Encode information from *n* grayscale images to a single "RGB" image (for training a CNN): our ideas in Matlab and PyTorch

Written by David Petrovic, Federico Luisetto, Umberto Salviati

Università degli Studi di Padova
Department of Information Engineering
Via Gradenigo 6/b 35131 - Padua, Italy

Abstract

This paper illustrates an in-depth study of an existing research framework titled "*Encode information from n grayscale images to a single 'RGB' image (for training a CNN)*". In pursuit of expanding the methodologies applied within the scope of this framework, we explore alternate approaches for the conversion of grayscale images to the RGB format. By implementing diverse transformation techniques, we endeavor to enrich the training process of a Convolutional Neural Network (CNN). Our study extends to a comparative analysis of the network's performance, achieved through training on both MATLAB and PyTorch platforms with the same data. Through comprehensive experimentation, we evaluate the impact of these distinct methodologies on the CNN's ability to extract meaningful features from the generated "RGB" representations. Our result show that by just applying our approaches to create color on images we have better result enhancing the accuracy from 2% to more than 5% with respect to the model presented by the original project (Mitra et al. 2019).

Our research

In our research, we embarked on an exploratory journey building upon the original study's MATLAB codebase. Our efforts encompassed multiple facets: we replicated the dataset creation process and model training using MATLAB, ensuring fidelity to the original methodology. Subsequently, we undertook the task of translating the training code into the PyTorch framework, which unveiled subtle divergences in the pre-trained AlexNet model imported between the MATLAB and PyTorch environments.

Expanding our approach, we introduced innovative techniques to augment the study's diversity. In MATLAB, we devised a script to transform grayscale images into RGB using the median algorithm, thus creating a new dataset. Meanwhile, in Python, we implemented the alpha-trimmed mean algorithm, generating eight distinct datasets by varying the alpha parameter. Our experimentations comprised training each dataset, including the original one, on both the original MATLAB model and the created PyTorch model.

Through a comparative analysis, we examined the ramifications of these modifications. This entailed assessing the performance of each trained model across different datasets and evaluating the divergence in outcomes between the two programming environments. Our investigation not only shed light on the intricate interplay of factors influencing model behavior but also underscored the significance of platform-specific nuances in yielding divergent results. Overall, our work contributes a comprehensive understanding of the intricacies associated with cross-environment model training, dataset manipulation, and algorithmic augmentation.

Datasets creation

The dataset construction involved a meticulous process. Each foraminifera was individually positioned near the central region of the microscope's field of view and brought into focus manually. The significance of the light source's direction in accentuating distinct geometric characteristics within the foraminifera led the original paper's authors to employ a Light Emitting Diode (LED) ring. By utilizing this equipment, they captured a set of 16 images, each taken under varying angles of illumination.

The arrangement of the LED ring ensured consistent light source heights, and the orientation of the lighting underwent a systematic alteration, encompassing a full 360° range with uniform spacing between adjustments. Through this comprehensive procedure, a dataset was meticulously curated, incorporating specimens from a diverse range of species. The dataset comprises specific species, including G. Bulloides (178 specimens), G. Ruber (182), G. Sacculifer (150), N. Dutertrei (151), N. Incompta (174), N. Pachyderma (152), along with a substantial representation of other species (450 specimens), primarily composed of planktic foraminifera.

Due to the limited scale of the foraminifers dataset, training deep neural networks from scratch was not viable. Thus, in the original paper, the authors used a pre-trained Convolutional Neural Network (CNN). By using this pre-trained CNN, the magnitude of training images needed is much lower. However, it is crucial to note that the pre-trained models are designed for color images. As a consequence, there was the necessary to create a new dataset in which each RGB image encodes information from all 16 grayscale images of a given specimen. In the following, the techniques that we have implemented for creating datasets of RGB images are explained.

Percentile algorithm

This technique was used by the authors of the original paper for creating the dataset of RGB images. We had the original Matlab script in which, for each specimen, it processes the grayscale values of each pixel from the 16 individual images. It computes the 10th percentile, median, and 90th percentile values for these pixel sets. Subsequently, these computed values are strategically assigned to the red, green, and blue channels of the corresponding pixel in the RGB image. This meticulous process culminates in the creation of a singular color composite image. So, by running that script with the original grayscale images as input, we were able to recreate the dataset that was used in the original project.

