ISSN 1060-992X, Optical Memory and Neural Networks, 2020, Vol. 29, No. 2, pp. 118–132. © Allerton Press, Inc., 2020.

# Method for Whale Re-identification Based on Siamese Nets and Adversarial Training

**W. Wang[a], *, R. A. Solovyev[b], A. L. Stempkovsky[b], D. V. Telpukhov[b], and A. A. Volkov[b]**

[a] *National University of Singapore, Singapore, 119077 Singapore*

[b] *Institute for Design Problems in Microelectronics (IPPM RAS), Russian Academy of Sciences, Moscow, 124365 Russia*

**e-mail: turbo@ippm.ru*

Received November 8, 2019; revised February 26, 2020; accepted March 6, 2020

**Abstract**—Training Convolutional Neural Networks that do well in one-shot learning settings can have wide range of impacts on real-world datasets. In this paper, we explore an adversarial training method that learns a Siamese neural network in an end-to-end fashion for two models—ConvNets model that learns image embeddings from input image pair, and head model that further learns the distance between those embeddings. We further present an adversarial mining approach that efficiently identifies and selects harder examples during training, which significantly boosts the model performance over difficult cases. We have done a comprehensive evaluation using a public whale identification dataset hosted on Kaggle platform, and report state-of-the-art performance benchmarking against result of other 2000 participants.

*Keywords:* convolutional neural networks (CNN), Siamese neural nets, adversarial learning, re-identification problem

**DOI:** 10.3103/S1060992X20020058

## INTRODUCTION

In recent years, deep learning has achieved remarkable results and human-level accuracy in computer vision. However, such algorithms usually do well in tasks when abundant labeled examples exist in each class, where neural networks can be trained to extract complex statistics and high level features from training data. However, similar models may suffer in tasks where few or even single labeled example is available in each category—which is usually named few-shot or one-shot learning—and the model cannot learn enough features or may end up severely overfitting. Model performance can even suffer when the dataset exhibits long-tail nature—labeled categories are too many, and most of them fall into one-shot or few-shot categories.

### *Dataset and Evaluation Metric*

In real-world settings, it is not rare to see few-shot learning cases. In the recently completed whale identification challenge hosted on Kaggle platform [1], the dataset presents a typical few-shot learning setting—a set of 25.361 training images of whale tails, covering over 5004 unique whale ids that the model needs to learn and predict. Among those training data, 9664 images are of un-identified whales, or 'new_whales', which belong to none of those 5004 known labels. This leaves only 15697 labeled images of 5004 ids.

As we can see, there are on average only 3 images per id, which makes the dataset very sparse. Each image belongs to only one of the 5004 ids. A glimpse of images from competition dataset is shown in Fig. 1.

These photos vary in dimensions, channels (RGB or Gray), image quality, object orientation, years taken, and so on, which mimics the real world data collection.

Another challenging aspect of this task comes with the long-tailed nature of data, as shown in Fig. 2. For example, there are around 2000+ labels with only 1 image, and around 1250+ labels with 2 images, and so on. The majority of classes have only 1 or 2 images, which makes it a typical few-shot problem.

In the test set which the submitted models will be evaluated on, there are in total 7960 images. Some of the 7960 may belong to 'new_whale' category.

