UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"CONVOLUTIONAL NEURAL NETWORK ENSEMBLE LEARNING FOR FORAMINIFERA CLASSIFICATION"

Relatore: Prof. / Dott. NANNI LORIS
Laureando: Vallazza Miro

ANNO ACCADEMICO 2021 – 2022
Data di laurea 14/03/2022

Table of contents

1. Introduction

Image classification is a non-creative and repetitive process, as such, it is well-suited for automation. Neural Networks (NNs) and more specifically Convolutional Neural Networks (CNN) are Deep Learning models used in such task. The stochastic nature of NNs, however, leads to results that are influenced by a fair share of randomness. While studying a specific subject we could need a more consistent method of classification, Ensemble Learning (EL) is a widely used implementation that combines results from different networks in order to achieve better performance.

The goal of this thesis is to study a practical application of EL, to do that we will work on the classification problem presented in the research paper [1]. The subject of the article is foraminifera classification, an invaluable task for industrial and academic purposes. Species of foraminifera are used as paleo-environmental indicators, while their radiocarbon measurement are used to infer parameters like global ice volume, salinity, PH, and more. Classification is performed by humans, usually on a range of 500-1000 individuals per sample, a very repetitive and time-consuming task. Automated solutions that help to alleviate such an unrewarding assignment have been implemented since the early 90, but they usually required strong human supervision. In 2004 the neural network SYRACO2 automatically identified single-celled organisms quite reliably, and in 2017 CNNs have been applied to diatom identification with great success [1].

The authors (of [1]) describe their attempt of automating foraminifera classification. They built a dataset of six species of foraminifera and a collection of other taxa and used two pre-trained CNNs (ResNet50 and Vgg16) to build their model. They also compared their results with those of human experts and novices. Working with the same dataset, our goal is to observe the difference in terms of performance while using multiple CNNs and pre-processing approaches in an EL environment.

2. Convolutional Neural Networks

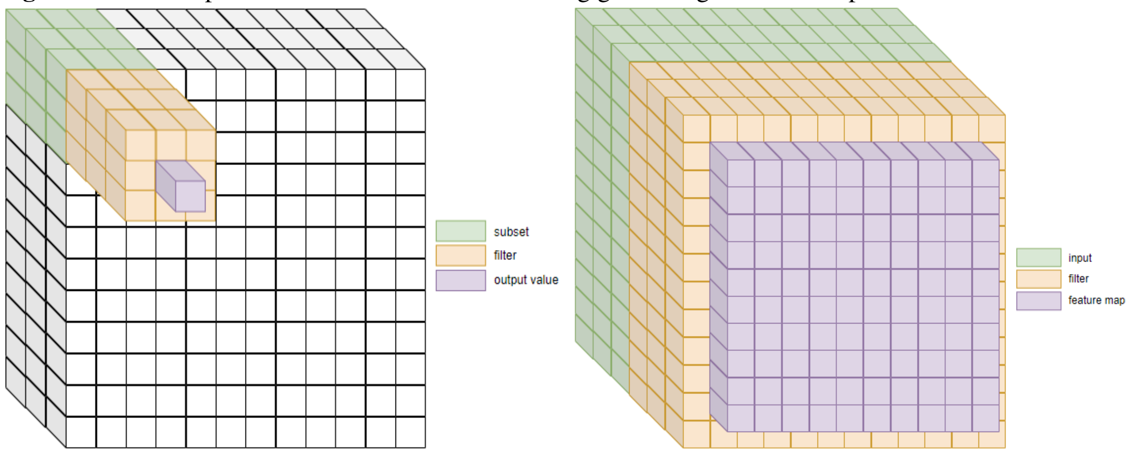
CNNs were first introduced in the 1980s by the French researcher Yann LeCun and already achieved good performances by the late 1990s. In the last decade, however, due to the advent of big data and GPU computing, they reached the status of state-of-the-art technology in the field of computer vision and image recognition.

As every other type of neural network, CNNs' structure is divided in three components: the input layer, which is usually a volume of NxNx3 neurons, and is directly connected to the image's pixels, hidden layers that utilize shared weights and local connections, and a fully connected output layer.

Local connections and shared weights are the main deviations from a normal multilayer perceptron (MLP) network. Neurons of the hidden layers are only connected locally to the adjacent neurons, which means that they will only process information from a subset of the input. Furthermore, weights and biases are shared in groups, in order to interpret information gained in different portions of the input in a consistent manner.

Convolution utilizes a digital filter (or mask) to extract data from a subset of the input**.** The result of filtering the whole input volume, by making the mask slide in every possible position is a feature map [10].

Figure 1. Visual representation of convolution filtering generating a feature map



Pooling is another filtering method that aggregates portions of an input feature map and is used to reduce variance between small transformations of the input [10]. Two of the most utilized methods are max and average pooling, in which, given a portion of the input only the maximum or average value is extracted.