Median algorithm

The first algorithm that we have implemented for the creation of the dataset is based on the median value of the pixels of the grayscale images. The aim of this algorithm is to take in input 16 grayscale images and compress them into a single RGB image. The algoritm is divided in three major phases as shown in Fig. 1:

1. image collection.
2. pixel median.
3. output composition.

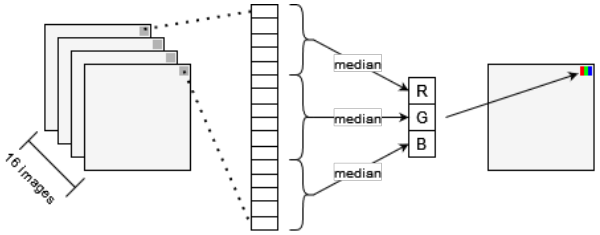
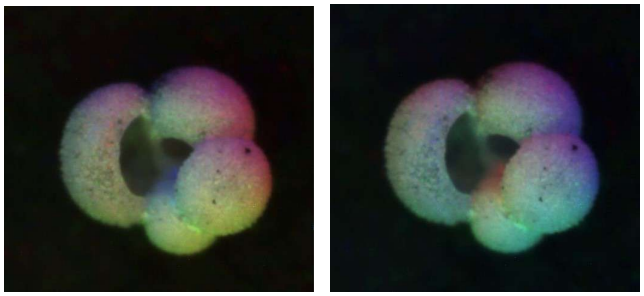


Figure 1: Median algorithm

Image collection The input of the algorithm is composed by 16 grayscale images of the same subject in the same position, with the only difference in light source positioned differently. The input images are divided in three sets to represent the RGB 3-channel image output: the first 5 images for R, 6 for G and the last 5 for B.
We give one more image to G because green light is perceived better by the eyes.

Pixel median From the collected images, for each formed group (R, G, B), we take all the pixels [i,j] from each image and we compute the median to establish the output value of the three channels in [i,j] of the output image.
We do the median so to cut off picks of brightness or darkness caused by noise and corpuscles in suspension.

Output composition For each pixel [i,j] we take the triplet (R,G,B) and we save it into a blank output image.
We save the image in RGB (Fig. 2a), and not in BGR (Fig. 2b) as OpenCV standard, to obtain a better looking and more saturated image.



(a) Median RGB (b) Median BGR

Figure 2: Output median images

Complexity and space Regarding time complexity and occupied memory space this algorithm computes each output image separately. So, for each round, we only have 16 images in input and one in output without any intermediate datastore. For creating the new image the algorithm passes each pixel [i,j] of the output image, assigning to each pixel the corresponding triplet calculated in that specific time on the specific pixel and layer. Given that, the algorithm is an O(1) in both time and space.

Alpha-trimmed mean algorithm using CNN (Zhang, Isola, and Efros 2016)

Our approach consists of using alpha-trimmed mean to clean the images, and then applying a convolutional neural network based on the Zang (Zhang, Isola, and Efros 2016) study for image colorization. Specifically, since our input data comprises 16 photos of the same subject taken under different lighting conditions, we chose to combine the 16 black-and-white images by calculating their mean. To reduce salt noise, we employed an alpha-trimmed mean with alpha=4, striking a balance between an arithmetic mean and a median filter.

Following this preprocessing stage, each subject is represented by one image with reduced noise. The next phase involves colorization. For this purpose, we adopted the approach and pre-trained CNN developed by Zang et al. (Fig. 3), which was trained on a dataset of one million images. Given that our study focuses on less common subjects, the foraminifera, obtaining their accurate colors is challenging, as they tend to be pale and often are represented by black and white pictures due to their small size.