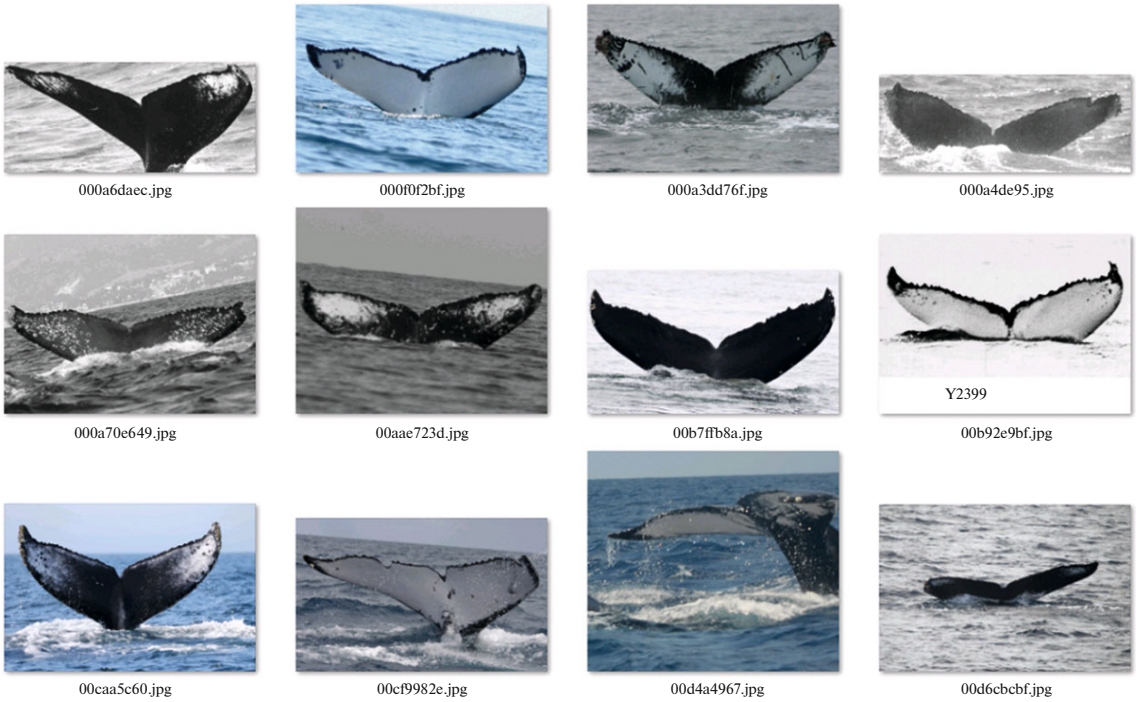


**Fig. 1.** Image examples in training set.

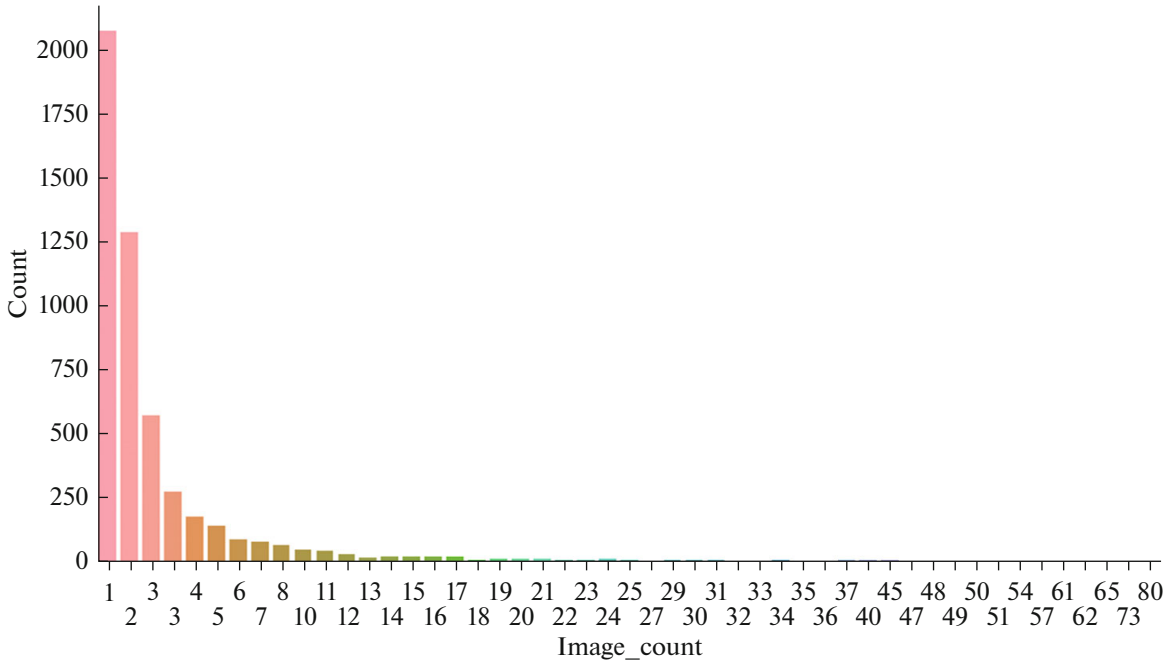


**Fig. 2.** Label distribution with number of images within each label.

Metric MAP@5 is used in this problem to evaluate contestant predictions (with small changes):

$$\mathrm{MAP}@5 = \frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{\min(n,5)} P_u(k)\,rel_u(k). \quad (1)$$

**Table 1.** Metric MAP@5 with different scores explanation

| True Label | Prediction | $k$ | Score for this prediction |
|---|---|---|---|
| [w] | [w, ?, ?, ?, ?] | 1 | 1.0 |
| [w] | [?, w, ?, ?, ?] | 2 | $0 + 1/2 = 0.5$ |
| [w] | [?, ?, w, ?, ?] | 3 | $0/1 + 0/2 + 1/3 = 0.33$ |
| [w] | [?, ?, ?, w, ?] | 4 | $0/1 + 0/2 + 0/3 + 1/4 = 0.25$ |
| [w] | [?, ?, ?, ?, w] | 5 | $0/1 + 0/2 + 0/3 + 0/4 + 1/5 = 0.2$ |
| [w] | [?, ?, ?, ?, ?] | 5 | $0/1 + 0/2 + 0/3 + 0/4 + 0/5 = 0.0$ |

Here, $U$ is the total number of images, $P_u(k)$ is accuracy, $rel_u(k)$ is a function that returns 1 if the answer at position $k$ is correct, $n$ is number of predictions. The change in metric is that calculation would stop after the first occurrence of the correct whale. We don't have to sum up to 5, only up to the first correct answer. Since each test image has only one correct answer, there are only six possible scores for each prediction entry. So the possible precision scores per image are either 0 or $P(k) = 1/k$. Table 1 shows the detailed scores explanation. In this table '*w*' means – correct prediction, while symbol '?' – incorrect prediction.

The final leaderboard score is the averaged MAP@5 score across each prediction in the 7960 test images

### *Solution Overview*

Solving this problem as a typical multi-class classification problem has been attempted by many teams. However, solution through classification would face significant number of challenges, which include the followings:

• Few-shot and long-tailed nature of the data set. There are significant numbers of classes that have only few or single training sample, which makes it hard for the model to learn well from limited training samples.

• Large number of classes greatly increases the last fully connected layer of neural networks, which eventually increases the number of trainable parameters and complexity of the system.

• The classification solution will not be easily scalable should the number of individuals increases from 5000 to an even higher number, say 10000, or 1000000, in the future. Adding new individuals will require the change of at least last layer of the network, which would need the network to be retrained.

This task is closely related to Person Re-identification [2–4, 23], but with photos of whales instead of people, and similarity learning [35–37]. For example, Double Sampling method proposed in [24] helps resolve the issue of sampling from enormous combinations of input, but due to the limited number of images per ID and long tailed distribution of labels for this data, this method would not seem as an ideal approach.

To address those challenges, we propose a Siamese network architecture that consists of a branch model (i.e. a ConvNets model) and head model, and a specially designed training procedure that optimizes both models efficiently in an end-to-end fashion.

Our method was inspired by the solution of previous contest winner Martin Piotte on Kaggle [5], but with improved design of ConvNets models and some training details, and a deeper understanding of how Siamese network extract features by aid of visualization. Metric MAP@5 is used here to evaluate all model predictions.

### *Siamese Nets Description*

Siamese networks were first introduced in early 1990s for signature verification tasks [6, 7] and are now often used for Person Re-identification task [26–28] and metric learning [32–34]. Siamese networks consist of two neural networks that extract high-level features from two input images. We call these parts Branch models (abbreviated as B-model). In the last layers these models are connected using the Head model (H-model), which measures the distance between B-model feature vectors. Both branch models share identical weights.

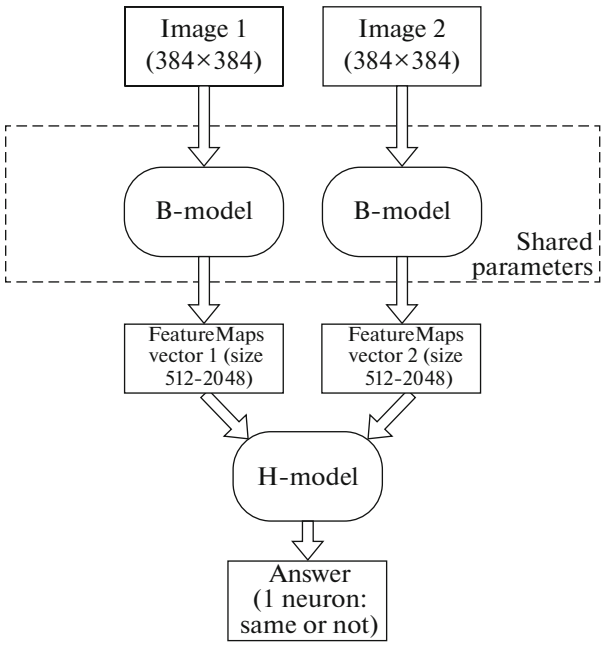


**Fig. 3.** Siamese network.

Siamese network is schematically shown in Fig. 3. The size of feature vector depends on the B-model used, which varies from about 512 to 2048 in our case.

In our proposed approach, we used well-known deep convolutional networks [40] as B-models: DenseNet121 [8], Inception_v3 [9], Renset50 [10], SE-Resnext50 [11], which were pre-trained on the large ImageNet [12] dataset. We have also used customized ConvNets architectures that embed similar, but fewer, residual blocks as in Resnet50, which we then train from scratch using only competition data. In notation form, we are extracting a d-dimension feature vector $f(x) \in Rd$ from the input image $x$ with our B-model to represent the image in a lower dimensional space.

The H-model compares two vectors obtained from B-model, and predicts the probability of two images being the same whale. The easiest way is to find the distance between these vectors, for example, using the Euclidean metric. However, as was shown in Martin's solution [5], we are able to design a neural network that 'learns' the distance in an end-to-end fashion together with B-model:

1. Calculate several metrics (e.g. sum, product, absolute distance, squared distance) between the two feature vectors: $[x + y, x*y, |x - y|, |x - y|\hat{}2]$.