CNNs implement pooling and convolution directly through its architecture. Local connections, non-linear activation functions and shared weights are used to build feature maps that autonomously create the filters needed.

The output layer of a CNN is a fully connected one, it utilizes a neuron for each class, and in modern models they utilize a SoftMax activation function:

$$f(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.1}$$

With n, the number of classes and z the input vector. The output of the function is a normalized value $0 \leq b \leq 1$ and can be interpreted as confidence.

3. CNN Ensemble Learning

The theory behind Ensemble Learning is simple, by combining different models it is possible to produce better and more reliable results. Ensemble can be performed on four different levels:

- Data: by splitting the dataset in different subsets
- Feature: by pre-processing the dataset with unique methods
- Classifier: by training different classifiers on the same dataset
- Decision: by combining the decisions of multiple models

Ensemble works best when applied to significantly diverse models [8]. We will be operating on the last three levels, by creating multiple datasets (applying different method for representing the input images as RGB images) to train multiple networks and ultimately combining the scores using the sum rule.

3.1 Image Pre-Processing

Each image in the dataset comes in the form of 16 grayscale pictures of the same foraminifera subject, each taken under different lighting conditions, using a AmScope SE305R-PZ binocular microscope at 30× magnification [1]. The dataset is divided in seven classes: one for each of the six species of foraminifera we are going to study: Bulloides, Ruber, Sacculifer, Dutertrei, Incompta, Pachyderma, each having ~ 150 entries, and a “rest of the world” (other taxa) class that counts 450 entries. Since we use pre-trained networks[1] and their input is expected to be an RGB image, the first step in our pipeline will be generating CNN-compatible inputs starting from the dataset.

[1] See section 4.

Multiple ways for completing this task were tested, and, in the end, we settled on a combination of four:

1. The “Percentile” method presented in the research paper
2. Discrete Cosine Transform (DCT)
3. Principal Component Analysis (PCA)
4. Multidimensional Scaling (MDS)

3.1.1 Percentile

This is the most straight forward method of generating the RGB image, it was directly taken from the research paper [1]: “A Python code takes the 16 individual grayscale values for each pixel, calculates the 10th percentile, median, and 90th percentile, and maps those three values into the red, green, and blue channels, to generate a single colour composite image”.
To speed up the process we used the nearest rank method: the list of 16 grayscale values is sorted in ascending order then the P-th percentile element is selected by extracting the value of cell *n*, where

$$n = \left\lceil \frac{P}{100} \cdot 16 \right\rceil \qquad (3.1.1.1)$$

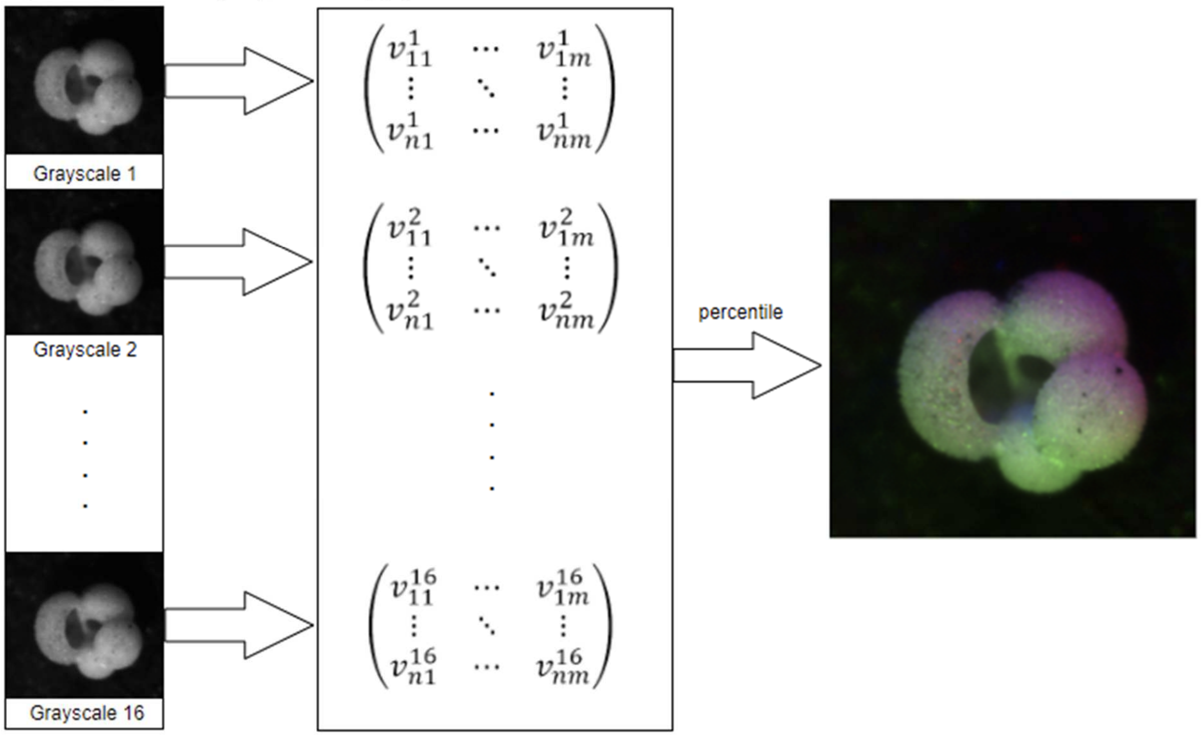
Since our input vectors are always the same size, after the 16 values are sorted the three values of *n* we will use are: 2, 8 and 15.

