.

g. Threshold used in KNN:

i. As mentioned above we are using cosine-values for finding the nearest neighbours. To get right threshold for cosine value, we iterated all the way from 0 to 1, in the steps of 0.1 and then decreased the increment in between the range where we got higher mAP value.

ii. If you observe our results, we got our best performance for threshold values between 0.6 to 0.7, so we again narrowed down our search space between these two values by incrementing in the steps of 0.01 and we carried this process till we reached a precision of 5 digits and we observed best mAP at 0.610103.

h. <u>Output values we got from 0 to 1 with increments of 0.1:</u>

```
CV at threshold 0.0: 0.7544756866202933
CV at threshold 0.1: 0.7544756866202933
CV at threshold 0.2: 0.7544756866202933
CV at threshold 0.30000000000000004: 0.7544756866202933
CV at threshold 0.4: 0.7545246726103001
CV at threshold 0.5: 0.7662323242219392
CV at threshold 0.6000000000000001: 0.809829855328043
CV at threshold 0.7000000000000001: 0.7630482348714934
CV at threshold 0.8: 0.6670846804480586
CV at threshold 0.9: 0.5819470298161392
CV at threshold 1.0: 0.547852780771366
```

i. <u>Output values we got from 0.6 to 0.7:</u>

```
CV at threshold 0.6101: 0.8100257992880704
CV at threshold 0.610101: 0.8100257992880704
CV at threshold 0.6101019999999999: 0.8100257992880704
CV at threshold 0.610103: 0.8100257992880704
CV at threshold 0.610104: 0.8100257992880704
CV at threshold 0.610105: 0.8100257992880704
CV at threshold 0.6101059999999999: 0.8100257992880704
CV at threshold 0.610107: 0.8100257992880704
CV at threshold 0.610108: 0.8099768132980635
CV at threshold 0.610109: 0.8099768132980635
CV at threshold 0.6101099999999999: 0.8099768132980635
```

8. **Evaluation Criteria:** Evaluation is done according to the Mean Average Precision@5.

$$MAP@5 = \frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{min(n,5)} P(k) \times rel(k)$$

where U is the number of images,
$P(k)$ is the precision at cutoff k
n is the number predictions per image,

Here $rel(k)$ is an indicator function equaling 1 if the item at rank k is a relevant (correct) label, zero otherwise. Once a correct label has been scored for an observation, that label is no longer considered relevant for that observation, and additional predictions of that label are skipped in the calculation.

For example, if the correct label is A for an observation, the following predictions all score an average precision of 1.0.

Predictions: [A, B, C, D, E], [A, A, A, A, A] and [A, B, A, C, A]

9. **Results:** The below table shows the results format we get. For every test image we will be predicting the top 5 ids of the images that the model believes that belongs to same individual animal. If there are less then five closely related images then model suspects that the test image might be of the new individual. As you can see the 5th and 7th rows of the result, the last prediction in both the rows is “new_individual”.

Image id	Top 5 Predictions for each image
59bc2d264c9731.jpg	47a2f9918aa2 91ed5caeb0d3 beec2e6997c8 73c68d52e748 03b002f56368
5fb61ff7e07076.jpg	2e0b381d3467 39fdf5369ae2 5525e545ae81 32d3b30b473f 19fbb960f07d
0cfc88bad51a18.jpg	e4a55c745bd9 4f64aee1cda9 5393adf1a8ac fc0f7c162cc0 d5b28257276e
3c52966f74d2ad.jpg	978520860ceb d4dfc9c0b367 cfd34e95a962 c917b552fc31 91ed5caeb0d3
561dd86fc30030.jpg	4a67e64bd3b7 5c8d68313a4c 938b7e931166 5bf17305f073 new_individual
d846a86edded63.jpg	03a3bbaeed84 73c68d52e748 982e0af21a70 0fce79a3c211 1bd5ae93c898
50df0a954eb94c.jpg	713eb1a00c3d 64cbd1f56354 f664abace56d 84832851eef0 new_individual

From the output, we can see that whenever we don't have at least 5 images that cross the threshold, we also predict that this image is a “new individual”. Currently at a threshold of 0.6101, we are observing an average confidence value of 0.81677, and with the predictions corresponding to this particular set of parameters, we got a public mAP of 0.72321 on Kaggle.