2. Train a small neural network that learns to find optimum weights combining these metrics. Such network can have additional hidden layers to learn non-linear interactions among the features.

3. Take weighted sum of these values and put it through the final sigmoid layer to predict the likelihood of being matched.

H-model is presented schematically in Fig. 4.

By notation, with the feature vectors $f(x_i)$ and $f(x_j)$ of the input image pair $(x_i, x_j)$ from B-model, we are getting $g(f(x_i), f(x_j)) \in R1$ from H-model that measures the probability of $(x_i, x_j)$ being the same whale.

Overall we strive to train our model that is able to extract representative features from any image pair $(x_i, x_j)$, so that $g(f(x_i), f(x_j))$ is as large as possible for $(x_i, x_j)$ where $y(x_i, x_j) = 1$, where $y(...)$ indicates the ground truth label for any pair of images, and as low as possible for $(x_i, x_j)$ where $y(x_i, x_j) = 0$.

### *Bounding Box Models*

Before feeding the raw images to the Siamese networks, we used bounding boxes to cut off the areas that are large enough to cover the whole tail. This is useful in two ways: first, it reduces redundant information that is most likely noise from the surrounding environment, such as sea water, flying birds, etc.; second, when being resized to the actual input required by the branch model, details which are important for determining the individual would be more preserved, compared to being resized from original image.

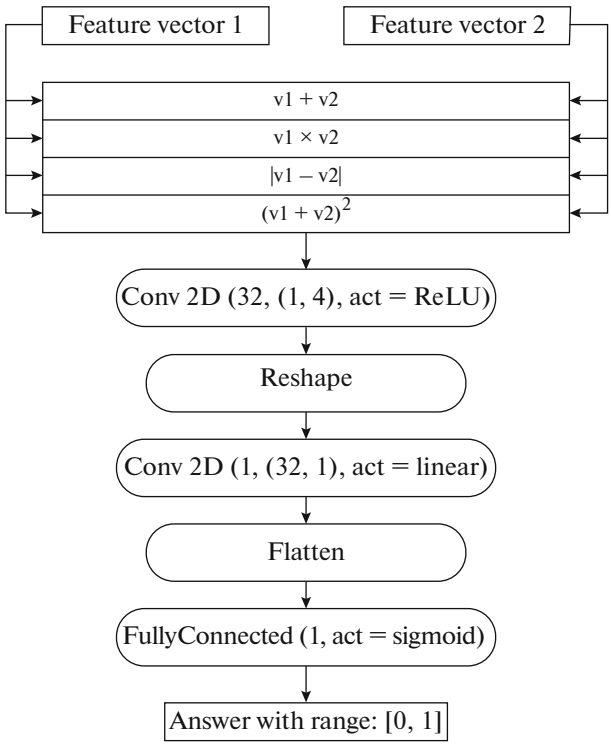


**Fig. 4.** Structure of H-model.

Two different Object Detection networks were used here as bounding box models: a customized ConvNets network built on the basis of VGG16, and the RetinaNet [13, 14] with backbone based on ResNet152, pre-trained on Open Images Dataset [15, 38, 39]. Both models output $(x, y)$ coordinates of two corners which determine the box, i.e. the model has four output nodes. Each network was trained for several epochs using cyclic annealing [16] on the small data of 1200 images with hand-labeled bounding-boxes. As we assumed there is only one whale in the image, the resulting coordinates are just averaged of both outputs. We then use both trained bounding box models to generate boxes on entire train and test dataset. Average IOU of the boxes based on cross validation is about 0.9273, which produce good enough boxes for most of the images, as shown in Fig. 5 for some examples.

### *Adversarial Training Scheme*

This is the most important part of our proposed approach, which details our training steps for Siamese network.

We use binary cross entropy as objective function at the end of graph, and formulate it as a binary classification task. In each epoch, we will sample both positive (i.e. matched images) and negative (i.e. unmatched images) pairs as training data. The positive cases are sampled from images within same whale identity, and negative cases are across different whale identities.

However, randomly sampling from the entire negative space would result in a lot of data $(x_i, x_j)$ having $g(f(x_i), f(x_j))$ close to zero, where $(x_i, x_j)$ is a random negative pair (i.e. $y(x_i, x_j) = 0$). To use as an intuitive example, Fig. 6 shows two cases of 'easy' and 'hard' pairs of unmatched whales (for negative pairs). The majority of negative pairs would become redundant as they do not contribute to training but are still used as training data, which results in poor convergence of the model.

A good mining strategy would be to constantly select difficult cases in each epoch based on the current snapshot of network, which is similar to the idea of Triplet Selection in [17]. This means we need to select training pairs $(x_i, x_j)$ such that the model predicts a higher value of $g(f(x_i), f(x_j))$ at $y(x_i, x_j) = 0$.

The naive way would be for each training image $x_i$, to compute pairwise probabilities against all other training images $x_j$ where $y(x_i, x_j) = 0$, and select $x_j$ which gives the highest $g(f(x_i), f(x_j))$ as pair $(x_i, x_j)$.