The percentile pre-processing pipeline goes as follows:

- Read the 16 images
- Populate a NxMx16 matrix with the grayscale values
- For each pixel extract its 16 grayscale values in a list

- Sort the list
- Use elements 2, 8 and 15 as RGB values in the new image

Figure 2, percentile pre-processing pipeline


Grayscale 1
Grayscale 2
Grayscale 16
$\begin{pmatrix} v_{11}^1 & \cdots & v_{1m}^1 \\ \vdots & \ddots & \vdots \\ v_{n1}^1 & \cdots & v_{nm}^1 \end{pmatrix}$
$\begin{pmatrix} v_{11}^2 & \cdots & v_{1m}^2 \\ \vdots & \ddots & \vdots \\ v_{n1}^2 & \cdots & v_{nm}^2 \end{pmatrix}$
$\begin{pmatrix} v_{11}^{16} & \cdots & v_{1m}^{16} \\ \vdots & \ddots & \vdots \\ v_{n1}^{16} & \cdots & v_{nm}^{16} \end{pmatrix}$
percentile

A few variations were tested, (20th/80th/median for example), but they were all discarded due to their performance being slightly worse than the original, losing out on 1-2% of accuracy per single-run training cycle on average.

3.1.2 Discrete Cosine Transform

DCT-II is the most popular function for spatial compression. It decomposes a signal into a sum of cosine functions through the formula:

$$X_k = \sqrt{\frac{2}{N}} \sum_{n=1}^{N} \frac{x_n}{\sqrt{1+\delta_{k1}}} \cos\left[\frac{\pi}{2N}(k-1)(2n-1)\right] \qquad (3.1.2.1)$$

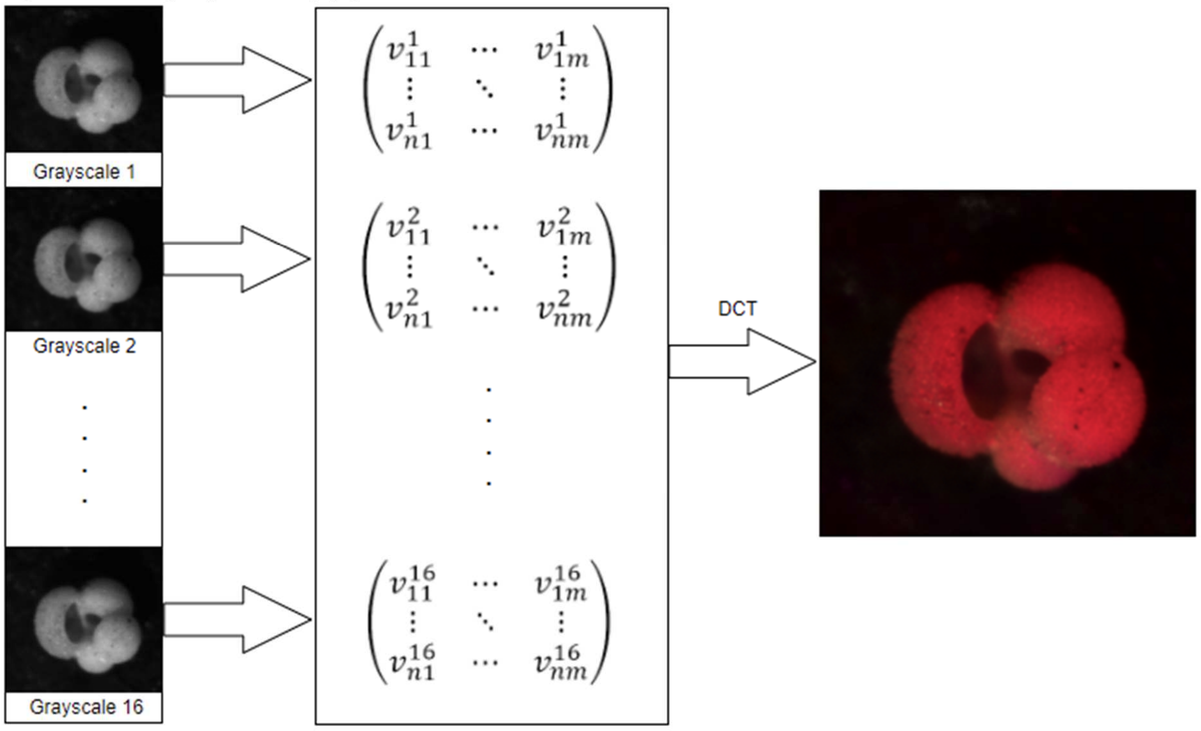
Where the $x_1 \mathinner{..} x_N$ datapoints are transformed into the $X_1 \mathinner{..} X_N$ real values and the Kronecker delta δ_{k1} is defined as:

$$\delta_{k1} = \begin{cases} 0 \; if \; k \neq 1 \\ 1 \; if \; k = 1 \end{cases} \tag{3.1.2.2}$$