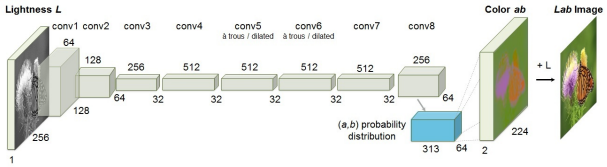
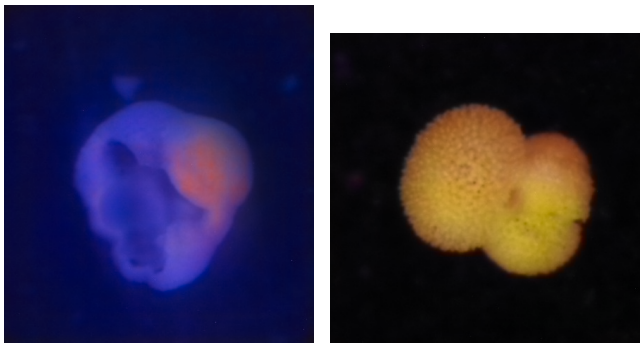


Figure 3: The diagram representig the CNN uesed in Zang study(Zhang, Isola, and Efros 2016)

While some colors might not have been accurately rendered due to the absence of ground truth, this method

has successfully generated colored images, often producing plausible results (Fig. 4b). It's worth noting that this CNN might not accurately handle photogrammetric pivots, making it less suitable for our specific case (Fig. 4a). Nonetheless, it has managed to generate believable images, although not always achieving perfection, as evident in the images.



(a) An example of implausible colors in an output

(b) An example of plausible colors in an output

Figure 4: Output alpha-trimmed and CNN algorithm

Differences between PyTorch's model and Matlab's model

In order to better evaluate the models (the one of the original paper and ours), each of these was trained and tested (with the original dataset and the ours) in the same laptop (which does not have a GPU so all the process was computed by a CPU Intel i7-1165G7 with 16 GB of RAM). In both models, we use the default pre-trained AlexNet, changing only the last layers of the CNN to adapt the standard networks to our problem.

During the process of recreate the original model in PyTorch, we have seen that the results were not comparable with the ones of the original network in Matlab. Thus, we have started to study the architectures of the networks and we have discovered that there are some differences that arise during the training and testing processes (as shown in Fig. 5, where we can see the difference in terms of accuracy between the two models over a total of 10 datasets that we have created).

In Tab. 1 are shown the architectures of the two networks and, for example, we can easily see that between a *Relu* layer and a *MaxPooling* layer in Matlab there is also a *CrossChannelNormalization* layer, used to normalize each activation, but in PyTorch this layer is not present.

To make a fair comparison of the model over the two environments, the architectures of the two networks must be equal. Thus, we have tried to export the pre-trained Alexnet model from Matlab and import it to PyTorch by using several methods but no one has accomplished the goal.

For this reason, we have decided to make a comparison between the two environments with the objectives of highlighting the difficulty of operating on cross-environments and testing the performance of the pre-trained Alexnet model on a wide number of datasets of Foraminifera.

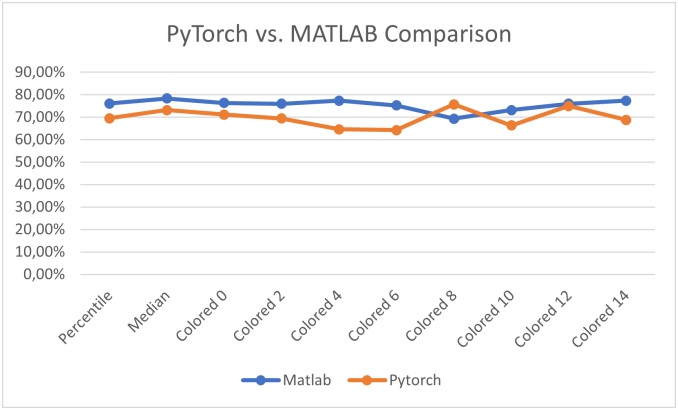


Figure 5: Evaluation of Models Accuracy

#	PyTorch model	Matlab model
1	Conv2d	Convolution2DLayer
2	ReLU	ReLULayer
3	MaxPool2d	CrossChannelNormalizationLayer
4	Conv2d	MaxPooling2DLayer
5	ReLU	GroupedConvolution2DLayer
6	MaxPool2d	ReLULayer
7	Conv2d	CrossChannelNormalizationLayer
8	ReLU	MaxPooling2DLayer
9	Conv2d	Convolution2DLayer
10	ReLU	ReLULayer
11	Conv2d	GroupedConvolution2DLayer
12	ReLU	ReLULayer
13	MaxPool2d	GroupedConvolution2DLayer
14	AdaptiveAvgPool2d	ReLULayer
15	Dropout	MaxPooling2DLayer
16	Linear	FullyConnectedLayer
17	ReLU	ReLULayer
18	Dropout	DropoutLayer
19	Linear	FullyConnectedLayer
20	Linear	ReLULayer
21	ReLU	DropoutLayer
22	Dropout	FullyConnectedLayer
23	Linear	SoftmaxLayer
24	ReLU	ClassificationOutputLayer
25	Linear	
26	CustomClassifier	

Table 1: Comparison between models architectures

Results and discussion

In this section we will compare the results obtained in our tests between datasets and CNN models.