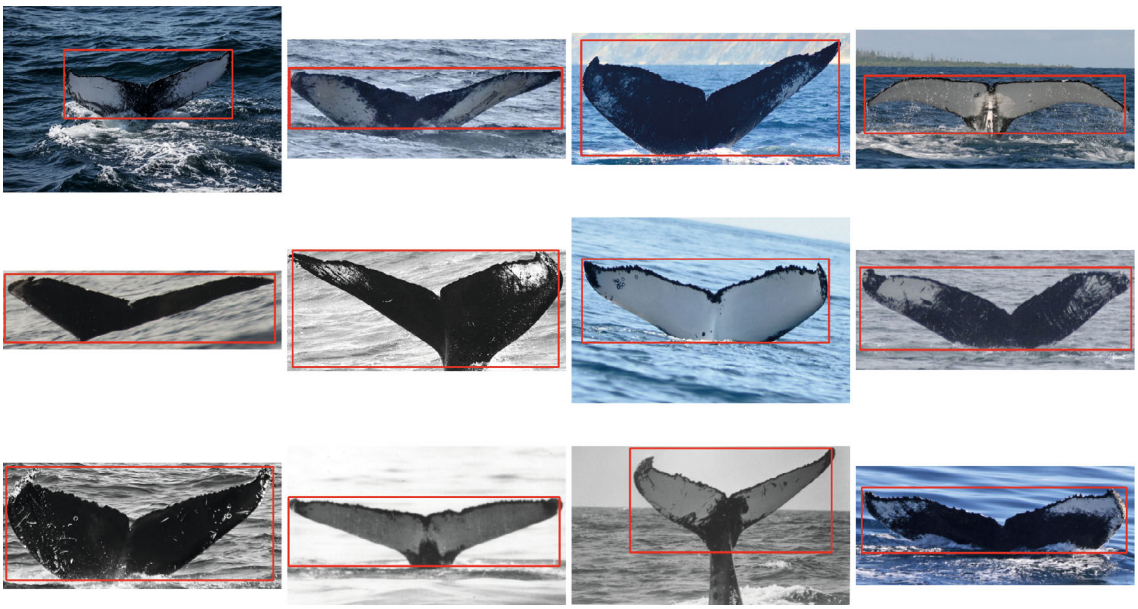

**Fig. 5.** Bounding boxes for whales.

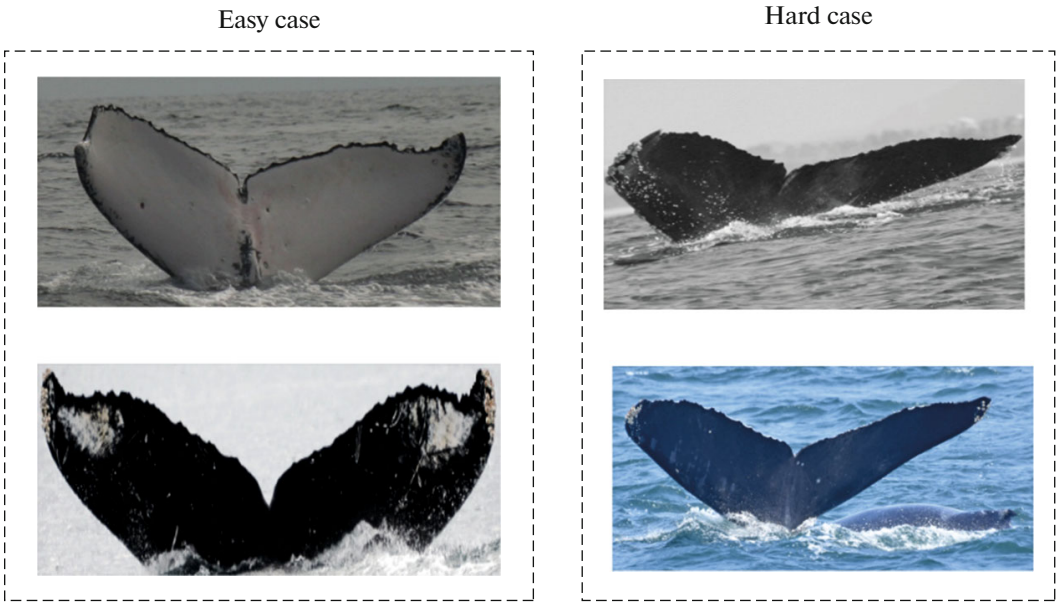


**Fig. 6.** An ‘easy’ or random sampled case versus a ‘hard’ case of unmatched training pairs.

However, this approach would lead to some outliers, or generally difficult-to-predict images, to always get selected, such as images with weird angle of views, or with too much noise, and the model would simply ‘memorize’ those images as negative samples.

To resolve this issue, we need a balanced pair mining algorithm - In each training epoch, we have image pairs *(A, B)* such that, exactly half of pairs belong to unmatched IDs, while the other half of pairs belong to matched IDs

We introduce proposed Adversarial Mining approach here for negative pairs selection, as detailed in Algorithm 1. Basically, in every $n$ (e.g. $n = 5$) epochs, we use current network snapshot to generate a train-vs-train square matrix $T$, where $T_{ij}$ represents the probability between image $x_i$ and image $x_j$ being a match.

In every subsequent training epoch, for every training image *xi*, we need to find its matched train image $x_j$, which satisfies: (1) $y(x_i, x_j) = 0$; (2) every $x_j$ is unique; (3) overall selection of $(x_i, x_j)$ results in maximum of sum of probabilities. We will use Linear Assignment Problem solver [18, 25] to find such solution over the train-vs-train matrix, as in Algorithm 1.

We also introduce a new hyperparameter $K$ which controls the scale of randomness during the selection. The higher the value of $K$, the more randomness thus the less difficult the selected pairs will be; the lower the value of $K$, the more difficult the selected pairs.

Algorithm 1: Adversarial mining approach

**Input:** Current Network Checkpoint *g*, All Training Images *X,* Scale Parameter *K*

Set the fixed large positive value*: maxval, e.g. maxval = 10000*

Set number of training epochs *n* after each adversarial mining, *e.g. n = 5*

**for** *i in* 1, 2, ... *, length(X)* **do**

  **for** *j in* 1, 2*, ... , length(X)* **do**

    **if** $y(X_i, X_j) = 1$ **then**

      // to prevent matched pairs from being selected

      *1.* $T_{ij} = maxval$

    **else**

      *2.* $T_{ij} = -g(f(X_i), f(X_j)) + K$ * random.uniform(0, 1)

    **end if**

  **end**

**end**

**for** *k in* 1, 2, ..., *n* **do**

  3. *l* = linear_assignment_solver(*T*)

  // l is the list of selected image pairs

  **for** *(i,j) in l* **do**

    // prevent same *(i,j)* to be selected in next epoch

    4. $T_{ij} = maxval$

  **end**

  // return *l* for epoch *k* as selected negative pairs

**end**

**Output**: the list of selected image pairs *l* in each training epoch *k*

At the beginning of training loop, it's not a good idea to start with a very low value of K. To put it to the extreme, if we start by feeding in unmatched pairs that appear to be 'more similar' than matched pairs, the model may learn to predict similar images as different whales, and dissimilar images as same whales. The model will simply not converge.

The values of K used during training can be seen in section 10 Full training schedule, which were found based on several rounds of experiments.

On the other hand, one may argue that such adversarial strategy should also be applied to positive pairs selection. However, it is not necessary based on our experiment. The training pairs generated for positive cases should be derangement pairs—within each ID, each image should only be paired up with a different image other than itself, and each image should be used in equal number of times in order to be balanced. There are on average only 3 images per ID in training set, so total number of derangement pairs that can be generated for each ID is only 2 on average, which is quite limited. Therefore a simple random sampling approach has been used to select positive pairs.

### *Gradual Increase of Input Image Dimension*

Another strategy we adopted in our proposed approach that significantly improved model performance is to gradually increase input size during training. Here *image_size* refers to the input image dimension. For example*, image_size = 384* means the images are resized to dimension of *(384, 384, 3)* to the network input layer. All input images we have used are having three channels of RGB, unless otherwise specified.

During training, we gradually increase the image_size from 224/384, 512, 768, up to 1024, as we noticed that model performance improved significantly when we increased the input image dimension as

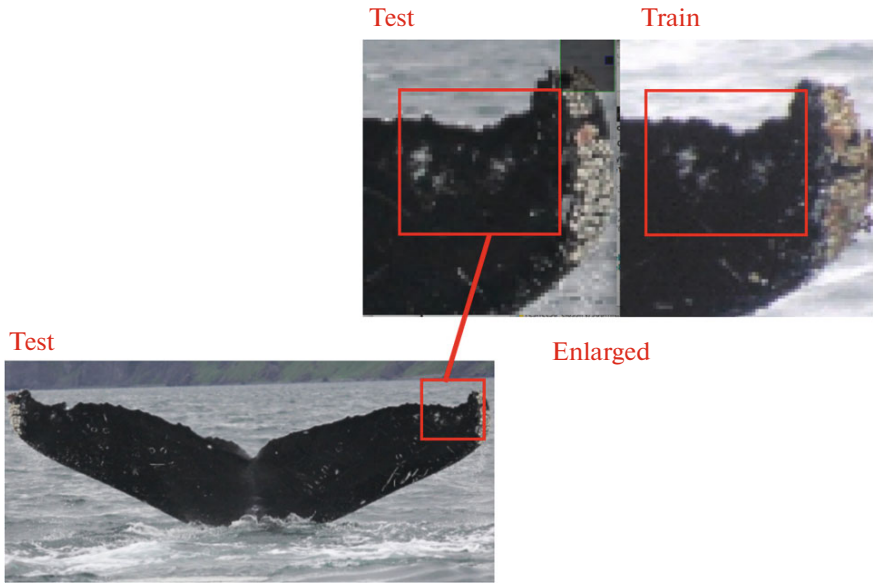


**Fig. 7.** Comparing matched image pair between test and train sets at higher resolution.

soon as the model's loss has converged at current *image_size*. This is because some of the tail features can be recognized as similar features only at higher resolution, as in Fig. 7 below:

By the time the *image_size* is increased, we should freeze the B-model, and only train the H-model, for at least several epochs, before we move on to train both models end-to-end. This is important in stabilizing the training, and to ensure the gradients flow would not disrupt the earlier layers of the network in first rounds of training.