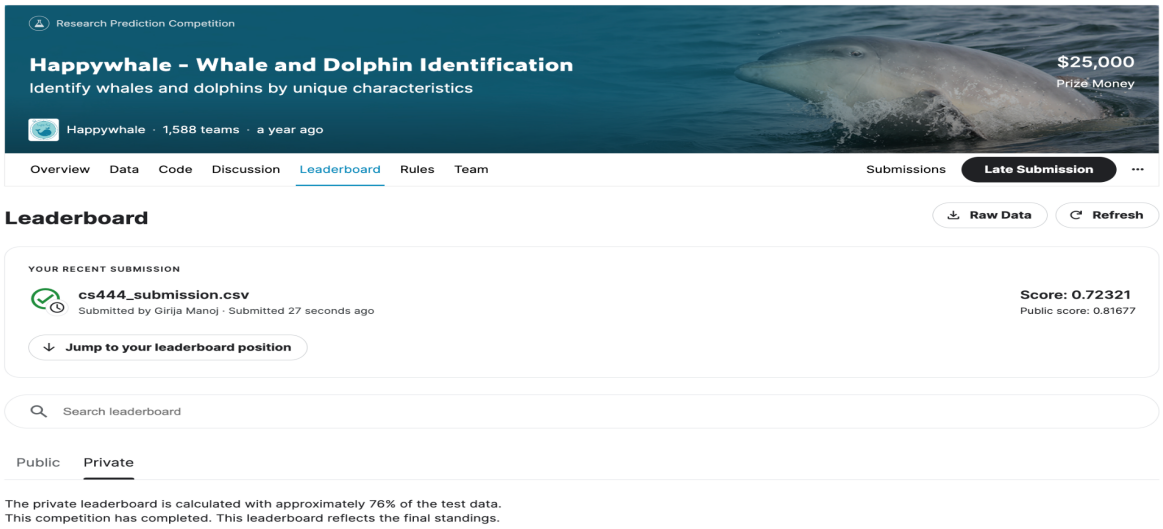


Figure 9: Our latest submission on Kaggle

10. Conclusion:

To summarize, we have developed an automated technique for photo-identification of whales and dolphins. This technique utilizes their unique characteristics and leverages deep learning techniques, specifically convolutional neural networks (CNNs) in order to replace the laborious manual identification process traditionally carried out by researchers. By using YoloV5, EfficientNet, combined with the ArcFace loss function, our strategy performs well with large datasets as well. Overall our system achieves a mean average precision (mAP) of 72% on the leaderboard. The results also include the potential identification of "new individuals" not previously seen in the training dataset thus further helping marine researchers to identify new marine animals. As discussed above in our results section, we got a private mAP(Contest mAP) score of 0.72321.

However, as seen on the leaderboard, the highest mAP score is around 0.8757. After further research we believe that maybe combining predictions from multiple Effnets can further improve our mAP. Apart from that towards the end of our project we came across an improved loss called ArcFace Loss with Dynamic Margin. An initial inspection tells us that using this could have further improved our mAP score given that this loss function suits to our task. Additionally we came across an approach where competitors have used multiple rounds of psuedo labelling in order to further improve their accuracy. At this point, we believe the mAP score improvement is dependent on fine-tuning the hyperparameters rather than having radical changes in the architecture.

The most exciting and interesting learning from our project was how we can use object detection as an image preprocessing technique in order to obtain better images for image identification. This combined process helped us in improving our mAP score from 0.55 to 0.72 which was a significant boost in our score.

11. **Statement of Individual Contribution:** We both dedicated an equal amount of time to the project, spending significant time discussing design decisions and collaborating to implement the system. To ensure efficient progress, we divided the project into modules and allocated each module to one person, and worked together to ensure the successful completion of the project.

Module	Team Member
Literature Survey	Girija Manoj(gmk6), Uday(uk3)
Dataset Preprocessing with YOLO v5	Uday(uk3)
EffNet B5 and B6 Modules	Girija Manoj(gmk6)
Global Average Pooling 2D and ArcFace Loss	Girija Manoj(gmk6)
K-NN module	Uday(uk3)
Project Proposal Report	Girija Manoj(gmk6), Uday(uk3)
Project Progress Report	Girija Manoj(gmk6), Uday(uk3)
Project Final Report	Girija Manoj(gmk6), Uday(uk3)

References:

[1] Dataset: https://www.kaggle.com/competitions/happy-whale-and-dolphin/data

[2] FIN-PRINT a fully-automated multi-stage deep-learning-based framework for the individual recognition of killer whales: https://www.nature.com/articles/s41598-021-02506-6.pdf

[3] Integral Curvature Representation and Matching Algorithms for Identification of Dolphins and Whales: https://arxiv.org/pdf/1708.07785.pdf

[4] Combined Color Semantics and Deep Learning for the Automatic Detection of Dolphin Dorsal Fins https://arxiv.org/abs/1708.02386

[5] WHAT IDENTIFIES A WHALE BY ITS FLUKE? ON THE BENEFIT OF INTERPRETABLE MACHINE LEARNING FOR WHALE IDENTIFICATION https://www.researchgate.net/publication/343401524_WHAT_IDENTIFIES_A_WHALE_BY_ITS_FLUKE_ON_THE_BENEFIT_OF_INTERPRETABLE_MACHINE_LEARNING_FOR_WHALE_IDENTIFICATION

[6] Individual minke whale recognition using deep learning convolutional neural networks https://researchonline.jcu.edu.au/54297/

[7] U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications https://ieeexplore.ieee.org/abstract/document/9446143

[8] Effnet: An Efficient Structure for Convolutional Neural Networks https://ieeexplore.ieee.org/abstract/document/8451339?casa_token=G6D3-QD5BLkAAAAA:lgjI4MuLEwd0gBJn6Q9SITFzOgFxiAhDwIdoiYA0To42BfGjOPR2moopZ9twZ6RBq6QetGQeVFv_

[9] Individual common dolphin identification via metric embedding learning: https://arxiv.org/abs/1901.03662

[10] Arc Face: https://arxiv.org/pdf/1801.07698.pdf

[11] EffNet B6: https://arxiv.org/abs/1905.11946

[12] Neural Architecture Search: https://arxiv.org/abs/1611.01578

[13] Effnet V2: https://towardsdatascience.com/efficientnetv2-faster-smaller-and-higher-accuracy-than-vision-transformers-98e23587bf04