Matlab model

Generated datasets have different results in the training process as shown in Fig. 6 (the one used for colored alpha is the best one of the set) but by looking at Fig. 7, the dataset that allow the best performance in terms of accuracy on test data (thus, a better generalization) is the one created with the median algorithm. This result arises because of the nature of the algorithms used:

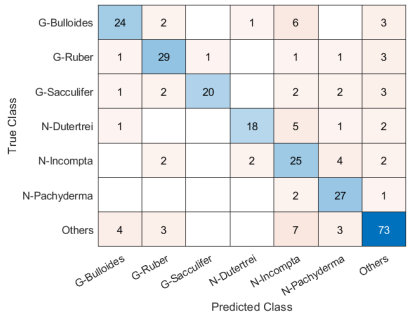
1. median, the best one with an accuracy of 78.32%, uses three subsets of the 16 images of the sample *i* to compute three output channels. For each subset the median is computed, so we exclude pixels with very low/high values, using only the more representative ones.
2. alpha-trimmed mean (alpha=4), with an accuracy of 77.35%, uses the 16 images of sample *i* to compute the RGB output image *i*. In the process the lowests and hightests values of the pixels are removed (trimmed), so we exclude pixels with very low/high values, and then the mean is computed in the remaining ones. This similarity with the median algorithm permits to obtain less than 1% of accuracy difference between the two.
3. percentile, the one of the original paper (Mitra et al. 2019), with an accuracy of 76.06% is the worst dataset created based on accuracy score in Matlab's training process.

PyTorch model

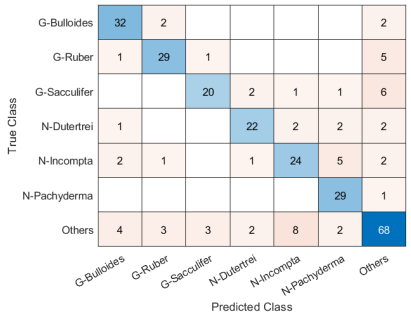
For what concern the model built in PyTorch, as shown in Fig. 8, we observe that the conclusions regarding are different then before.
As stated in Fig. 9 the best dataset for PyTorch's training process is the one created with the alpha-trimmed mean algorithm. Once again, this result arises because of the nature of the algorithms used:

1. alpha-trimmed mean (alpha=8), the best one with an accuracy of 75.69%.
2. median, with an accuracy of 73.17%.
3. percentile, the one of the original paper (Mitra et al. 2019), with an accuracy of 69.47% is once again the worst dataset created based on accuracy score in PyTorch's training process.

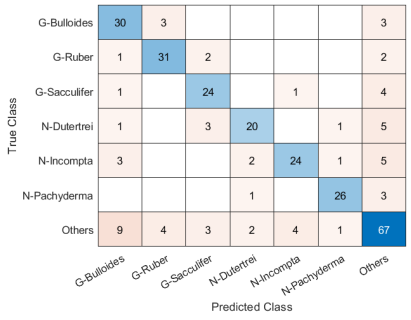
This time we observe a much larger variance in the accuracy (∼2,25% in Matlab, ∼6,25% in PyTorch), and since the training samples are the same for both methods, there will be differences in the training process between Matlab and PyTorch that allow the former to better "understand" the peculiarities of the different specie.



(a)



(b)



(c)

Figure 6: Heatmaps of Matlab percentile (6a), Matlab median (6b), Matlab colored alpha (6c)

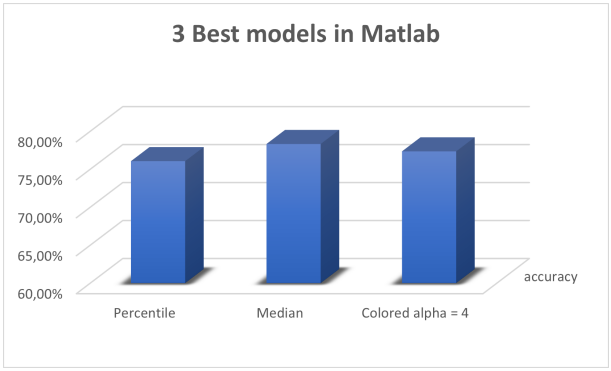
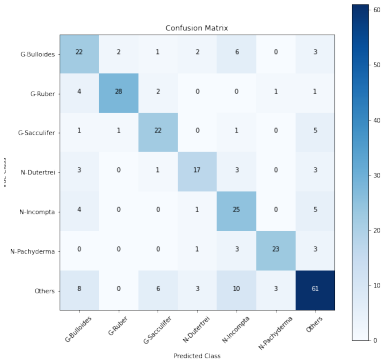
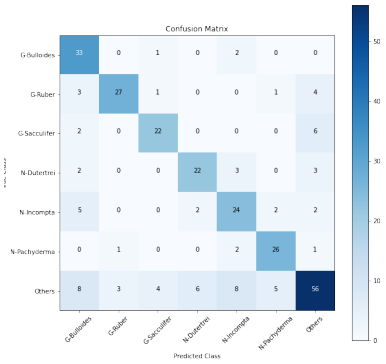


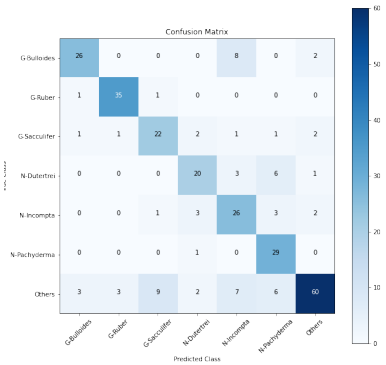
Figure 7: Comparison between top 3 best models in Matlab



(a)



(b)



(c)

Figure 8: Heatmaps of PyTorch percentile (8a), PyTorch median (8b), PyTorch colored alpha (8c)

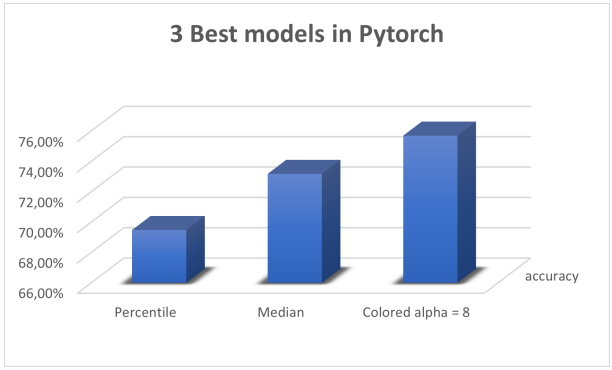


Figure 9: Comparison between top 3 best models in PyTorch

Conclusion

Through our efforts, we have successfully attained our objective by surpassing the performance benchmarks of the original project. This accomplishment was realized through rigorous training and testing within a consistent environment. In addition to our achievement, we personally encountered the intricacies of working across diverse environments and grappling with the complexities of result comparison. This hands-on experience served as a catalyst for our decision to delve deeper into the study, aiming to uncover the underlying reasons behind the disparities in results between the models created in Matlab and PyTorch.

Undoubtedly, the framework within which we operate holds significant importance, as we can see in the results. This becomes particularly apparent when we confront the challenge of making accurate comparisons among models, even those trained on the same dataset, but within different environments. This intricacy highlights the nuanced nature of this type of job and underscores the necessity for comprehensive analysis and understanding as we proceed with our research.

References

Mitra, R.; Marchitto, T.; Ge, Q.; Zhong, B.; Kanakiya, B.; Cook, M.; Fehrenbacher, J.; Ortiz, J.; Tripati, A.; and Lobaton, E. 2019. Automated species-level identification of planktic foraminifera using convolutional neural networks, with comparison to human performance. *Marine Micropaleontology*, 147: 16–24.

Zhang, R.; Isola, P.; and Efros, A. A. 2016. Colorful Image Colorization. In *ECCV*.