### *4-fold Training Approach*

We adopt a typical 4-fold cross validation approach to train our models—for each round, we use one fold of the data as validation, and the other three folds as training. Final result is the average of all four rounds. During the split, we will ignore IDs having only single images, as they cannot be used to generate matched image pairs. For IDs that have $n$ images where $n > 1$, we can do 4-fold split in the following way:

- if $n$ is 2, the images will only be used for training
- if $n$ is 3, they will be randomly split into 3 folds, so that only 1 or no image will be in validation
- if $n$ is larger than 3, they will be randomly split into 4 folds.
- Additionally for the validation fold, we will randomly add in around 25% of '*new_whale*' images from the training that are not being used. This is simply to mimic the distribution of new whales in test set.

Once the data has been split up based on the above approach, we will train four models on each split while using the validation fold for early stopping, and to select the best model checkpoints.

### *Model Inference*

After the models have been trained, inference can be done in the following way:

For each test image, we first use the trained model to obtain a test-vs-train probabilities vector across all train images. Since we have four models, we will average the four test-vs-train vectors.

Based on this averaged test-vs-train vector, we then obtain the top 5 IDs based on ID probabilities, in descending order. Each ID's probability is simply the max of all images' probabilities belonging to that ID.

From top 1 to top 5, if any of the ID's probabilities is below a predefined threshold $t$, we will then insert the '*new_whale*' token into the first position that is below $t$, and remove the 5th ID from the prediction due to this insertion.

The final 5 prediction outputs, including the '*new_whale*' if inserted, will be the output for this test image. The above steps are then repeated for all test images.

We show an example in Table 2 on how to generate inference output for one test image.

**Table 2.** Predictions for single test image

| Image Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ... |
|---|---|---|---|---|---|---|---|---|---|
| Train ID | w1 | w10 | w1 | w5 | w7 | w8 | w9 | w7 | ... |
| Probability | 0.995 | 0.991 | 0.985 | 0.8 | 0.7 | 0.55 | 0.13 | 0.11 | ... |

If threshold = 0.98, the prediction would be: (w1, w10, new_whale, w5, w7) for this test image. We tuned the threshold $t$ such that it gives expected distribution of new whales in test set. In Fig. 8, we show our end-to-end whale identification workflow using our trained Siamese network.

### *Image Augmentations*

Data augmentation [20] plays a critical role in our training due to the limited number of images as well as few-shot characteristic of the dataset. We adopted on-the-fly image augmentation strategy, and used ways such as shear, rotation, flipping, contrast, Gaussian noise, blurring, color augmentations, graying and random crops.

The logic of augmentation was as follows:

1. Randomly convert 5% of images to gray scale, based on the estimated gray percentage in the dataset.

2. The data contains a lot of images where JPEG artifacts are seen, so JPEG compression with different quality settings was applied to the images.

3. Augmentations related to color changes were also applied that emulate different color balance and brightness on cameras, as well as all sorts of distortions emulating blurred, out-of-focus or fuzzy pictures.