We can apply DCT to the sixteen values of each pixel, then map [X_1, X_2, X_3] into the RGB channels to build the new image. Similarly to the percentile pre-processing pipeline, DCT was applied this way:

- Read the 16 images
- Populate a NxMx16 matrix with the grayscale values
- For each pixel extract its 16 grayscale values in a list
- Compute DCT on the list
- Use elements 1, 2 and 3 as RGB valeus in the new image

Figure 3, DCT pre-processing pipeline


Grayscale 1
Grayscale 2
Grayscale 16
$\begin{pmatrix} v_{11}^{1} & \cdots & v_{1m}^{1} \\ \vdots & \ddots & \vdots \\ v_{n1}^{1} & \cdots & v_{nm}^{1} \end{pmatrix}$
$\begin{pmatrix} v_{11}^{2} & \cdots & v_{1m}^{2} \\ \vdots & \ddots & \vdots \\ v_{n1}^{2} & \cdots & v_{nm}^{2} \end{pmatrix}$
$\begin{pmatrix} v_{11}^{16} & \cdots & v_{1m}^{16} \\ \vdots & \ddots & \vdots \\ v_{n1}^{16} & \cdots & v_{nm}^{16} \end{pmatrix}$
DCT

3.1.3 Principal Component Analysis

PCA is a data analysis tool, often used for dimensionality reduction. Starting from a real multidimensional data collection, whose domain is R^n, we can use PCA to build a new one, directly mapped through projection on a smaller space: R^m, with $m < n$. The basis of the new space is obtained by calculating the principal components.

The principal components of a n-dimensional data collection are a set of n orthogonal unit vectors, each representing the direction of the line that minimizes the average squared distance from the data points. They can be extracted using different methods, the most common is the covariance eigen decomposition [7].

The n-dimensional data is organized in a $n \times k$ matrix, $M_{n \times k}$, each of the n rows represents a variable, and the values stored in the columns are the k readings for each variable.

The first step of the decomposition is data standardization, achieved by simply subtracting the mean vector to each row, as in 3.1.3.1.

$$X_i' = X_i - \bar{X}_i \tag{3.1.3.1}$$

With X_i the ith row and $\bar{X}_i$ the mean vector of the same row. The new row vectors populate the standardized matrix $M'_{n \times k}$.

The second step consists in computing the covariance matrix, $C_{n \times n}$, of the standardized data:

$$C = \frac{1}{n} M' \, M'^T \tag{3.1.3.2}$$

At this point we can extract the eigenvectors from C, and, after sorting them in order of descending eigenvalue we can choose the first m vectors as our new basis, building the projection matrix $P_{m\times n}$. Lastly, we can project the initial data in the new space:

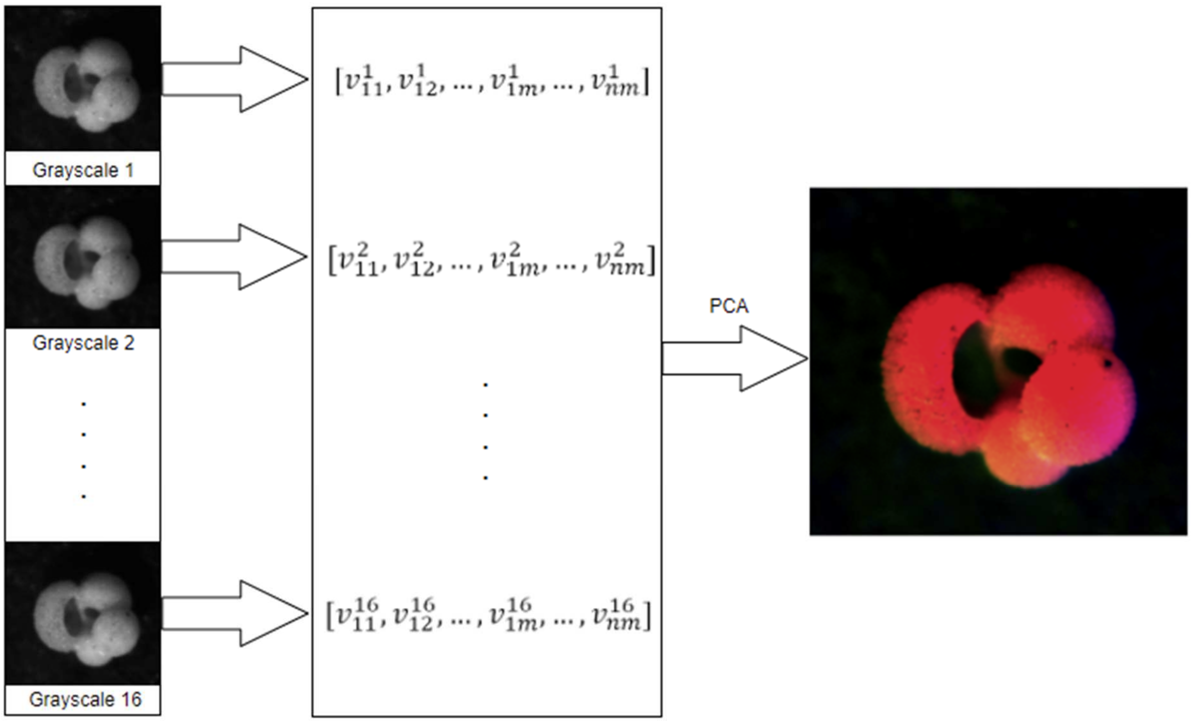
$$G = P_{m\times n} M_{n\times k} \tag{3.1.3.3}$$

