

# Application of Deep Learning techniques for the photo-identification of fish individuals

nuriagomv

## Model implementation: code outline

The main file of our system for training is “main\_training.py”, whose function calls are outlined below. The optimization of hyperparameters in “bayesian\_optimization\_parameters.py” works essentially the same way, but without fixing the parameters to a value.

The following parameters are set: **dataset\_path**, **batch\_size**, **load\_dataset\_again**, **use\_augmentation**, **params\_augmentation**, **learning\_rate**, **momentum**, **std\_prob\_threshold**, **input\_shape**, **model\_name**, **tensorboard\_log\_path**

Instantiating class...

```
siamese_network = SiameseNetwork(...)
```

The following parameters are set: **process = 'train'**

Instantiating class...

```
Images_loader= Loader(...)
```

The following parameters are set: **image\_height**, **image\_width**, **images\_dictionary**, **ray\_names**, **train\_dictionary**, **source\_to\_evaluate**, **validation\_dictionary**, **evaluation\_dictionary**, **evaluable\_rays**

All images are loaded with **load\_dataset()** and stored in **images\_dictionary**

If **use\_augmentation** is true then the dictionary of images is augmented...

```
images_dictionary = ImageAugmentor(...).apply_augmentation()
```

The following parameters are set: **n\_eval**

Split the dataset with **divide\_train\_valid\_eval()** which, for each ray in **evaluable\_rays** takes **n\_eval** images for test set, and divides the remaining in 75%-25% for train and validation. All images of non evaluable rays are for the train set.

Filtering of rays with sufficient images and saving divided sets in a .pkl file

The following parameters are set: **summary\_writer**

Defining architecture of the network with **construct\_siamese\_architecture()** as defined in the methodology, which also prints summaries of the **model** defines the optimizer as Stochastic Gradient Descent and copies.

The training starts when the following parameters are set: **validate\_each**, **number\_of\_train\_iterations**...

```
validation_accuracy = siamese_network.train_siamese_network()
```

Loop as many times as **number\_of\_train\_iterations**

Getting train batch...

```
images_labels = images_loader.get_train()
```

As many rays as **batch\_size** are randomly selected and for each of them:

- We append in **images\_path** 3 photos of that ray and 1 photo of a different ray so that for each individual we get a pair of same-individual images and a pair of different-individuals images.

Finally, what this function returns is...

```
images, labels = convert_path_list_to_images_and_labels(images_path, is_one_shot_task = False)
```

which creates a list of two lists of length **number\_of\_pairs** in which to store each image of the pairs, and the list **labels** containing 1s or 0s depending on whether the respective pair is a same-individual or a different-individuals, and returns both randomly permuted.

```
train_loss, train_accuracy = model.train_on_batch(images, labels)
```

Each **validate\_each** iterations:

```
validation_accuracy = images_loader.few_shots_task()
```

For each **ray** in **evaluable\_rays**:

For each **test\_image** of that **ray**:

```
images, _ = get_one_shot_batch(current_ray = ray, test_image)
```

which creates the batch of pairs of images: the **test\_image** of the **current\_ray** against an image of the support set of that ray and against an image of all the remaining evaluable rays (i.e., the labels are [1,0,0,...]).

```
probabilities = model.predict_on_batch(images)
```

Right prediction if max(probabilities) is for the first position and there is variability.

The accuracy for each **ray** is the mean of the accuracies obtained for each of its **test\_image**.

The **validation\_accuracy** is the mean global accuracy across all evaluable rays.

Writing results to tensorboard and overwrites the model and its weights if the obtained validation accuracy is better.

Once trained and validated, we make the predictions over the evaluation (test) set loading the weights of the best model

```
evaluation_accuracy = siamese_network.predict_after_train()
```

which predicts following the defined methodology that resembles Ensemble Learning.

Figure 1: Training outline.

And the main file for prediction is “main\_prediction.py”, which shares classes with the previous scheme but whose outline goes as follows:

The following parameters are set: **dataset\_path**, **input\_shape**, **model\_name**

We set the architecture of our net...

```
siamese_network = SiameseNetwork(...)
```

And load the model (i.e., the weights) with which we want to predict.

Then, we load in the dictionary **images\_to\_predict** the images of the unidentified individuals that we want to predict, which are those in folders containing “**new\_recapture**”.

Predicting with...

```
flag_new_individuals = siamese_network.load_and_predict(images_to_predict), which predicts following the defined methodology that resembles Ensemble Learning and creates the list flag_new_individuals that will contain the path of the individuals detected as new ones.
```

We store the information of new individuals in a .txt file...

```
store_new_individuals(dataset_path, flag_new_individuals)
```

Finally, when enough new individuals have been gathered, we process them to create new folders of identification...

```
siamese_network.process_new_individuals()
```

```
"Creating pairs to compare in a pairwise manner...\n "  
"Predicting same/different over the pair:"  
print(pair)  
images, _ = images_loader.convert_path_list_to_images_and_labels(pair, is_one_shot_task = True)  
probability = model.predict_on_batch(images)[0][0]  
if probability > 0.5:  
    print("The pair of images belong to the same individual")  
else:  
    print("The pair of images belong to different individuals")
```

Figure 2: Prediction outline.