4. Noise augmentations were added, which are often present in photographs taken in low light or with high ISO.

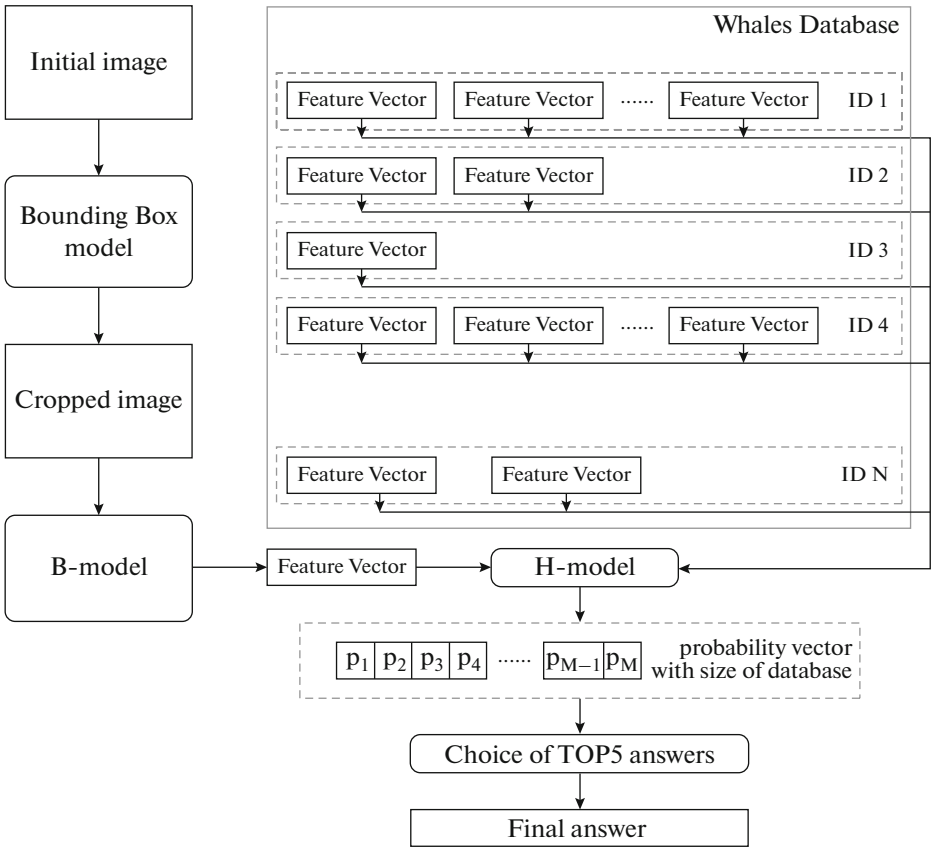


**Fig. 8.** Complete whale identification procedure for inference.

**Table 3.** Training scheme for Siamese network with branch model using Densenet121

| B—model type: Densenet121 pretrained on ImageNet | | | | | |
|---|---|---|---|---|---|
| Epoch | K | Lr | Image size | Dataset | Freeze BModel |
| 1–20 | 1.0 | 8e-5 | 224 × 224 | PlayGround | – |
| 21–40 | 0.5 | 2e-5 | 224 × 224 | PlayGround | – |
| 41–100 | 0.25 | 1e-5 | 224 × 224 | PlayGround | – |
| 101–200 | 0.25 | 8e-6 | 224 × 224 | PlayGround | – |
| 1–20 | 1000 | 2e-5 | 384 × 384 | PlayGround | + |
| 21–40 | 1.0 | 2e-5 | 384 × 384 | PlayGround | – |
| 41–60 | 0.5 | 1e-5 | 384 × 384 | PlayGround | – |
| 61–120 | 0.25 | 1e-5 | 384 × 384 | PlayGround | – |
| 121–220 | 0.25 | 8e-6 | 384 × 384 | PlayGround | – |
| 1–15 | 1.0 | 8e-5 | 384 × 384 | Main | + |
| 16–35 | 0.5 | 8e-5 | 384 × 384 | Main | – |
| 36–55 | 0.25 | 4e-5 | 384 × 384 | Main | – |
| 56–75 | 0.25 | 2e-5 | 384 × 384 | Main | – |
| 76–95 | 0.25 | 1e-5 | 384 × 384 | Main | – |
| 96–195 | 0.25–0.05 | 8e-6 | 384 × 384 | Main | – |
| 1–15 | 1.0 | 8e-5 | 512 × 512 | Main | + |
| 16–35 | 0.5 | 8e-5 | 512 × 512 | Main | – |
| 36–55 | 0.25 | 4e-5 | 512 × 512 | Main | – |
| 56–75 | 0.25 | 2e-5 | 512 × 512 | Main | – |
| 76–95 | 0.25 | 1e-5 | 512 × 512 | Main | – |
| 96–195 | 0.25–0.05 | 8e-6 | 512 × 512 | Main | – |

5. Random skew within 20 degrees was also added to emulate skewed horizon.

### *Full Training Schedule*

In addition to the hyperparam $K$ and *image_size*, we also have the following parameters that we are changing through the training process:

1. Learning rate—$Lr$

2. L2 regularization—$L2$

Since each model only differs in their B-models, which used different deep convolutional network architectures, we will show our full training schedule only for our best single model—Siamese network with DenseNet121 [8] pretrained on ImageNets as B-model—in Table 3.

There was around 5K labeled training data from other PlayGround whale competition [19]. This dataset was used for the similar task of whale identification before, but the IDs have been masked differently from this competition, so we do not have 1-to-1 mapping between PlayGround IDs to the current IDs in our main dataset. We therefore pretrained our models on this additional data, before we train models on main dataset. Pretraining using PlayGround dataset offers the benefits of having additional data source, which helps the model converge faster subsequently. This demonstrates another advantage of proposed method—easy adapting on new datasets for pretraining or fine-tuning model, even if the dataset has different labels.

### *Experimental Results*

Table 4 below shows a comprehensive comparison of private LB, public LB as well as Local Scores (LS) among all 9 individual models we have trained. The models vary in both input image dimensions, as well as B-model architectures—DenseNet121 [8], SE-ResNeXt [11], and Customized ConvNets models that use residual modules and SE-ResNet modules [31] (with 'Customized ConvNet v2' in Model Name).

**Table 4.** A comparison among all trained Siamese networks with various input dimensions, pretrained model weights and architectures

| Model Name | ImageNet weights | Private LB | Public LB | Validation score | Input image size |
|---|---|---|---|---|---|
| Densenet121 | + | 0.959 | 0.960 | 0.964 | 512 × 512 × 3 |
| SE-ResNext50 | + | 0.959 | 0.960 | 0.963 | 384 × 384 × 3 |
| Customized ConvNet v2 1024 | – | 0.954 | 0.958 | 0.960 | 1024 × 1024 × 3 |
| Customized ConvNet 1024 | – | 0.956 | 0.955 | 0.960 | 1024 × 1024 × 3 |
| Customized ConvNet 768 | – | 0.952 | 0.951 | 0.956 | 768 × 768 × 3 |
| Customized ConvNet v2 768 | – | 0.952 | 0.954 | 0.958 | 768 × 768 × 3 |
| Customized ConvNet 512 | – | 0.943 | 0.936 | 0.949 | 512 × 512 × 3 |
| Customized ConvNet Gray Image 512 | – | 0.928 | 0.925 | 0.937 | 512 × 512 × 1 |
| Customized ConvNet 384 | – | 0.928 | 0.926 | 0.936 | 384 × 384 × 3 |

**Table 5.** Comparison of scores between Top 5 teams

| Team/ Model Name | Public leaderboard | Private leaderboard |
|---|---|---|
| Top 1 Team | 0.97461 | 0.97309 |
| Top 2 Team | 0.97419 | 0.97208 |
| Top 3 Team | 0.97359 | 0.97113 |
| Top 4 Team | 0.97296 | 0.96783 |
| Top 5 Team | 0.96914 | 0.96781 |
| Proposed best single model (DenseNet 121) | 0.95925 | 0.95981 |
| Proposed ensemble model | 0.97052 | 0.96916 |

The top 2 models from Table 3—Densenet 121 and SE-ResNext50—achieved the best private LB score of 0.959. However, due to GPU memory limitation, we trained both models with up to image size of 512 and 384, respectively.

We also notice that performance for Customized ConvNets models improves significantly when the input image dimension increases (e.g. from 512 to 768).

Since all models here have varieties in their architectures as well as training procedures, ensemble would help reduce model variances thus improve prediction accuracy. We simply take the linear weighted sum of test-vs-train matrices for the top 5 models in Table 3 to get our final ensembled test-vs-train matrix. We then apply the same inference procedure described on this ensembled matrix to get our final output for test data.

Table 5 shows comparison between our results against top 5 teams for this competition. As it shows, our proposed method would have got 4th place (0.96916 on private LB) by using ensemble of 5 models.

The following tools were used for the experiments: Python 3.* with libraries Keras, Tensorflow, OpenCV (cv2), Numpy, Albumentations, LAP (linear assignment problem solver using Jonker-Volgenant algorithm). The product is cross-platform: the code runs on both Windows and Linux and available on GitHub [21].

### *Siamese Network Visualization*

In this session, we use HeatMaps (or activation maps [29]) to visualize how neural networks recognize whales and where its attention focuses given the input images.

Output of the last BatchNormalization [30] layer from B-model was used to generate HeatMap. For DenseNet121 with input image size of 512, this output has dimension of (16, 16, 1024). Here 16 × 16 is the size of each feature map, and the total number of feature maps is 1.024. This layer is followed by RELU activation, and then by GlobalMaxPooling layer, which simply selects the maximum value from each of the 1024 feature maps.