Each of the 16 grayscale images is turned into a 1-dimensional vector of length *NM*, by simply concatenating its rows, then, these arrays are organized into a *16*×*NM* matrix, applying PCA to it will return a *3*×*NM* matrix, restructuring it into the *N*×*M*×*3* format will give us the RGB image.

The PCA pre-processing pipeline goes as follows:

- Read the 16 images
- Rearrange each image in a vector
- Populate a 16xNM matrix with the vectors obtained
- Compute PCA
- Restructure the resulting matrix in the NxMx3 format

Figure 4. PCA pre-processing pipeline


Grayscale 1
Grayscale 2
Grayscale 16
$[v_{11}^1, v_{12}^1, \ldots, v_{1m}^1, \ldots, v_{nm}^1]$
$[v_{11}^2, v_{12}^2, \ldots, v_{1m}^2, \ldots, v_{nm}^2]$
$[v_{11}^{16}, v_{12}^{16}, \ldots, v_{1m}^{16}, \ldots, v_{nm}^{16}]$
PCA

3.1.4. Multidimensional scaling

Multidimensional scaling is a dimensionality reduction method, its classical utilization is also referred as Principal coordinates analysis. Starting from an n-dimensional dataset of m datapoints the first step of the MDS algorithm is to compute the dissimilarity matrix D_{mxm}, where each of its values d_{ij} is given by computing the Euclidean distance between the datapoints X_i and X_j. The end goal is to find m vectors of length $k < n$, that best fit the condition:

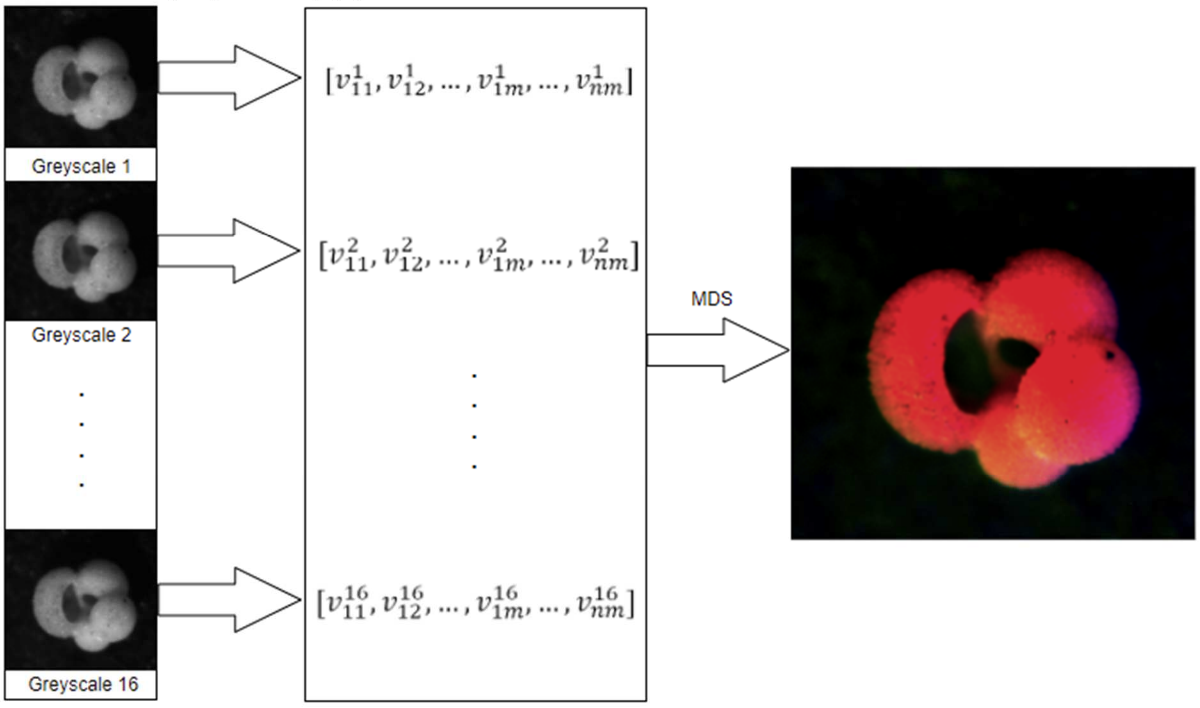
$$|x_i - x_j| \cong d_{ij} \, \forall \, i, j \qquad (3.1.4.1)$$

This can be achieved through eigen decomposition, for example, but there are multiple implementations that utilize operational research or stress functions [6].

The approach used for arranging the data and calculating the final image is the same used for PCA in 3.1.3:

- Read the 16 images
- Rearrange each image in a vector
- Populate a 16xNM matrix with the vectors obtained
- Compute MDS
- Restructure the resulting matrix in the NxMx3 format

Figure 5. MDS pre-processing pipeline


Greyscale 1
Greyscale 2
Greyscale 16
$[v_{11}^{1}, v_{12}^{1}, \dots, v_{1m}^{1}, \dots, v_{nm}^{1}]$
$[v_{11}^{2}, v_{12}^{2}, \dots, v_{1m}^{2}, \dots, v_{nm}^{2}]$
$[v_{11}^{16}, v_{12}^{16}, \dots, v_{1m}^{16}, \dots, v_{nm}^{16}]$
MDS

3.2 Training

Once the dataset is built, we can start training our networks. We will be using the pre-trained CNNs:

- **GoogLeNet**, a 22 layers deep CNN, the first to implement inception layers, a combination of parallel, small sized convolutional layers. The input layer is a 224x224x3 zero-pad layer and the output presents 1000 fully connected neurons [3].
- **ResNet50**, a 50 layers deep CNN, the input layer is a 224×224×3 zero-pad layer, it has 48 hidden layers, two of which are a max and an average pool respectively. Networks

of the ResNet family utilize residual blocks to maximize depth while diminishing the number of parameters [5]. The output layer is a 1000 neurons SoftMax layer.

- **ResNet101**, a deeper version of ResNet50, it has 101 layers in total, but still maintains the two max and average pooling layers [5].
- **MobileNet v2**, a very efficient 53 layers deep network, originally built to run on mobile devices. As the ResNet models it utilizes residual blocks [2].
- **EfficientNetb0**, a residual block based network, it scales its width and depth uniformly through a compound coefficient that is linked to the computational resources available [4].

All networks were pre-trained with an open-source image dataset called ImageNet, which contains millions of labelled images. Hyperparameters for all networks where set as:

- Mini Batch Size: 30
- Max Epochs: 20
- Learning Rate: 10^{-3}

These networks were chosen after a long testing period and were selected after considering their performance and duration of training cycle. Running on an AMD R5 3600 processor, an RTX 2060 super GPU and 16 GB of RAM.

3.2.1. Transfer Learning

The main benefit of using pre-trained networks is transfer learning. Modern neural networks can carry over some of the knowledge gained in previous training cycles to perform the same task on different data [9]. This occurs in deep learning models especially since they operate on wide array of weights and features. Transfer learning has the effect of greatly reducing the number of images we will need to train the networks.

We of course need to adapt the output layer to specialise the networks. All the output layers of the networks we are using will be replaced with fully connected SoftMax layers of 7 neurons, one for each class of foraminifera available in the dataset.

Images of the dataset will be resized in order to match the dimensions of the input layers.

By replacing the last layer with a 7-neurons SoftMax layer the outputs will come in the form of a vector v, whose values (scores) are:

$$\begin{cases} 0 \leq v_i \leq 1 \; \forall \, i \\ \sum_{i=1}^{7} v_i = 1 \end{cases} \tag{3.2.1.1}$$

Scores reflect the level of confidence with which the networks classify an input.

By diversifying the models, the information processed differs by each of them. Scores fusion is a method that combines confidence values to build a more robust prediction. The fusion technique we used is the sum rule [8]:

$$sum = \sum_{i=1}^{N} v \quad out = argmax \left\{sum_j\right\}, j = \; 1 \dots n \tag{3.2.1.2}$$

Where N is the number of models and n the size of each confidence vector v. The sum rule is one of the best fusion methods in practice since it does not suffer from potentially destructive operations like, for example, multiplications by zero.

3.2.4. Scores Collection and Metrics

Data used to produce the final results was collected during the training cycles of each network utilizing the scores of the test-set of each run. Our training pipeline goes as follows:

- Networks are trained independently on multiple datasets, for multiple iterations, using 4-fold cross validation.
- The scores are extracted from the tests
- The scores are fused using the sum rule
- F-Score is computed

We will use F-score (F_1) as the metric to evaluate our model

$$F_1 = \frac{T_p}{T_p + \frac{1}{2}(F_p + F_n)} \tag{3.2.4.1}$$

Where T_p, F_p, F_n are the total number of True positive, and False positive/negative predictions made by the model.