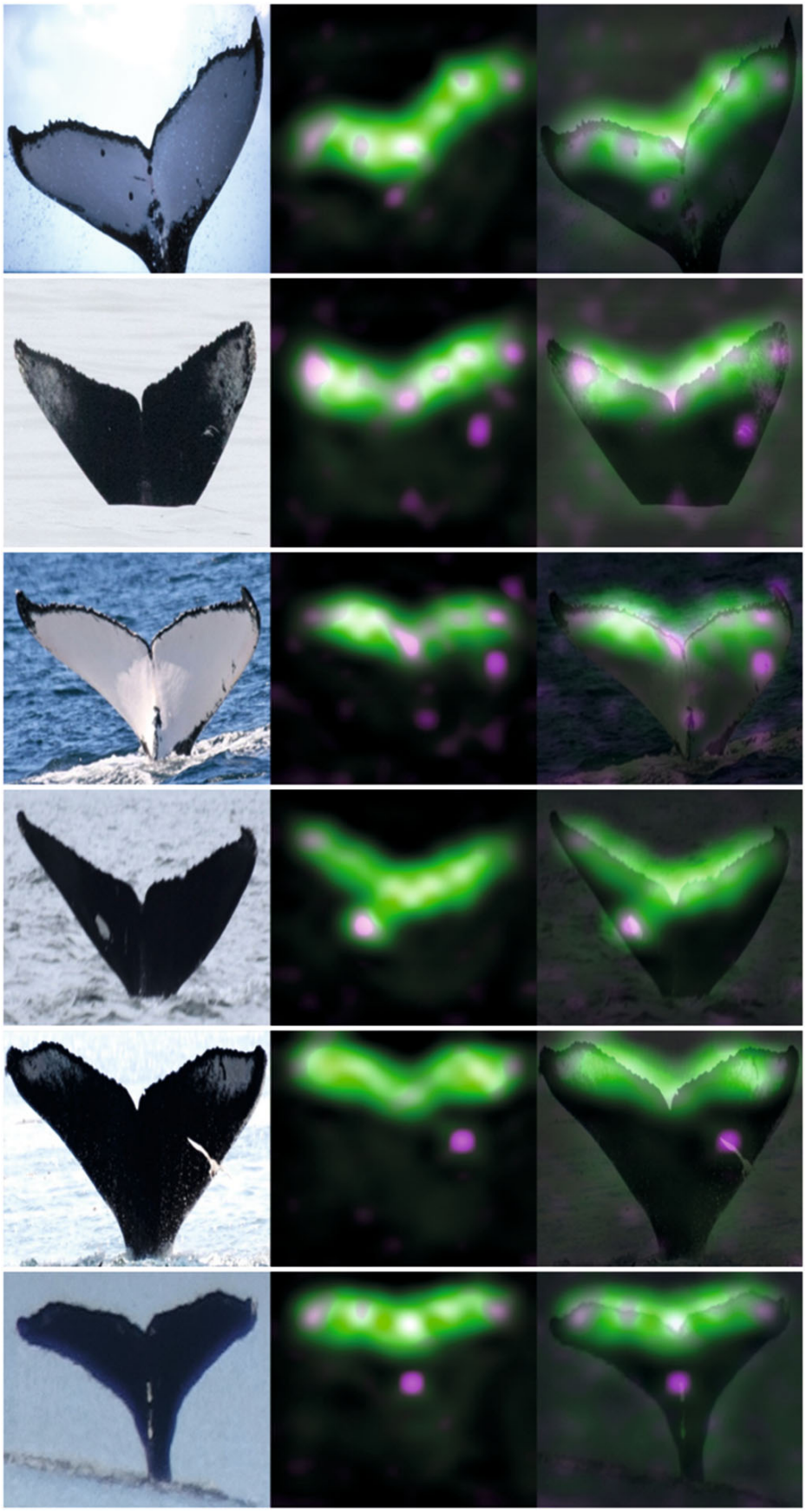

**Fig. 9.** HeatMap examples for test images for neural network based on B-model DenseNet121.

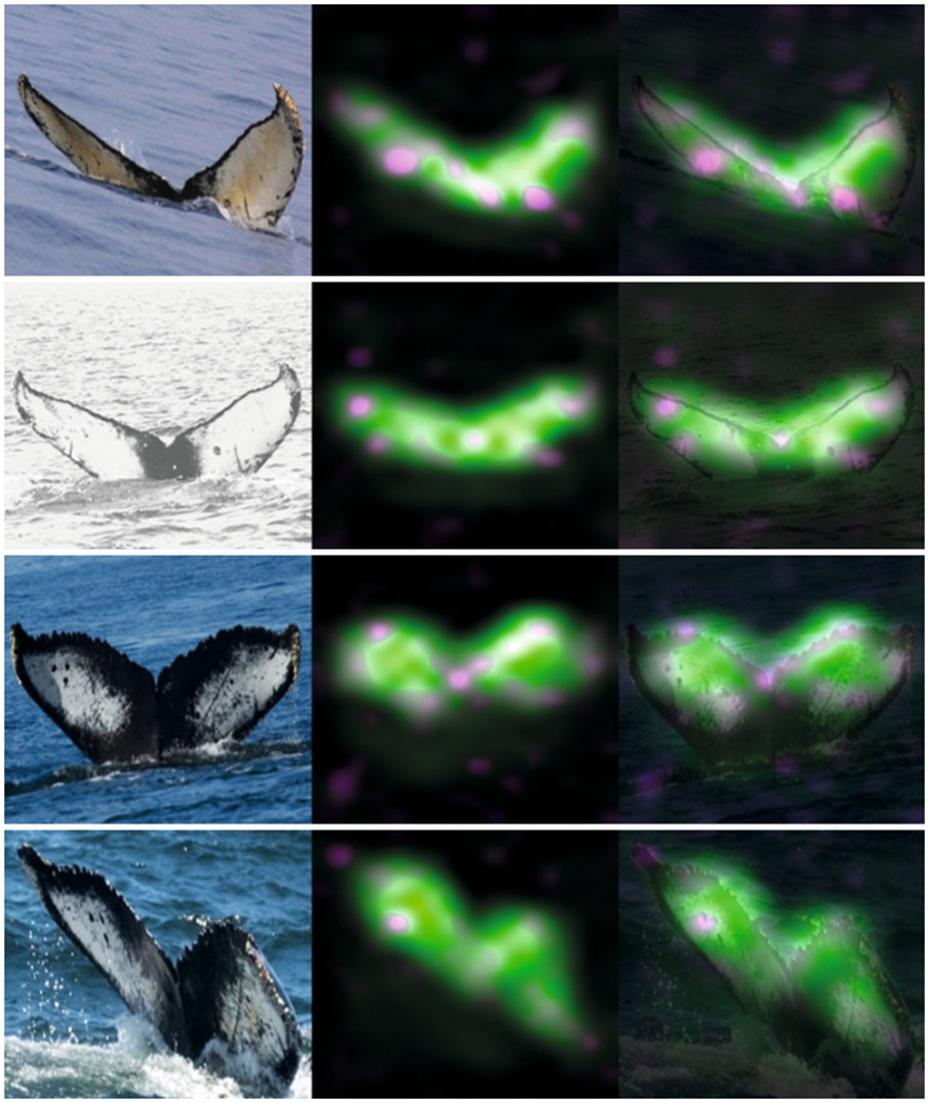

**Fig. 10.** HeatMap examples for matched test and train image pairs (1st and 3rd rows are two different test images; 2nd and 4th rows are their matched train images).

To understand where the model is 'focusing on', we can count the total number of times each of the (16 × 16) positions takes the max value over all 1024 feature maps. This would give a (16 × 16) matrix that stores the counts. We then use this matrix as two channels (e.g. Red and Blue) in the final HeatMap image. The last channel (e.g. Green) comes from the standard deviation for each of the 16 × 16 position over all the 1024 feature maps. Finally, we concatenate all three channels as an RGB HeatMap image, and use upsampling to project the HeatMap image to the size of its original input image. We show the HeatMaps generated by our model on some images from test dataset in the Fig. 9 below.

Areas with higher standard deviation are marked in green, where the (G)reen channel on the HeatMap image is high. Pink dots highlight the 'activation areas'—areas where the HeatMap image has higher values in its Red and Blue channels—which further determine the feature vectors that will go to H-model. White dots are the combinations of the both—both higher standard deviation and higher 'activation'. As we can see, the model very well marks the whales' visual features, scars, birthmarks and the edge and shape of the tails, which contain the key features that are needed to make differentiation.

In Figure 10, we show HeatMaps of some correctly matched image pairs – test images and its corresponding matched train images. We can see that our method is quite robust to color shift as well as affine transformation. Code for visualization is at [22].

## CONCLUSIONS

In this work we proposed a Siamese networks architecture consisting of both B-Model and H-Model, and adversarial mining method to improve training. We evaluated our model based on a whale identification challenge against other 2000 contestants, and demonstrated state-of-the-art performance of the method.

We believe our method has wider range of impacts on real-world datasets. However, there are also future areas to further improve our method, including a deeper understanding of representation learning from the ConvNets model (or B-Model), a more efficient way for inference on larger scale of images, and more domain-adaptive ways of image augmentations.

## CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

## REFERENCES

1. https://www.kaggle.com/c/humpback-whale-identification/.
2. *Person Re-Identification,* Gong, S., et al., Eds., Springer Science & Business Media, 2014. https://doi.org/10.1007/978–1–4471–6296–4_1
3. Zhang, X., Luo, H., Fan, X., Xiang, W., Sun, Y., Xiao, Q., and Sun, J., Alignedreid: Surpassing human-level performance in person re-identification, arXiv preprint arXiv:1711.08184, 2017.
4. Farenzena, M., Bazzani, L., Perina, A., Murino, V., and Cristani, M., Person re-identification by symmetry-driven accumulation of local features, *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* 2010, pp. 2360–2367.
5. https://www.kaggle.com/martinpiotte/whale-recognition-model-with-score-0-78563.
6. Koch, G., Zemel, R., and Salakhutdinov, R., Siamese neural networks for one-shot image recognition, *ICML Deep Learning Workshop,* 2015, vol. 2.
7. Bromley, J., Bentz, J.W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Säckinger, E., and Shah, R., Signature verification using a Siamese time delay neural network, *Int. J. Pattern Recognit. Artif. Intell.,* 1993, vol. 7, no. 4, pp. 669–688.
8. Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K.Q., Densely connected convolutional networks, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2017, pp. 4700–4708.
9. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z., Rethinking the inception architecture for computer vision, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016, pp. 2818–2826.
10. He, K., Zhang, X., Ren, S., and Sun, J., Deep residual learning for image recognition, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016, pp. 770–778.
11. Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K., Aggregated residual transformations for deep neural networks, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2017, pp. 1492–1500.
12. Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, and Li Fei-Fei, ImageNet: A large-scale hierarchical image database, *2009 IEEE Conference on Computer Vision and Pattern Recognition,* 2009, pp. 248–255.
13. Lin, T.Y., Goyal, P., Girshick, R., He, K., and Dollár, P., Focal loss for dense object detection, *Proceedings of the IEEE International Conference on Computer Vision,* 2017, pp. 2980–2988.
14. https://github.com/ZFTurbo/Keras-RetinaNet-for-Open-Images-Challenge-2018.
15. Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., and Ferrari, V., The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale, arXiv preprint arXiv:1811.00982, 2018.
16. Smith, L.N., Cyclical learning rates for training neural networks, *2017 IEEE Winter Conference on Applications of Computer Vision (WACV),* 2017, pp. 464–472.
17. Schroff, F., Kalenichenko, D., and Philbin, J., Facenet: A unified embedding for face recognition and clustering, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015, pp. 815–823.
18. Burkard, R.E. and Cela, E., Linear assignment problems and extensions, in *Handbook of Combinatorial Optimization,* Boston, MA: Springer, 1999, pp. 75–149.
19. https://www.kaggle.com/c/whale-categorization-playground.
20. Buslaev, A., Parinov, A., Khvedchenya, E., Iglovikov, V.I., and Kalinin, A.A., Albumentations: Fast and flexible image augmentations, arXiv preprint arXiv:1809.06839, 2018.
21. https://github.com/aaxwaz/Humpback-whale-identification-challenge.
22. https://www.kaggle.com/zfturbo/visualisation-of-siamese-net.

23. Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., and Tian, Q., Scalable person re-identification: A benchmark, *Proceedings of the IEEE International Conference on Computer Vision (ICCV),* 2015, pp. 1116–1124.
24. Liao, W., Ying Yang, M., Zhan, N., and Rosenhahn, B., Triplet-based deep similarity learning for person re-identification, *Proceedings of the IEEE International Conference on Computer Vision (ICCV),* 2017, pp. 385–393.
25. Jonker, R. and Volgenant, A., A shortest augmenting path algorithm for dense and spare linear assignment problems, *Computing,* 1987, vol. 38, pp. 325–340.
26. Chung, D., Tahboub, K., and Delp, E.J., A two stream Siamese convolutional neural network for person re-identification, *Proceedings of the IEEE International Conference on Computer Vision (ICCV),* 2017, pp. 1983–1991.
27. Zheng, M., Karanam, S., Wu, Z., and Radke, R.J., Re-identification with consistent attentive Siamese networks, arXiv preprint arXiv:1811.07487, 2018.
28. Shen, C., Jin, Z., Zhao, Y., Fu, Z., Jiang, R., Chen, Y., and Hua, X.S, Deep Siamese network with multi-level similarity perception for person re-identification, *Proceedings of the 25th ACM International Conference on Multimedia,* 2017, pp. 1942–1950.
29. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A., Learning deep features for discriminative localization, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016, pp. 2921–2929.
30. Ioffe, S. and Szegedy, C., Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167, 2015.
31. Hu, J., Shen, L., and Sun, G, Squeeze-and-excitation networks, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2018, pp. 7132–7141.
32. Sanakoyeu, A., Tschernezki, V., Büchler, U., and Ommer, B., Divide and conquer the embedding space for metric learning, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2019.
33. Oh Song, H., Xiang, Y., Jegelka, S., and Savarese, S., Deep metric learning via lifted structured feature embedding, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2016, pp. 4004–4012.
34. Schroff, F., Kalenichenko, D., and Philbin, J., Facenet: A unified embedding for face recognition and clustering, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2015, pp. 815–823.
35. Sanakoyeu, A., Bautista, M.A., and Ommer, B., Deep unsupervised learning of visual similarities, *Pattern Recognit.,* 2018, vol. 78, pp. 331–343.
36. Bautista, M.A., Sanakoyeu, A., and Ommer, B., Deep unsupervised similarity learning using partially ordered sets, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2017, pp. 1923–1932.
37. Zagoruyko, S. and Komodakis, N., Learning to compare image patches via convolutional neural networks, *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2014.
38. Iglovikov, V. and Shvets A., U-net with VGG11 encoder pre-trained on ImageNet for image segmentation, arXiv preprint arXiv:1801.05746.33, 2018.
39. Iglovikov, V., Seferbekov, S., Buslaev, A., and Shvets, A., TernausNetV2: Fully convolutional network for instance segmentation, arXiv:1806.00844 [cs.CV], 2018.
40. Solovyev, R.A., Stempkovsky, A.L., and Telpukhov, D.V., Study of fault tolerance methods for hardware implementations of convolutional neural networks, *Opt. Mem. Neural Networks,* 2019, vol. 28, no. 2, pp. 82–88.