The code was written and ran in a Matlab environment, utilizing the DeepLearning Toolbox.

4. Results

F_1 measures in Table 1 come from single run 4-Fold cross validation training cycles of each network. PCA and MDS did not perform well enough in general and were therefore discarded in order to save time during the tests.

Table 1. F1 scores of single run training cycles.

	ResNet50	GoogleNet	MobileNet	EffNet	ResNet101
Percentile	0.833	0.764	0.860	0.795	0.832
PCA	0.742	0.667	---	---	---
DCT	0.818	0.691	0.748	0.673	0.794
MDS	0.722	0.635	---	---	---

Table 2 contains the F-score values given by the decision fusion of the same classifier trained independently for multiple iterations, using 4-Fold cross validation.

Table 2, Percentile(6) means that each model was trained 6 times independently with the percentile dataset, the final F-Score values are calculated after the scores are fused using sum rule. Percentile(3)+DCT(3) works the same way, but in this case 3 classifiers are trained with the DCT and 3 with the percentile datasets.

	ResNet50	GoogleNet	MobileNet	EffNet	ResNet101
Percentile (6)	0.8786	0.836	0.885	0.828	0.8786
Percentile(3)+DCT(3)	0.8817	0.830	0.882	0.838	0.8734

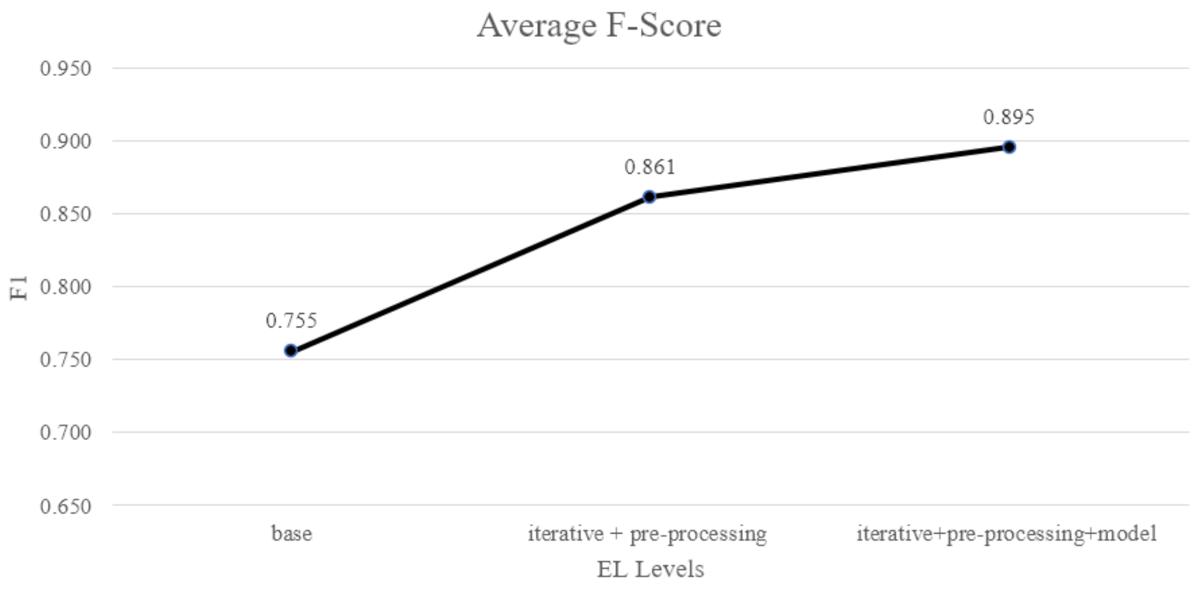
Scores are then combined to calculate the final F-Score results, Table 3.

Table 3, Fusion of F1 scores gained as in **Table 2**, and sum rule with different CNNs.

	ResNet50 +GoogLeNet	ResNet50 +GoogLeNet +MobileNet	ResNet50 +GoogLeNet +MobileNet +Efficientnet	ResNet50 +GoogLeNet +MobileNet +Efficientnet +ResNet101
Percentile (6)	0.8785	0.8956	0.9000	0.8986
Percentile(3)+DCT(3)	0.8866	0.9037	0.9015	0.8982

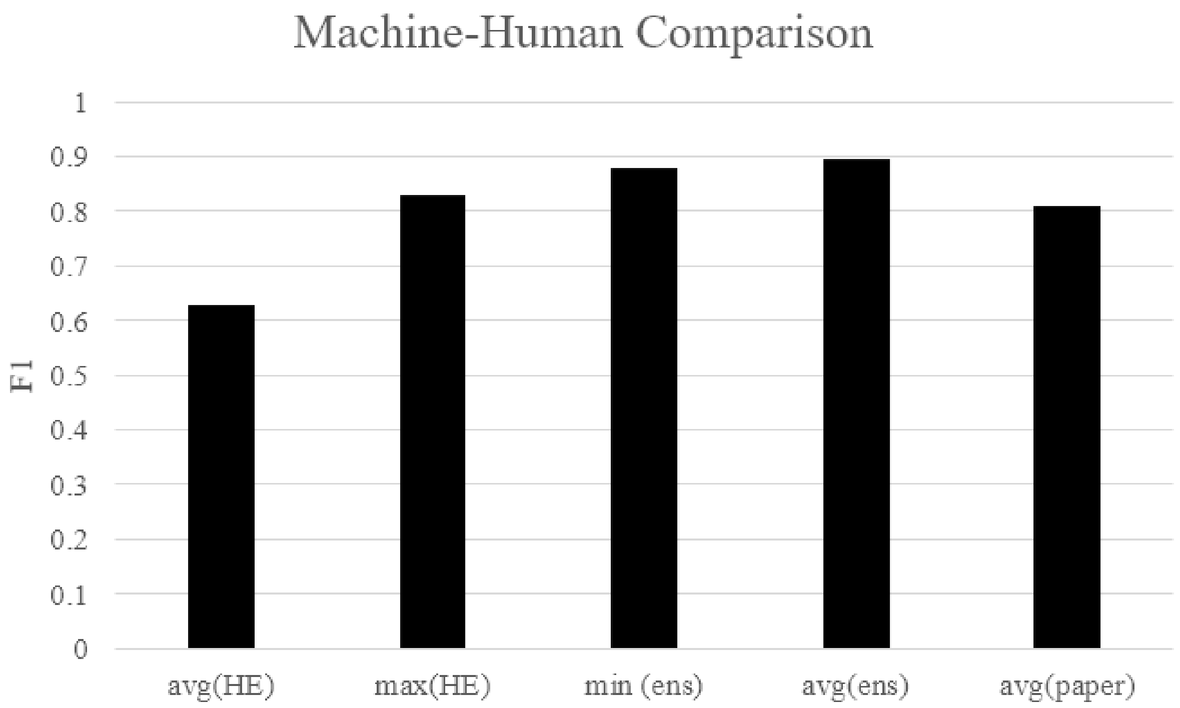
F-Score results get strictly better as Ensemble is applied to more levels.

Figure 6, the effect of EL being applied to more levels



Both minimum and average performance after Ensemble show better results than human experts, Figure 7.

Figure 7, human expert (HE) and automated models comparison, ens refers to out solution, paper to [1].



It is important to know that the paper’s networks were trained using 62% of the images as training set, while ours utilized 75%, however, results presented in the paper for the same partition of the dataset show that our solution still performs better on average ($\sim + 5\%$).

5. Conclusions

Although the sample size of the experiment is small, the impact of Ensemble Learning on the problem is definitely noticeable. We managed to bring a ~75% average F-Score on baseline single-trained networks, to a ~89% average F_1 result in our final models.

There are, however, a few points of concern. The dataset only contained ~1500 images and 7 classes of foraminifera, the scope of the experience was therefore limited to a very small portion of the real-world problem. We can safely assume that with more samples per class, performance could be enhanced even further. What we don't know, without experimental data, is how the model would respond to more classes being implemented. Image pre-processing and classification on multiple CNNs are very time-consuming tasks, since the model was running on a sub-optimal machine, our focus was directed towards maximizing accuracy, while neglecting the process' duration.

The fact that the model outperformed human experts is still an impressive feat, especially if we consider the fact that both PCA and MDS did not work as well as we hoped they would, leaving us with effectively only two pre-processing techniques to work with.

Abbreviations

NN	Neural Network
CNN	Convolutional Neural Network
EL	Ensemble Learning
GPU	Graphics Processing Unit
MLP	Multilayer Perceptron
PCA	Principal Component Analysis
MDS	Multidimensional Scaling
DCT	Discrete Cosine Transform

References

[1] R. Mitra, T.M. Marchitto, Q. Ge, B. Zhong, B. Kanakiya, M.S. Cook, J.S. Fehrenbacher, J.D. Ortiz, A. Tripati, E. Lobaton, Automated species-level identification of planktic foraminifera using convolutional neural networks, with comparison to human performance.

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, Google Inc, MobileNetV2: Inverted Residuals and Linear Bottlenecks

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Going Deeper with Convolutions

[4] Mingxing Tan, Quoc V. Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition

[6] Laurens van der Maaten, Eric Postma, H. Jaap Van Den Herik, Dimensionality Reduction: A Comparative Review

[7] Lindsay I Smith, A tutorial on Principal Components Analysis

[8] Kuncheva L.I. Combining Pattern Classifiers. Methods and Algorithms, Wiley, 2nd edition, 2014.

[9] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, Qing He, A Comprehensive Survey on Transfer Learning.

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning