MASTER IN HIGH PERFORMANCE COMPUTING

Feature Learning and Clustering Analysis for Images Classification

Supervisor:
Stefano COZZINI

Candidate:
Piero CORONICA

4th EDITION
2017–2018

Abstract

The problem this thesis is addressing is to improve an existing classification in 10 categories of the images captured by SEM microscopes. In particular, the challenge faced is to classify those images according to a hierarchical tree structure of sub-categories without requiring any further human labelling effort. In order to uncover intrinsic structures among the images, a procedure involving supervised and unsupervised feature learning, as well as cluster analysis is defined. Moreover, to reduce the bias introduced in the supervised phase, various strategies focusing on features of different nature and level of abstraction are analyzed.

Summary

Chapter 1

Introduction

In the framework of the NFFA-EUROPE project, the Information and Data management Repository Platform (IDRP) is being developed to answer to the data sharing needs of the nanoscience community. The aim of the infrastructure is to allow users to access, share and publish richly diverse data produced by individual scientists. A central problem is represented by guaranteeing the accessibility, or better the searchability, of the data in such heterogeneous dataset. Thus, in order to organize the repository, it is necessary to automatically enrich each data with appropriate meta-data defining its content.

Focusing on the data produced by a single instrument, a Scanning Electron Microscope (SEM), almost 150 thousands images were collected. From this first set, headed to increase in the near future, a sample counting almost 18 thousands images was labelled by hand in 10 categories [1] forming the `SEM_Dataset`. In previous works ([2], and [3]) the `SEM_Dataset` was used to train various Convolutional Neural Networks. The resulting models were able to classify images captured by a SEM in the 10 categories with astonishing accuracy (reaching, in some cases, almost 98% of test accuracy).

Despite the effort to make the repository searchable, a division in 10 categories does not provide a sufficient fine partition of the data. The problem this thesis is addressing is to improve the existing categories providing a hierarchical tree of sub-categories. In opposition to what has been done before, the thesis faces the challenge of avoiding the huge human effort required to classify by hand a training set, by automatically finding hidden structures in the data via unsupervised machine learning methods.

Outline of the thesis

The rich diversity of the data considered required some pre-processing cleaning operations described in chapter 2. Moreover, the problematic related to the huge scale variations between the images is discussed. The final part of the chapter introduces the reader to the dataset `1u-2u` used in the rest of the thesis. As reported in table 2.5, it counts 52 682 images at the scale of one and two microns and the 14% of this dataset inherits the classification labels from `SEM_Dataset`.

In order to find structures within the data in `1u-2u`, a notion of similarity between images in required. Chapter 3 explores a possible way to solve this problem via feature learning. The idea is that, instead of comparing directly the images, the similarity should take into account the high level features of the objects they represent. Unfortunately, unsupervised feature learning methods in such high dimensional spaces are heavily affected by the so-called curse of dimensionality. Therefore the feature learning procedure is carried out firstly in a supervised way by using a Convolutional Neural Network, then in an unsupervised way via autoencoders. At the end of the chapter four distances on `1u-2u` are defined and analyzed. These differ according to the type of supervision and level of abstraction considered in the feature learning procedure.

In chapter 4 a first cluster analysis on `1u-2u` is performed. After introducing the clustering algorithm used, a criterion to score their solutions against the labels provided by `SEM_Dataset` is discussed. Finally, the scores obtained by applying the various algorithms on `1u-2u` are analyzed for each one of the distances defined in the previous chapter.

Acknowledgement

The activity that led to this thesis is developed as part of the NFFA-EUROPE project and is funded by CNR-IOM. I personally thank Dr. Stefano Cozzini for giving me the opportunity to explore the wonderful topics faced in this MHPC thesis and all eXact Lab staff for providing the expertise and the technical support needed developing this project.

I owe my deep gratitude to Alex Rodriguez for the enthusiasm shown during the very instructive discussions we had, and to Giangiacomo Sanna for sharing his pearls of wisdom with me.

Finally, I want to thank the wonderful Emilia, my family, Luca and all the friends of the master (with a special mention to Classico mates). Without you all the last year would had been much much harder.

Chapter 2

The dataset

The data analyzed in the project are images captured via a scanning electron microscope (SEM) of numerous objects at various scales and stored using different formats and meta-data. The purpose of this chapter is to illustrate some insights obtained exploring these data.

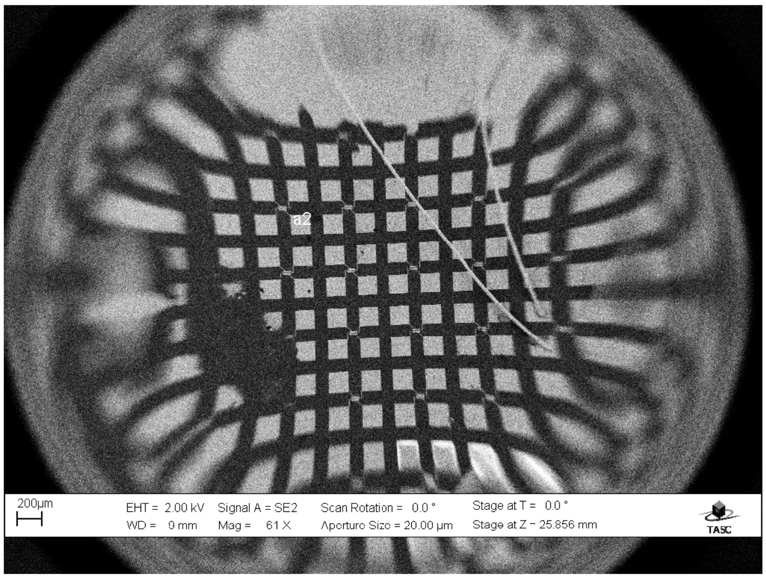

Figure 2.1: A fancy image in `images_flat`

The first step to perform statistical analysis on the images was to create a uniform dataset containing files of the same format, named after a unique and anonymized identifier. This procedure, described in section 2.1, produced a dataset called `images_flat` containing 146 917 images each one stored in two formats: TIFF and JPG. Some thousands of those images were classified by hand in 10 categories and used in previous works (see [1],[3], and [2]). Section 2.2 explains how to retrieve the information about the labelled images and how the labels are distributed on the new dataset. Section 2.3 faces a distinct problem: different features have different relative sizes; it makes sense to explore images representing objects of the same size. Unfortunately, the scale of the images is not stored as a meta-data of the TIFF files but appears written on the very images. The implementation of a simple OCR program, its accuracy, and its output are presented.

At the end of the chapter the dataset `1u-2u` used in the rest of the thesis is defined. It contains only the images from `images_flat` having scale 1 µm and 2 µm and its details are reported in table 2.5.

2.1 Dataset preparation: from `new_dati_SEM` to `images_flat`

The images used in the thesis were collected by a multitude of scientists. As a result, the directory (`new_dati_SEM/`) where those images were collected is a jumble of nested sub-directories with puzzling names and without a clear hierarchy. Worst of all, it contains duplicates, corrupted files, and images stored using different formats and, where present, meta-data.

format	**occurencies**
TIFF	151 312
JPG	150 937
PDF	125
BMP	111
DM3	2
DJVU	1

Table 2.1: Image file formats in `new_dati_SEM/`

As shown in table 2.1, the most frequent image file formats appearing in `new_dati_SEM/` are JPG and TIFF. An analysis of the MD5 hashes of the TIFF files detected 148 918 unique binary contents. Moreover, converting the images from TIFF to JPG using various compression methods, it was possible to replicate almost every JPG from a TIFF file. For this reason the focus was placed on the TIFF files only.

For each unique MD5 hash one of the associated TIFF files has been copied into the folder `images_flat`, renamed after the hash, and converted to JPG. Note that the uniqueness of the hashes entails the uniqueness on the binary content of the TIFF files and not on their visual content, thus there could be different TIFFs that produce the same JPG (e.g. files representing the same image but with incongruities in the meta-data). In order to guarantee also the uniqueness of the MD5 hashes of the JPG files, other 2001 pairs of files were removed. Surely this selection discarded some data but has the virtue of removing most duplicate images, creating two uniform set of files in TIFF and JPG format in a one to one relation by their anonymized basename (MD5 hash on the TIFF file), and assuring the absence of corrupted files.

After this cleaning operation `images_flat` counts 146 916 pairs of files in TIFF and JPG format, both uniquely identified by their content-based MD5 hash[1].

2.2 The labelled images

Tens of thousands of images among the ones in `new_dati_SEM` were classified by hand. The different categories are based on the shape and on the structure of the represented objects. Figure 2.2 lists the 10 categories that were mainly used providing for each category a representative image.

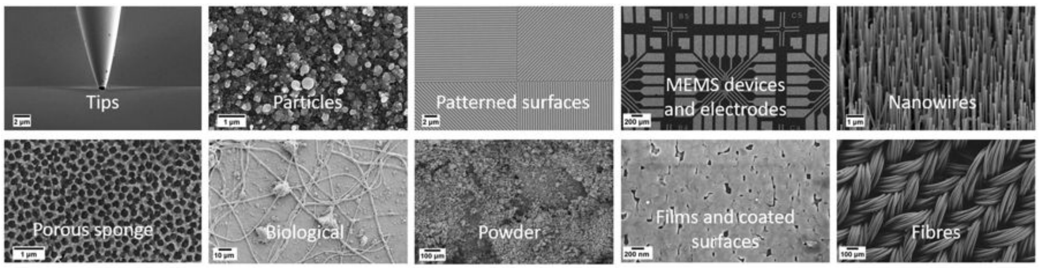

Figure 2.2: Representative images of the 10 categories

From this huge work, 4 datasets were created and published (see [4]). They differ for size (as shown in table 2.2), sampling criterion, and purpose. The original dataset used in [1] is called `SEM_Dataset` [5]. The `Hierarchical` [6] dataset is very small, exceeding slightly the thousand images, but it has

[1]It is worthy to stress again that unique hashes do not necessarily correspond to unique images. Think about images differing in resolution or, more subtly, having some pixels with slightly different brightness (this situation was spotted in `images_flat`: in c3790570d62f15c2b0778eeb63d45426.jpg and 536b95a78bb3a5643208066d6cccc4e8.jpg the brightness values of 33 pixels differ by 1).

a tree structure that splits the original 10 categories in a total of 36 subcategories. The last two datasets (`100%` [7] and `Majority` [8]) are the results of a validation procedure on a larger dataset.

In order to identify the images in `images_flat` that were classified, the JPGs in it were compared to the files in the 4 datasets searching for matches in the MD5 hashes. Table 2.2 summarizes the results of the pairings and provide some information about those datasets: the actual number of categories considered, the total JPG files they contain, the amount of unique hashes on the content of these files (i.e. the number of images up to duplicates), and the quantity of the hashes for which was not possible to find a direct match within `images_flat`.

Name	Num. categories	Num. JPGs	Num. unique hashes	no matches in `images_flat`
`SEM_Dataset`	10	18 577	18 343	85
`Hierarchical`	36	1038	1026	0
`100%`	10	25 430	20 837	183
`Majority`	10	25 537	20 937	237

Table 2.2: Comparison of the various `SEM_Dataset`

Note that in `100%` and `Majority`, there is a big discrepancy between the number of files and the number of unique hashes due to numerous duplicates. More importantly, it is not guaranteed that duplicate files belong to the same category; indeed, there are respectively 4109 and 4113 hashes corresponding to duplicate files labelled in different categories (for a total of 8370 and 8378 files). Although the problem is limited to two categories (i.e. `Patterned_surface` and `MEMS_devices_and_electrodes`), it affects the 25% of the unique contents of those datasets and thus it undermines the reliability of the information they contain.

Focusing on the original dataset, it was not possible to find a direct match for 85 images; nevertheless, 3 of them where matched using a different compression in the conversion from TIFF to JPG. The information about the remaining 82 images were left out: they do not correspond to any TIFF file in `images_flat` because originally stored in JPG format only. Figure 2.3 shows the distribution of the 10 categories on the 18 261 labelled images in `images_flat` and on the 82 missing data from `SEM_Dataset`.

The bar chart highlights the imbalance of the labels. The 4 major categories represent almost the 80% of the total while the three minor categories

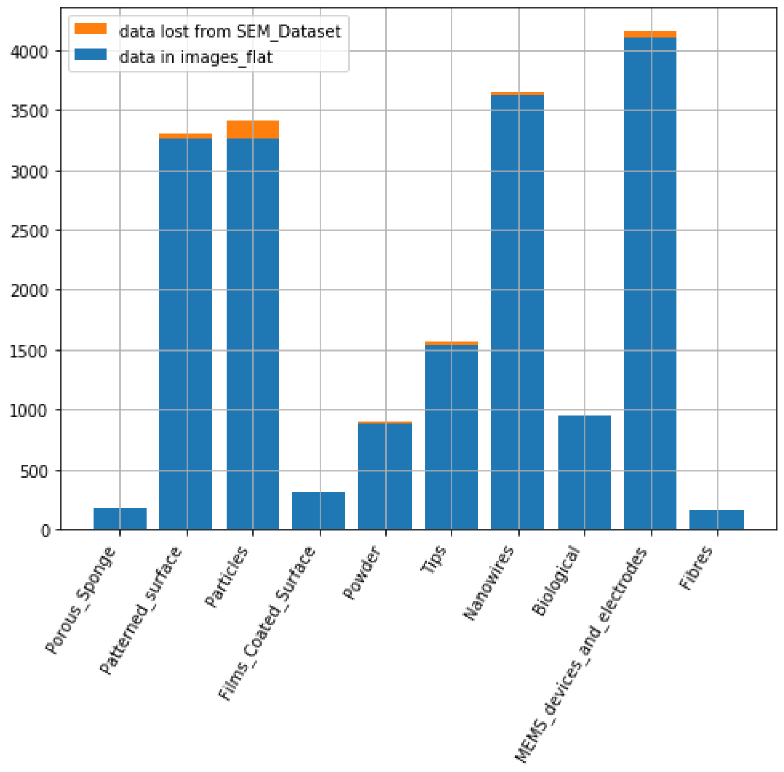

Figure 2.3: Distribution of the 18261 labelled images in `images_flat`

together do not reach the 5%. Fortunately, the missing data belong mainly to the most frequent labels.

The imbalance of the labels was the main reason behind the creation of the `Hierarchical` dataset. Despite its small size, it is a useful guideline as it introduces a first refinement of the 10 classes splitting the largest ones in 26 subclasses according to a depth 2 tree structure. Figure 2.4 lists all the subcategories and shows how they are distributed among the 10 classes.

2.3 The scale problem and the OCR program

Images on different scales represent different objects, thus exhibit different high level features. That is why it makes sense to split the dataset by the scale.

Recovering the information on the scale is not as easy as one can think. This metadata, where present, is not always stored in the same way in the TIFF files but in the vast majority of the cases is hard coded on the image. Visually, they can be immediately recognized as a segment, and the ground

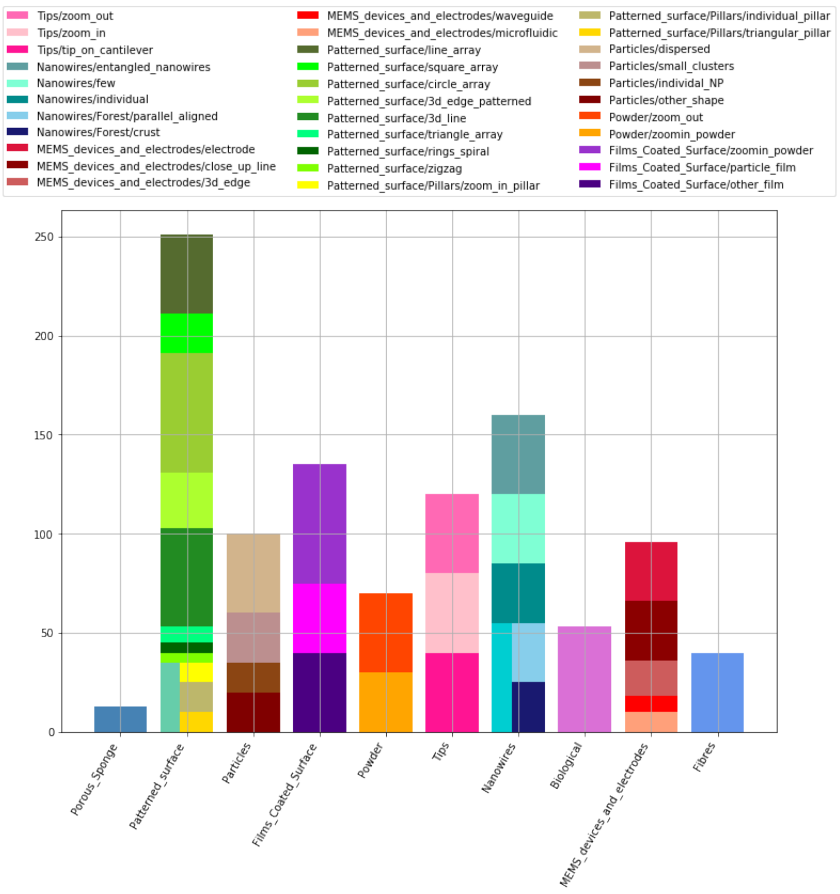

Figure 2.4: Distribution of the sub-classes in `Hierarchical`

distance corresponding to it, appearing on the left side of a stripe containing other data. The stripes are usually located on the lower part of the images and can differ in size, background color, text font, text color, and metadata displayed.

In order to read and store the scale data, the following algorithm was implemented using the library `OpenCV v3.4.1` for image segmentation and contours detection, and the OCR engine `Tesseract v3.04.01`:

1. Detect the stripe.
    - Find a square region in the lower part of the image where the brightness of the pixels have almost zero covariance.
    - Store the average brightness of the square (probable color of the stripe).

    - If the color is too dark (dark background stripe with bright text) revert the colors.
    - Use the above color as threshold and perform an image segmentation.
    - Return all big white contours with width comparable to the total width of the image.

2. If one, and only one, stripe is detected, spot the segment.
    - Do a segmentation of the left part of the stripe.
    - Return all big contours with small extent (i.e. the ratio between the contour area and the area of bounding rectangle).
3. If one, and only one, segment is detected, read the text above it.
4. Clean the text selecting the scale elements only (digits followed by mm, μm, or nm).
5. Store the scale and the pixel length of the segment.

Over the 146 916 images in `images_flat` the program did not detect the stripe in 4105 cases (2.8%) and did not spot the segment in just 26 cases (0.02%). To validate the program, the scales of a uniform sample of 1000 images had been written down by hand and compared to its outputs. The validation results are reported in table 2.3).

Positive	**98.9%**	989		
Negative	**1.1%**	10	1.0%	stripe not detected
		1	0.1%	wrong scale
Total		1000		

Table 2.3: Validation of the OCR program

When the stripe is not detected the program returns a specific error. It is worth noticing that among the 35 images of the validation sample where this happened, in 25 cases the images had no stripe. Indeed, in 10 of the 11 not validated outputs, the program was not able to read the data and in a unique case it returned a wrong scale.

Table 2.4 details the results of the OCR program on the `images_flat` dataset. The scale was succesfully read on the 98% of the labelled images and on the 97.2% of the whole dataset. One can see that the scale ranges from a millimetre to a nanometre and thus on the images there is up to 10^6x magnification power difference.

Scale	`images_flat`, labelled from `SEM_Dataset`											Total in
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	Tot	`images_flat`
1 mm	0	6	0	0	0	0	0	0	17	0	23	163
300 µm	0	1	0	0	0	0	0	0	0	0	1	8
200 µm	2	128	5	0	11	34	11	0	332	7	530	3265
100 µm	57	298	31	1	32	33	92	30	187	29	790	5941
30 µm	0	5	0	0	0	0	1	0	4	2	12	60
20 µm	14	271	36	22	44	164	55	63	595	52	1316	9475
10 µm	25	698	170	24	136	211	278	224	1032	42	2840	14729
3 µm	0	0	0	0	0	0	2	0	0	2	4	69
2 µm	34	722	311	64	22	359	744	467	862	13	3598	22575
1 µm	21	681	622	83	214	354	1147	121	710	6	3959	30107
300 nm	0	0	0	1	0	0	5	0	2	0	8	187
200 nm	10	338	1396	84	148	279	910	45	216	0	3426	37052
100 nm	5	39	548	19	224	70	269	0	23	0	1197	16531
30 nm	0	0	0	0	0	0	0	0	0	0	0	20
20 nm	1	3	122	0	1	6	39	0	0	0	172	2423
10 nm	0	0	1	0	0	0	0	0	0	0	1	62
4 nm	0	4	0	0	0	0	0	0	0	0	4	6
2 nm	0	0	0	0	0	0	0	0	0	0	0	6
1 nm	0	24	0	0	0	0	0	0	1	0	25	69
Total images with scale metadata											17906	142748
											98.1%	97.2%

Table 2.4: Scales in `images_flat`

In order to focus on images displaying similar features, in the following work only the images in `images_flat` having scale 1 µm and 2 µm are considered. For sake of clarity, this subset will be referred to as `1u-2u`. Table 2.5 reports the distribution of the labels in it.

`1u-2u`, labelled from `SEM_Dataset`											Total in
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	Tot	`1u-2u`
55	1403	933	147	236	713	1891	588	1572	19	7557	52682
Percentage										14.3%	

Table 2.5: Distribution of the labels in `1u-2u`

Chapter 3

Distances and Feature Learning

In order to perform cluster analysis on the data in `images_flat` a notion of distance between images is needed. In literature it is possible to find various definitions, mainly in the form of image similarity indices (see for example [9] and [10]). A particular family of metrics, derived from the so called Structural Similarity Index (SSIM), has been widely used in the last ten years. Although the SSIM metrics compare local patterns of pixel intensities, and thus they are robust to small rotations and translations, they do not perform a content analysis; therefore they are not able to evaluate the similarity between different objects. As expected, the results obtained using these distances were not satisfying and the idea of applying a definition of distance directly on the images was soon abandoned.

A different way to proceed is to pre-process the data selecting high-level features characterizing the images and, in this way, to reduce the dimensionality of the dataset. Once the images are projected as data points in a real space of a sufficiently low dimension, the Euclidean distance can be computed.

Although, theoretically, both feature learning and dimensionality reduction can be achieved in a completely unsupervised way, the curse of dimensionality (the sparsity of the data in the input space) forces to find a middle ground between supervised and unsupervised learning methods.

In section 3.1, the `Inception V3` architecture is introduced; this Convolutional Neural Network reported good performance both in the ImageNet 2012 visual recognition challenge (see [11]) and, after a fine tuning on the `SEM_Dataset`, in classifying the SEM images (as described in [3]). It is known that the last layers of a Deep NN provide high-level features of the input data, features learned using labelled data during the supervised training phase. The first step in the feature learning process is performed considering the outputs of the final part of this deep CNN.

To shrink the space where the data lie even further, it is possible to choose between various dimensional reduction tools. Models that seem particularly suited for the task are undercomplete autoencoders. Section 3.2 is devoted to describe this special class of Neural Networks. Moreover, a python implementation using `TensorFlow` is presented and the results of the training phase are discussed.

The size of the space where to project the data has to be chosen carefully, if it is too big the important features of the dataset are not accentuated; on the other hand, if it is too small the projection results in a loss of information. Thus, it is fundamental to understand the Intrinsic Dimension of the dataset. This is done in section 3.3.

The results achieved combining the above tools are collected in section 3.4. In particular, the heatmaps of the distances of a data sample are presented and their correlation to the classification in 10 categories is scored.

3.1 Supervised Feature Learning via CNNs

Images can be seen as two dimensional arrays of pixel brightness. Thus, assuming all images to have the same size, our dataset can be seen as a cloud of points in a space of dimension 1024 x 768. Whatever p-norm distance in this space would compare the values for each dimension, that is the brightness values pixel by pixel. These intensity-based distances do not lead to meaningful information; dark images will be closer to each other than to a bright one, no matter what they are representing. Neither is useful, for the nature of the problem and the data considered, to compare the brightness local behaviour at each pixel (as the CW-SSIM [9] do). What is needed is to compare the features characterizing the represented objects.

To detect patterns within data (i.e. common features in the images) in such a high dimensional space is not an easy task due to the so called *curse of dimensionality*; the presence of unspecified structures in an inflated high-dimensional space can be hard to spot because of the inherent sparsity of the data.

For the reason above, a complete unsupervised approach for feature learning was abandoned, leaning toward a supervised solution, namely a pre-trained Convolutional Neural Network designed for image classification.

It is commonly understood (see for example [12]) that the first layers of deep CNNs learn to detect local low level features; minor details like edges, dots or circles. Moving deeper in the network, the units learn meaningful combinations of lower level features producing higher level of abstraction. Therefore, the output of the last layers of a network quantify the presence in the input data of certain high level features learned in the training phase (thus specific to the dataset used). These values are then normalized by a softmax layer to provide a probability for each class. Hence, for the purpose of extracting features, the layers immediately before the softmax are the most interesting ones.

3.1.1 The `Inception V3` architecture

The architecture chosen is `Inception V3`, a heavily engineered CNN (see figure 3.1 for a schematic representation) designed to predict the 1001 classes of the ImageNet ILSVRC 2012 classification challenge (check [13] for more details about the dataset, the classes and the challenge results). Presented in 2015 in [11], it is the third version of a network popularly known as `GoogLeNet`. The core idea under this family of networks is the inception module: it consists of parallel convolutional layers of different kernel sizes that improve the ability of the network to detect efficiently features of objects having different sizes. The benchmarks of `Inception V3` on the ImageNet 2012 validation set report astonishing results: 78.0% of Top 1 accuracy and 93.9 of Top 5 accuracy [14].

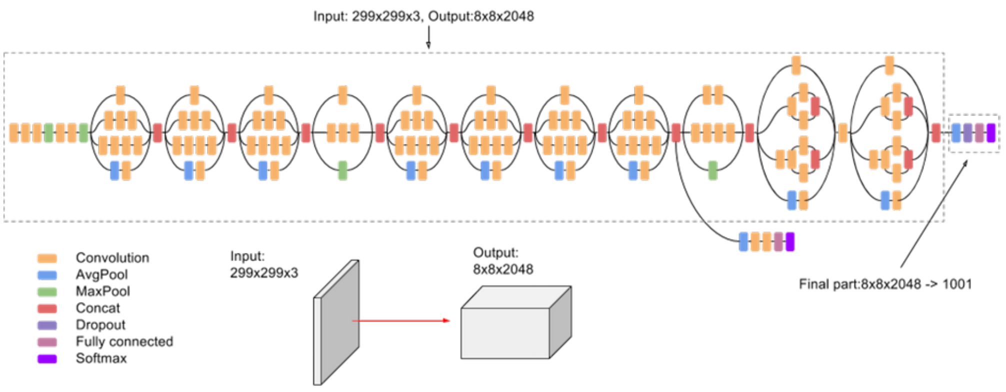

Figure 3.1: `Inception V3` architecture[1].

Although `Inception V3` is not the architecture performing the best on the ImageNet dataset, it was chosen in accordance with the previous works legacy. In [2] the model trained on ImageNet was modified by substituting the last fully connected layer with a smaller one according to the classification problem, thus passing from an output dimension of 1001 to 10. While almost all the weights of the network were left untouched, the new layers were

[1]Image retrieved from https://cloud.google.com/tpu/docs/inception-v3-advanced.

retrained on 80% of `SEM_Dataset` (the rest of the dataset was equally split in test and validation sets). Using this technique, called *transfer learning*, an average train accuracy on the test set of 86.2% was reached. An improvement in accuracy was obtained later on in [3] via fine tuning; a technique where the retraining part involves all layers of the network. In this case, no layer has been modified. As the network architecture is kept unchanged the prediction consists of 1001 values where only the first 10 have meaning for the SEM classification problem. The reported average accuracy on the test set during the fine tuning is 96.7%.

3.1.2 Feature Learning methods

At this point two different approaches can be considered:

- to reuse the model after fine tuning on `SEM_Dataset` thus expecting a strong correlation between the features analyzed and the classification in 10 classes;
- or to consider the model trained only on ImageNet and observe general-purpose features learned on a sheer dataset with a great number of classes.

Both procedures suggest interesting perspectives. In the first case the outputs will be distributed with a strong bias towards the 10 classes of `SEM_Dataset` so it is more suitable for an analysis of possible hierarchical trees of sub-classes. On the other hand, the second method reveals the presence of hidden structures in the dataset. Indeed, although it cannot be considered purely unsupervised feature learning it is still dataset independent.

The `TensorFlow` inference graph of `Inception V3` trained on ImageNet is easily obtainable as a binary ProtoBuf file from the TF-Slim API [14]. In the same document is explained how to update the model weights from training checkpoints. Luckily, the checkpoints of the fine tuning on `SEM_Dataset` were available, thus it was possible to perform inference on the images with the very same model trained in [3].

An analysis of the final part of the `Inception V3` architecture (immediately after the last inception module) is necessary to understand where to search for interesting features. The input tensor has shape 8x8x2048; still too big but it is immediately reduced by the average pooling layer to 1x1x2048. The following layer is a dropout thus, during the inference phase, it acts just like the identity. At this point a last fully connected layer shrinks the tensor dimension further to 1001. This tensor quantifies the presence of the training classes in the input data, and it is squashed by the softmax function in order to extract a probability distribution over the different classes. To summarize the above analysis, there are two tensors it makes sense to analyze: the output of the last fully connected, which provides information of the 1001 highest level features, or its input, that exhibits a slightly lower level of abstraction considering 2048 features.

Layer / Model	**Dropout**	**Fully connected**
Training on `ImageNet` 2012	`1u-2u_2048in`	`1u-2u_1001in`
Fine tuning on `SEM_Dataset`	`1u-2u_2048ft`	`1u-2u_1001ft`

Table 3.1: Names of the dataset obtained from `1u-2u` by the four supervised feature learning procedures.

In the framework of the thesis, four different feature learning procedures have been analyzed: using 2 models of the same architecture (the one trained on ImageNet and the one fine tuned on `SEM_Dataset`) and considering 2 different tensors (`Logits/Dropout_1b/Identity:0`, the output of the dropout layer, and `Logits/SpatialSqueeze:0`, the squeezed output of the last fully connected). Each procedure generates different outputs thus, as reported in table 3.1, four dataset were produced from `1u-2u`.

3.2 Autoencoders

The feature learning methods explained in the previous section produce dataset with still a high dimensionality; to be specific they count 2048 and 1001 features. There is a wide range of techniques to perform feature projection in lower dimensional spaces: from linear methods like Principal and Independent Components Analysis to nonlinear ones like Isomap, autoencoders, and much more. The best technique to adopt depends strongly on the dataset. For example, linear methods are really fast but they could be a non optimal choice when dealing with real data, as real data are likely to be disposed on nonlinear manifold (figure 3.2 shows how confusing can be selecting the principal components within the 2048 features of dataset `1u_2u_2048in`). On the other hand most of the nonlinear techniques are expensive in terms of computations and memory usage (see [15] for a comparative review of various techniques).

One way to perform nonlinear dimensionality reduction without requiring a lot of memory is using autoencoders. These are special neural networks

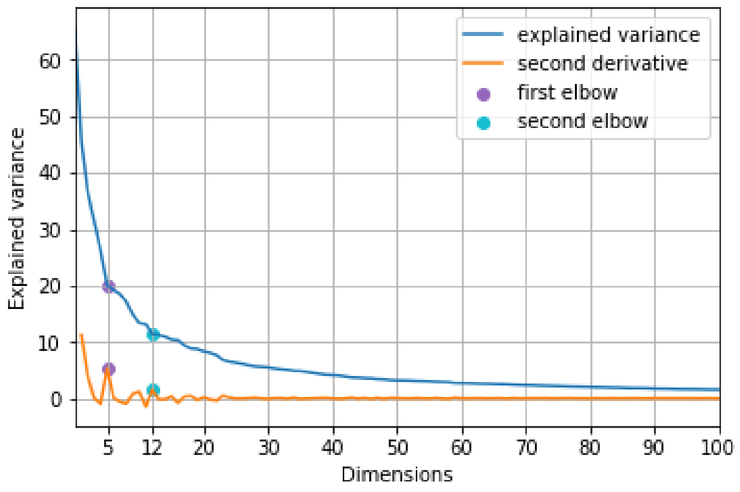

Figure 3.2: The PCA scree plot of the `1u_2u_2048in` dataset. Two faint elbows are detected at 5 and at 12 components.

with numerous applications but mainly employed for features detection and dimensionality reduction. The idea behind autoencoders can seem quite trivial but turns out to be really powerful: to learn to reconstruct the identity map. To be more specific an autoencoder is composed by two symmetric modules of dense hidden layers. The first module (called *encoder*) transforms the input data in *codings* while the second one (the *decoder*) try to reconstruct the original inputs. Each layer of the encoder module has a corresponding one in the decoder with reversed number of inputs and outputs.

For the purposes of dimensionality reduction, the number of units per layer decreases moving forward in the encoder, reaches a minimum in the codings layer, and increases symmetrically in the encoder as shown in figure 3.3. In this case the autoencoder is said to be *undercomplete*. For more information about autoencoders, their various applications, and their implementation in `TensorFlow` refer to chapter 15 of [16].

Depending on the complexity of the problem to solve, the depth of an autoencoder can vary. It has been proven that if the encoder consists of a single layer with linear or sigmoid activation function then the optimal solution of the autoencoder is equivalent to a PCA. The more layers are added, the more meaningful could be the optimal solution. Nevertheless too many variables could lead to undesired over-fitting behaviours (i.e. the network memorizes every single training input). There is no rule to get an

[3]Image retrieved from https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

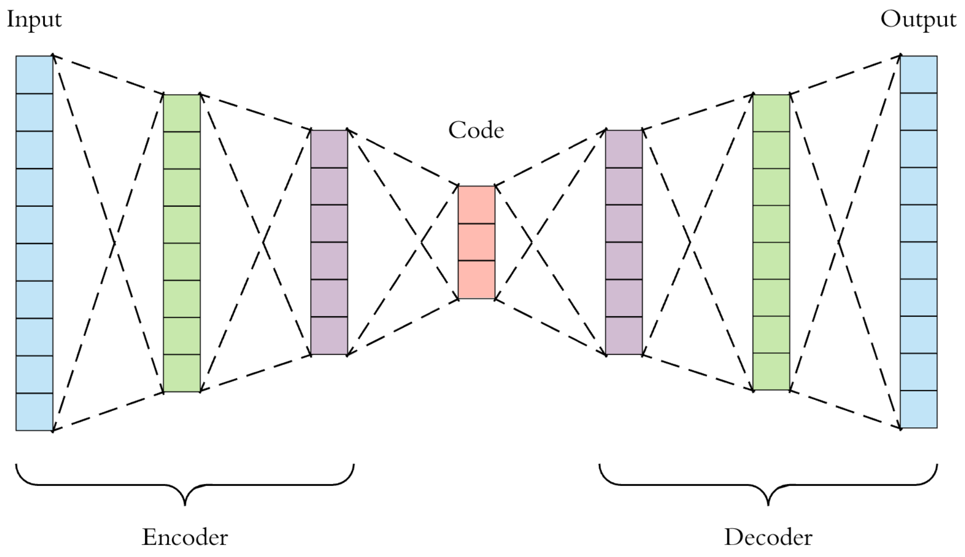

Figure 3.3: Schematic structure of an undercomplete autoencoder[3] highlighting the units in each layer.

optimal number of hidden layers and number of units per layer but a balance between the number of variables and the size of the training set is needed.

3.2.1 Tying weights

A common technique adopted to reduce the memory footprint of the autoencoder is to tie the weights of the encoder layers to the weights of the decoder layers. Suppose to have N hidden layers, where L_1 is the first hidden layer of the encoder, $L_{N/2}$ is the code layer, and L_N is the output layer. Each hidden layer is fully connected, thus respects the following rule:

$$L_i(x_i) = \sigma_i(W_i \cdot x_i + b_i) \quad \text{for each } i = 1, \ldots, N,$$

where x_i is the input of L_i, W_i is the weights matrix, b_i is the bias vector, and σ_i is the activation function. The symmetry hypothesis on the layers implies that the sizes of corresponding layer have to agree, that is $size(W_i) = size(W^t_{N-i+1})$ for each i in $1, \ldots, N$. To tie the weights of the i-th layer means to impose

$$W_{N-i+1} = W_i^t.$$

It is easy to see that this strong constraint reduces enormously the number of parameters to optimize reaching, when applied to all the hidden layers, half of the parameters. This technique is not used only for diminishing the memory footprint of the autoencoder but also because it has a strong regularization effect, avoiding singular behaviours where the weights of the encoder and decoder are highly unbalanced.

3.2.2 Training phases

Autoencoders are feed-forward networks where the training is usually achieved by minimizing via back-propagation the deviation of the output from the input. Typically the loss function is computed using the mean square error:

$$\text{Loss}(x) = \text{MSE}(x - x'),$$

where x is an input data and x' is its reconstruction, i.e. the output of the autoencoder. When dealing with deep autoencoders (having more than a single encoding layers) the training phase can be performed in a different way: the multiple hidden layers are consider as stacked single layer autoencoders where each autoencoder takes the inputs from the codings of the previous one. To be more specific, the training of a deep autoencoder with N hidden layers can be performed training the layers pairwise from the outer layers (L_1, L_N) to the inner ones $(L_{N/2}, L_{N/2+1})$. In successive training phases, two hidden layers (L_i, L_{N-i+1}) are considered as a single autoencoder and are trained on data x^i computed in a recursive fashion:

$$x^i := L_{i-1}(x^{i-1}), \quad \text{where } x^1 := x.$$

At the end of each training phase the parameters of the two layers are stored and the codings of all input data are computed and memorized in order to be used in the next phase. Although this training technique seems more complex than training the whole deep autoencoder in one go, it is often much faster.

3.2.3 `TensorFlow` implementation

For the sake of completeness, the original interface used to implement deep autoencoders in the rest of the project is documented here.

```
class autoencoder_3MPL (n_inputs, n_hidden = [100, 20, 3],
tie_weights = [False, False, False], activation_f = tf.nn.elu,
learning_rate = 0.001, l2_reg = 0.001, root_logdir = '/tmp/',
name = None)
```

This class creates a `TensorFlow` graph implementing a deep autoencoder with 6 hidden layers. Although the depth is fixed, all other parameters can be easily set.

Parameters

`n_inputs` : *int*
Number of units in the input layer.

`n_hidden` : *array-like of int (default =* `[100, 20, 3]`*)*
Number of units to use in the first 3 hidden layers.

`tie_weights` : *array-like (default =* `[False, False, False]`*)*
Specifies if the weights of the paired layers are tied. The pairs are considered in the following order: innermost layers, second and fifth layers, outermost layers.

`activation_f` : *callable or iterable (default =* `tf.nn.elu`*)*
Specifies the activation function for each layer. If a single callable is passed then it is set on all hidden layers except for the output layer that remains linear. If the iterable length is not 6, a warning is printed and the default option is considered.

`learning_rate` : *float (default =* `0.001`*)*
The learning rate for the training phase.

`l2_reg` : *float (default =* `0.001`*)*
The L2 regularization rate to apply during the training phase.

`root_logdir` : *string (default =* `'/tmp/'`*)*
The path where a sub directory containing the training records is created. The sub-directory is named after the model name and the creation time.

`name` : *string (default =* `None`*)*
The name of the model. If `None`, the model is named after the chosen architecture.

Methods

`train` `(inputs, n_epochs = [5, 5, 5, 5, 10], order = ('g'), batch_size = 150, summary_step = 0)`

Trains the autoencoder to reconstruct the input data. The training phases, their order, and the number of epochs they have to perform can be specified. Table 3.2 shows the available phases and their labels.

`inputs` : *array-like, shape =* `(n_samples, n_features)`
Training data.

Phase label	**Hidden layer**					
	1	**2**	**3**	**4**	**5**	**6**
`1_a`	X					X
`1_b`		X			X	
`1`	X	X			X	X
`2`			X	X		
`g`	X	X	X	X	X	X

Table 3.2: Hidden layers involved per training phase

`n_epochs` : *array-like (default =* `[5, 5, 5, 5, 10]`*)*
Number of epochs for each training phase considered in the following order: `1_a`, `1_b`, `1`, `2`, `g`.

`order` : *array-like (default =* `('g')`*)*
Specifies the training phases to perform and their order. For example ('g') performs a global phase only, while ('1_a', '1_b', '2') performs the training in a stacked autoencoder fashion. The phases can be repeated and performed in whatever order but they are subject to the number of epochs specified in `n_epochs`.

`batch_size` : *int (default =* `150`*)*
Size of the batches considered in the training. A big batch size entails a faster but less accurate training.

`code` *`(inputs)`*

Encodes the input data.

`inputs` : *array-like, shape =* `(n_samples, n_features)`
Data to encode.

Returns

`coded` : *array-like, shape =* `(n_samples, n_coded_features)`
Encoded data.

`decode` *`(inputs)`*

Decodes the input data.

`inputs` : *array-like, shape =* `(n_samples, n_coded_features)`
Data to decode.

Returns

> **`decoded`** : *array-like, shape* = `(n_samples, n_features)`
> Decoded data.

`reconstruct` *`(inputs)`*

Reconstructs (i.e. encodes and decodes) the input data.

> **`inputs`** : *array-like, shape* = `(n_samples, n_features)`
> Data to reconstruct.

Returns

> **`outputs`** : *array-like, shape* = `(n_samples, n_coded_features)`
> Reconstructed data.

`load_weights` *`(model_root_logdir, deepcopy = True)`*

Loads the parameters (weights and training records) of a previously trained model.

> **`model_root_logdir`** : *string*
> Path where to find the training records.
>
> **`deepcopy`** : *booleand (default* = `True`*)*
> If `True`, all the training records are copied in the model directory. If `False`, the model directory is changed to `model_root_logdir`; every further training will modify the previous checkpoint.

`info` *`(verbose = False)`*

Print the model information.

> **`verbose`** : *boolean (default* = `False`*)*
> If `False`, prints the name of the model only. Else, prints all the model information, including the training history.

3.2.4 Grid search

In the following section 3.3 the Intrinsic Dimension of the 4 datasets is computed (or at least bounded) providing an idea of how much the representation of the data can be reduced in the codings layer. Once the inputs and the codings dimensions are known, there are a lot of other parameters to set properly. There is no rule here to follow, thus a grid search on various sets of parameters helps to select them satisfactorily. The scoring criterion is to minimize the reconstruction error (i.e. MSE(x,x') where x belong to a test set not used in the training). In the following the sets of parameters used to build and train the grid search models are discussed.

- Number of inputs: Depending on the dataset it could be 1001 or 2048.
- Units in hidden 1: In a range between 600 and 200.
- Units in hidden 2: In a range between 200 and 50.
- Units in hidden 3 (codings dimension): Depending on the dataset it could be 10, 13, or 18. How to compute these numbers is explained in section 3.3.
- Tied layers: Empirically was shown that tying the inner layers does not achieve better performance, thus the only combination considered were without tying and tying just the outer layers.
- Training phases: Different training solutions were tested: block by block, global and mixed. The blockwise training is faster than the global training. But the depth of the network and the small number of units per hidden layer makes the gain in run time irrelevant compared to the loss in accuracy.
- Activation functions: Various combination of elu, tanh, and sigmoid functions were tested. In every case the best performing was applying the elu function on all layers but the output, left linear.
- Batch size: After a few tests the batch size was fixed at 300 to balance the time required for the training and its accuracy.
- Learning rate: The learning rate was set at 0.001. With smaller rates the convergence is slow, while higher rates entail a great risk to annihilate the weights and to fall into the local minimum where all data points are projected in their geometric center.

The models minimizing the reconstruction error in the grid search were applied to reduce the dimensionality of the 4 datasets. Their parameters are summarized in table 3.3.

Input Dataset	N. inputs	N. units hidden1	N. units hidden2	Codings Dimension	Output Dataset
`1u-2u_1001ft`	1001	T500	150	10	`1u-2u_1001ft_10`
`1u-2u_2048ft`	2048	T500	200	13	`1u-2u_2048ft_13`
`1u-2u_1001in`	1001	500	200	18	`1u-2u_1001in_18`
`1u-2u_2048in`	2048	T400	150	18	`1u-2u_2048in_18`

Table 3.3: Parameters minimizing the reconstruction error on the test set. The 'T' character indicates that the corresponding pair of hidden layers are tied.

3.3 Intrinsic Dimension

Understanding the dimension of the manifold where the data lie is fundamental to obtain a reduced representation without loosing essential information. Ideally, the dimension of the space where to project the data should be greater than or equal to the intrinsic dimensionality of the data.

3.3.1 The 2-NN algorithm

An efficient way to compute the Intrinsic Dimension (ID) of the data is using the so called 2-NN (here NN stands for Nearest Neighbors) algorithm recently presented in [17]. To outline briefly the algorithm, the main steps are listed with a python implementation:

```python
import numpy as np
from sklearn.neighbors import NearestNeighbors
from sklearn.linear_model import LinearRegression

def compute_ID (X):
    ''' X: array-like of shape (n\_samples, n\_features) '''
    N = X.shape[0]
```

1. For each data find the distances r_1 and r_2 of two Nearest Neighbors.

```python
ngbr = NearestNeighbors(n_neighbors=3,\
                        algorithm='kd_tree',\
                        n_jobs=-1).fit(X)
nn_distances, nn_indices = ngbr.kneighbors(X)
```

2. For each data in $i = 1 \dots N$ compute $\mu_i = \frac{r_1}{r_2}$

```python
mu = nn_distances[:,2] / nn_distances[:,1]
```

3. Sort in ascending order the values of μ via a permutation σ and define $F^{emp}(\mu_{\sigma(i)}) := \frac{i}{N}$.

```
i_sorted = np.argsort(mu)
F_emp = np.zeros(N, dtype=float)
F_emp[i_sorted] = [i/N for i in range(N)]
```

4. Return the slope of the homogeneous linear fit of the points having coordinates $\{(\log(\mu_i), -\log(1 - F^{emp}(\mu_i)), i = 1, \dots, N\}$.

```
x = np.log(mu).reshape(-1,1)
y = -np.log(1. - F_emp).reshape(-1,1)
l = LinearRegression(fit_intercept=False, n_jobs=1).fit(x, y
    )
return l.coef_[0,0]
```

It is worthy to analyze the time complexity of the above implementation of the algorithm. The most expensive part in terms of run time is certainly the search of the 3 nearest neighbors (NN) in the first step. Indeed step 2, 3, and 4 have linear time complexity (the least squares Linear Regression is applied on 2 coordinates only). On the other hand the `fit` method of the `NearestNeighbors` class builds a k-d tree, which already has $\mathcal{O}(N \log N)$ complexity, and the `kneighbors` performs 3 NN searches for each of the N data. The complexity of a single NN search on a k-d tree depends on the distribution of the data but, on average, can be considered logarithmic. Therefore, the overall time complexity of the algorithm is $\mathcal{O}(N \log N)$.

Before applying the above ID estimator to the four datasets, it is important to call the attention to some observations about the algorithm. First of all, the theoretic machinery behind the 2-NN method requires the dataset to be locally uniform in density. Although it could seem a strong hypothesis, it involves just the first 2 neighbors of each data point and, empirically, it has been shown that the estimator stands small density perturbations.

Another thing to take care about is that the estimation of the linear fit parameters is heavily affected by outliers. These correspond to data where r_2 is much bigger than r_1. Hence, to reduce their effect in the linear regression, it is better to discard the points of the plane having big abscissa. In [17] the authors suggest to take into account only the first 90% of the points.

Moreover, the estimation of the dimension can be altered by the presence of noise in the data. For example figure 3.4 represents two samples of a uniform distribution along the x axis (between 0 and 100) with a small gaussian noise on the y axis ($\mu = 0$ and $\sigma = 0.001$). The two samples differ by the number of points: there are 50000 blue points and 1000 orange points. Clearly, more points are drawn, thicker is the line. The linear fit of the two samples changes dramatically: in the first case (blue points) the slope is close to two while in the second case (orange points) the slope is around one.

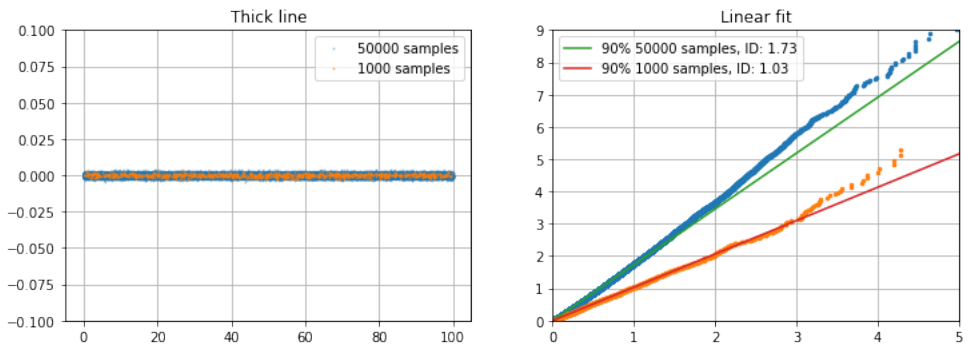

Figure 3.4: Intrinsic dimension of two sampling of the same distribution. The estimated dimension of the orange line is 1 while for the thick blue line it is close to 2.

A best practice to avoid this effect is to analyze how the estimated ID of data samples varies with their sizes. As shown in figure 3.5, a plateau should be found where the estimation reaches the ID of the data distribution.

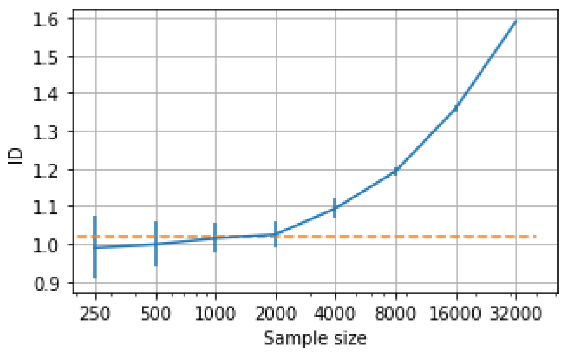

Figure 3.5: Estimated ID of thick line with increasing sample sizes. The plateau can be found around one, that is the expected dimension of a line.

3.3.2 ID of the 4 datasets

As explained in section 3.1, the data produced by the CNN fine tuned on `SEM_Dataset` are expected to be more structured than the ones coming from the CNN trained on `ImageNet`. That is mainly because the last layer of the network, during the fine tuning, learned to activate only the 10 units corresponding to the 10 labelled classes. Thus, it is right to assume the dimensionality of `1u-2u_1001ft` to be at most 10 and that it slightly increases when considering a lower level of abstraction as in `1u-2u_2048ft`. On the other hand the data points produced by the network trained exclusively on `ImageNet` should have less constraints as the last layers units are activated by more general features.

Figure 3.6 shows the fitting functions in the four datasets. In each case the fit is performed thrice: the yellow line fits the totality of the points, the green line is computed discarding the last 10%, and the red line fits just the first half of the points.

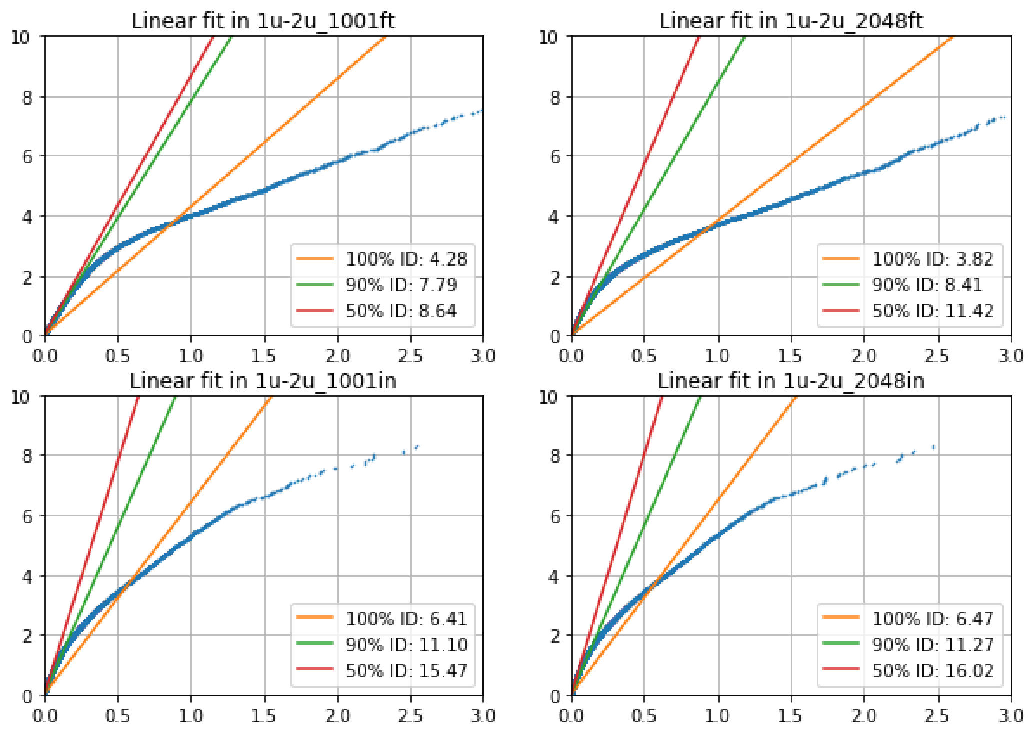

Figure 3.6: Linear fit for the 4 datasets. In each case three linear fit are shown: considering the totality of the points, the first 90%, and the first half.

One can clearly see that in all cases the blue curve has a heavy tail. Thus, the ID computed without discarding the outliers is too low and it is not reliable. Moreover, while in the upper left graph (corresponding to `1u-2u_1001ft`) the slopes of the green and red lines are close, in all the other three cases there is a substantial discrepancy.

The results presented in figure 3.6 consider the whole datasets. As said above, to perform a more accurate analysis, it is necessary to take into account samples of different sizes.

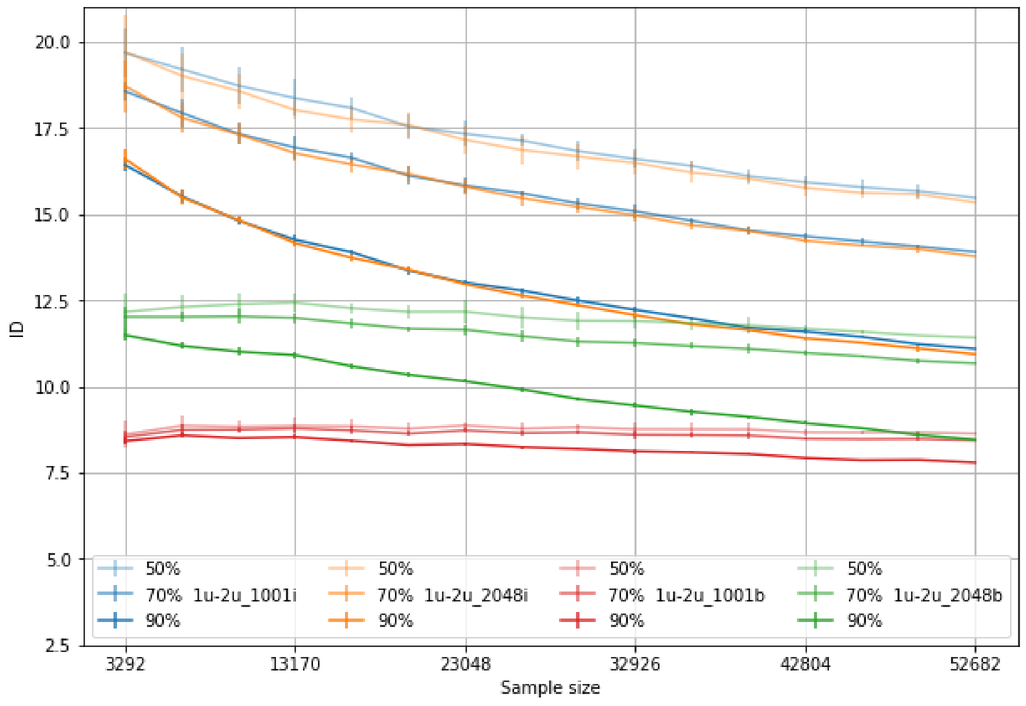

Figure 3.7: ID of different samples of the 4 datasets varying the sample size. Each dataset is represented by a color and the brightness corresponds to the percentage of points used in the linear fit.

Figure 3.7 summarizes the results produced by sampling each datasets and averaging the outputs of the 2-NN estimator on various samples of the same size. Each color characterizes a dataset: the green and red curves correspond to the datasets obtained via fine tuning while the blue and orange curves correspond to the datasets produced by the CNN that has never seen the images in `images_flat`. The three brightness level appearing for each color are related to the number of points considered in the linear fit: 90%, 70%, and 50%.

Although this exploration provides more information on the datasets, it does not present more meaningful insights than figure 3.6. As anticipated, the most clear results comes from `1u-2u_1001ft`. Indeed the red curves show an evident plateau around 9. The green curves (corresponding to `1u-2u_2048ft`) are not so close to each other but a constant behaviour around 12 can be observed when considering the 70% fit. Completely different situation is the one depicted by the blue and orange curves: there is no constant behaviour at all. One can only observe that the estimation of the ID for samples of interesting sizes can be bounded above by 18. The above analysis led to consider the codings dimensions reported in table 3.3 of the previous section.

3.4 Distances

Combining the tools presented in the previous sections, the representation of the images can be drastically reduced. Indeed, by applying a truncated NN the number of features considered is 1001 or 2048 (depending on the level of abstraction chosen), then it is dropped further by an autoencoder to a number greater than the ID of the dataset.

A way to evaluate if the above features extraction methods provide meaningful information with respect to the previous classification in 10 categories, is to analyze and compare the distances between certain data points. To be more specific, between labelled images one can define a discrete distance:

$$d_{disc}(x_i, x_j) = \delta_{l_i, l_j},$$

where l_i is the label of the image x_i. Therefore d_{disc} can be compared with the distance obtained via one of the features extraction method.

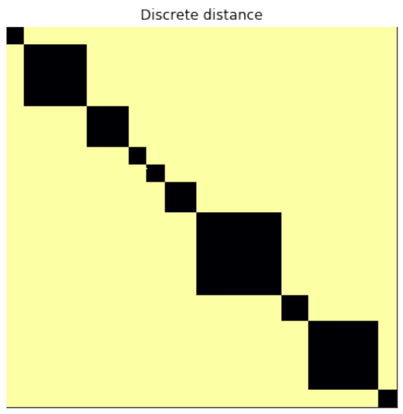

Figure 3.8: d_{disc} heatmap.

Porous_Sponge	20
Patterned_surface	70
Particles	46
Films_Coated_Surface	20
Powder	20
Tips	35
Nanowires	94
Biological	29
MEMS_devices_and_electrodes	78
Fibres	19
Total	**431**

Table 3.4: Number of samples per category.

In the following, a sample of the labelled images in `1u-2u` is considered. This set, containing 431 data, is obtained in a non-uniform way apt to balance the categories distribution (see table 3.4). The images are sorted by category, thus the heatmap of d_{disc} on this sample shows black squared block on the diagonal representing the zero distance pairs within the same category.

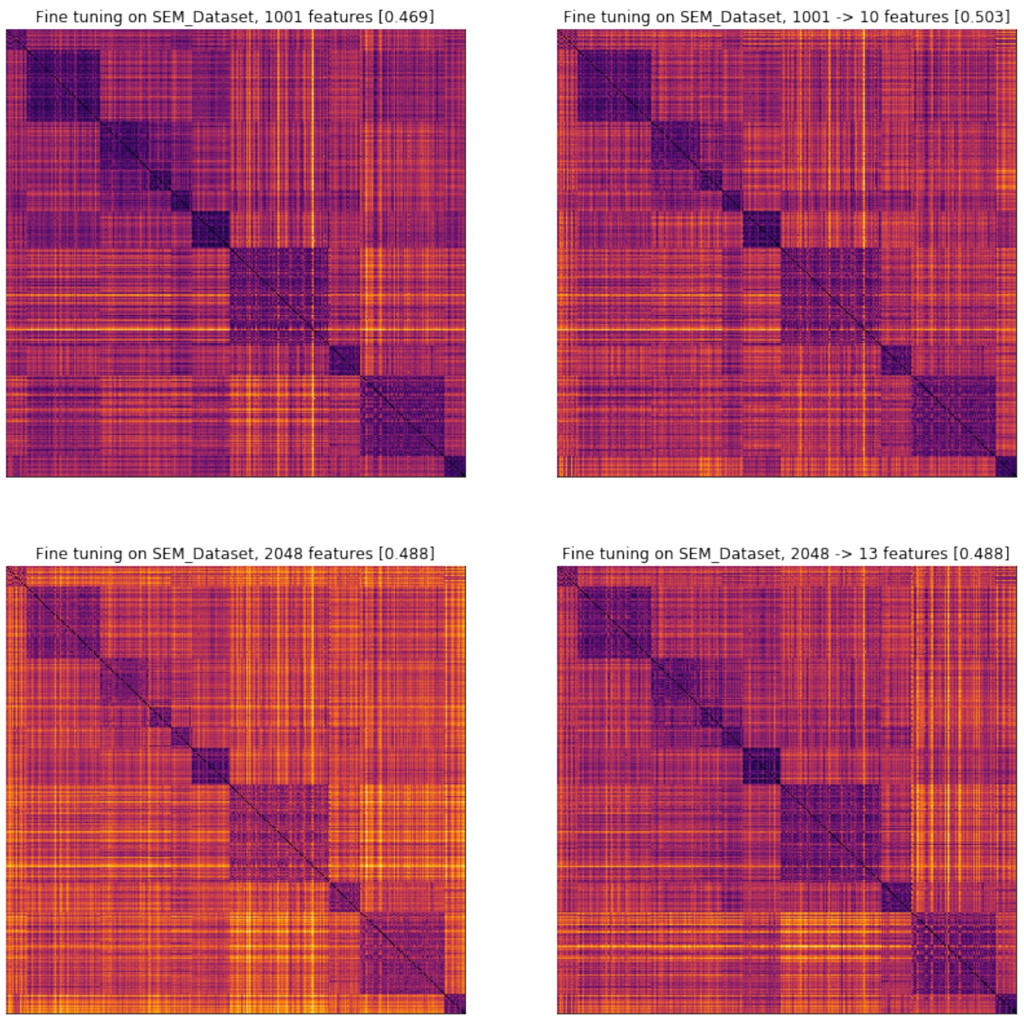

Figure 3.9: Heatmaps of the distances obtained via the fine tuned CNN. The captions recall the method used and indicate the correlation index with d_{disc}.

In figure 3.9 the heatmaps of 4 distances computed on the same sample via the fine tuned CNN are displayed. In the left column the distances are computed on the data before applying the autoencoder while the right column displays the distances between the same sample after the dimensionality reduction. In all the cases the block structure characterizing the discrete distance is quite evident and seems to became more definite moving from left to right in the same row. This observation is verified by looking at the correlation index between those distances and d_{disc}; it increases after the dimensionality reduction.

As expected the block structure almost disappears considering the out-

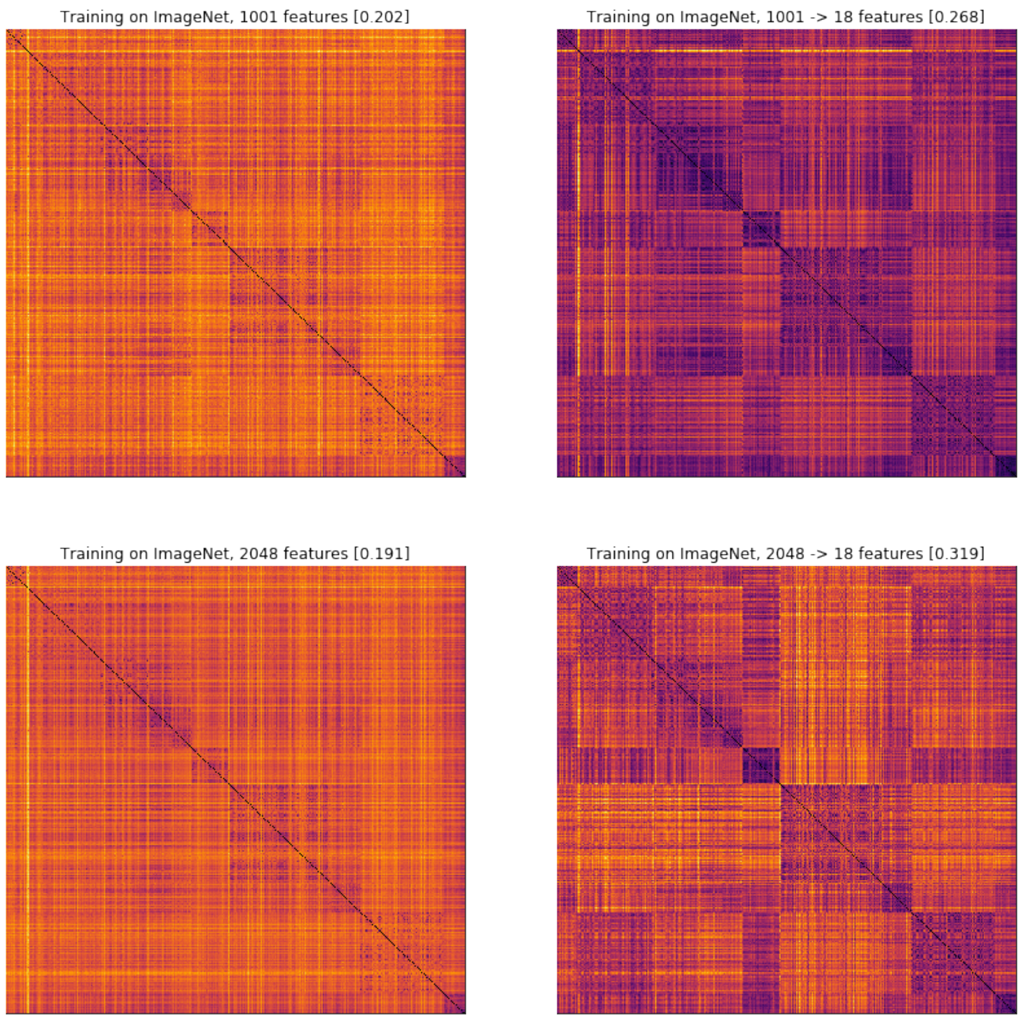

Figure 3.10: Heatmaps of the distances obtained via the CNN trained on `ImageNet`. The captions recall the method used and indicate the correlation index with d_{disc}.

puts of the CNN trained only on `ImageNet`. But, surprisingly, the correlation index between d_{disc} and the distances of the data points after the feature reductions is higher in both cases. In particular, the result obtained in the case considering the lower level of abstraction is impressive: the correlation index after the dimensionality reduction increases of the 67%, reaching 0.32.

Chapter 4

Clustering Analysis

Once a distance between the images in the `1u-2u` dataset is defined, Clustering Analysis can be performed. Although several different algorithms can be found in literature, this chapter focuses just on two of them: section 4.1 discusses the agglomerative clustering (introducing four linkage criteria) and section 4.2 is dedicated to Density Peaks. In each section these algorithms are briefly described and their results on artificial datasets (figure 4.1) are analyzed.

Due to the great amount of images contained in `1u-2u` it is impossible to validate a clustering analyzing one by one the label assigned to each image. Thus, an automatic way to score how good an algorithm performs on the dataset is needed. Moreover, it is important to score the similarity of a clustering solution against the classification in 10 categories. This problem is faced at the end of the chapter. Section 4.3 provides the reader with the tools to understand the evaluation criterion used. It introduces the Normalized Mutual Information score and provides some examples of how this score works. Finally, for each distance defined in chapter 3, the clustering algorithms studied in the previous sections are applied to the dataset `1u-2u`. The consequent scores are presented and commented in section 4.4.

4.1 Hierarchical Agglomerative Clustering

The idea behind hierarchical agglomerative clustering algorithms is really intuitive: at the beginning all data points are considered to be singleton clusters, then, at each step, the closest clusters are merge together and, the algorithm ends when a single cluster is formed.

The above algorithm depends completely on the choice of the *linkage criterion*, that is the distance between clusters considered. Although it has to derive from the distance between the data points, there are several ways to define it. In the following, some of the most common criteria are defined.

Let X be a set equipped with a distance d between its elements, C and C' two disjoint subsets of X (two clusters), then:

1. The *single* linkage is defined by
$$\tilde{d}(C, C') = \min_{x \in C,\ y \in C'} (d(x, y)).$$

2. The *complete* linkage derives from
$$\tilde{d}(C, C') = \max_{x \in C,\ y \in C'} (d(x, y)).$$

3. The *centroid* linkage can be defined if $X \subset \mathrm{R}^n$ and $d(x, y) = ||x - y||_2$ as
$$\tilde{d}(C, C') = ||x_C - x_{C'}||_2,$$
where x_C is the centroid of C.

4. The *Ward* linkage is defined recursively. Based on the fact that the distance between two singleton clusters is d, the distance between a newly formed cluster $\hat{C}$, obtained merging the clusters C and C', and a cluster D is
$$\tilde{d}(\hat{C}, D) = \sqrt{\frac{|D| + |C|}{T} d(D, C)^2 + \frac{|D| + |C'|}{T} d(D, C')^2 - \frac{|D|}{T} d(C, C')^2},$$
where $T = |D| + |C| + |C'|$.

To better understand the differences between these clustering algorithms, it is useful to compare their solutions and their performances on synthetic datasets. The first four columns of figure 4.1 show the clustering obtained by the above linkages on 2D artificial dataset and, for each case, the running time is provided. Observing the images in the first two rows, the single linkage is the only linkage between the ones analyzed able to detect the non-convex clusters. On the other hand, it does not perform well on dataset with noisy clusters, as shown by the results in the third row; in this case the Ward method provides the best result. All the algorithms detect successfully separated globular clusters (fifth row), but when the distances between the clusters are small they all perform poorly. The last row could seem a trivial case but it highlights the "rich get richer" behaviour of the single linkage. This is common behaviour to almost all linkages except for the Ward linkage that has a strong bias toward globular clusters and generally provides more balanced solutions.

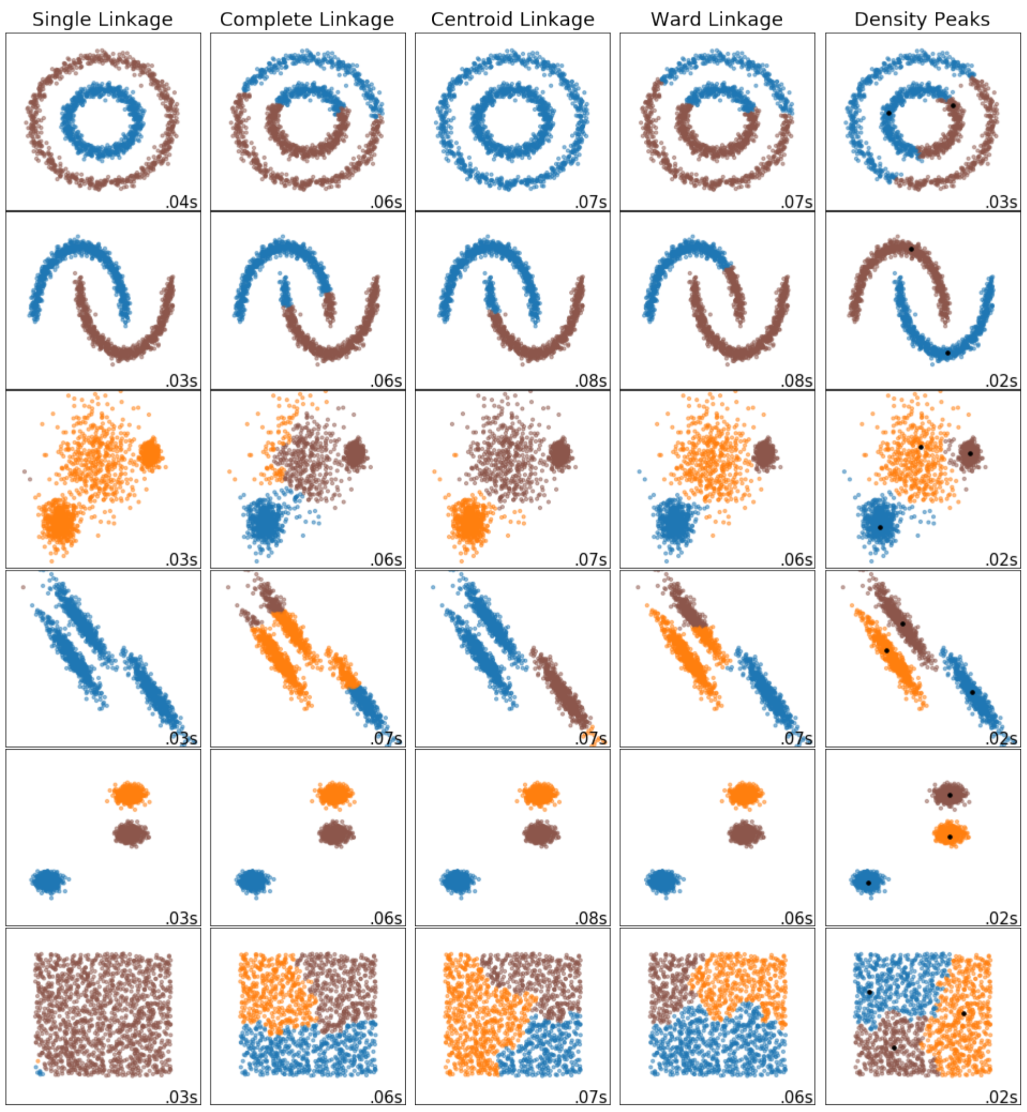

Figure 4.1: Comparison of the solutions and of the running time of the five clustering algorithms used in the thesis on 2D artificial datasets with different shapes. The datasets are generated by the `sklearn.datasets` [18] utilities and count 3000 points each. Note that the number of clusters the algorithms had to provide was fixed for each dataset: 2 for the first two rows, 3 in all the other cases. The first four columns correspond to the agglomerative clusterings produced by different linkages while the last column reports the solutions obtained by Density Peaks. In the last case the black points represent the cluster centers.

4.2 Density Peaks

The second algorithm studied is a density based clustering algorithm called *Density Peaks*. This section sketches the main idea behind this algorithm while the full details can be found in [19].

The core concept in Density Peaks is that, as clusters should be formed by objects that are close to each other, the objects density should be higher in the center of a cluster and lower at its border. Therefore, cluster centers can be thought to be peaks of the density function.

Following this idea the search for the cluster centers starts considering all the local maxima of a density function. It is unlikely that two density peaks that are close to each other form two different clusters. Thus, cluster centers are considered to be elements that are peaks of high density and lies far from the nearest element with an higher density.

Once the cluster centers are identified, the label assignation starts from them and goes trough every element following the slopes of the density function.

Density Peaks reported astonishing results in detecting both convex and non-convex clusters also in presence of background noise [19]. Indeed, by modifying slightly the algorithm, the background noise can be recognized and discarded. Anyway, the implementation presented below and used in the thesis does not consider the noise detection.

Although Density Peaks is able to identify clusters of various shapes, it performs poorly on extended distributions of points with uniform density. Consider for example the first dataset presented in figure 4.1. The inner ring has higher density than the outer one and the two peaks with highest density lie on it. Although these two points are connected by a path of similar high density, their are quite far from each other. On the other hand, the highest peak in the outer ring has much lower density and its nearest element with a higher density lies in the inner ring. Hence, it has low density and its distance to the nearest denser element is comparable to the distance between the two rings. Therefore, the assignation of the two cluster center goes to the two highest peaks, and so Density Peaks is not able to distinguish the two rings.

To prevent the behaviour described above, the authors are releasing a new version of the algorithm [20]. In the assignation of the cluster center, it discards peaks connected by a path of similar density rewarding peaks that are separated by low density regions. In this way it avoids to split clusters with uniform density.

4.2.1 Implementation

The implementation of the Density Peaks algorithm used in the thesis retraces an idea suggested by G. Sanna (personal communication, September 2018). The core of the algorithm is to construct the array `nearest_denser` containing for each element the index of its nearest element with higher density. This can be computed in two steps:

1. Find the local density peaks;
2. For each peak find the nearest element with higher density.

Both steps can be faced using a k-Nearest Neighbor algorithm. In the first one, the density of each element is compared to the densities of its k-nearest neighbors. For each element, the index of its nearest neighbor with higher density is stored on `nearest_denser`. Thus, the indices of the local density peaks are the ones where `nearest_denser` coincides with the identity.

Once the local peaks are identified, their nearest denser must be found. For each local peak its density is compared to the density of all the elements sorted by distance from it. Computationally, this is the most expensive part as it requires $\mathcal{O}(PN\log(N))$, where N is the total number of elements and P the number of peaks. A small trick can reduce the computational cost: instead of considering all the elements it takes into account only the elements having density higher than the lowest peak. The benefit of this trick depends enormously on the dataset, the density function, and the number of neighbors considered. Although no statistical study has been conducted, in some cases faced in the thesis it reduced N by almost two order of magnitude.

To proceed to the labels assignation one has to select the cluster centers. In [19] this is done by looking at a decision graph. This procedure can be naively automatized, requiring just the number of clusters to consider. This is done by sorting the elements in decreasing order by the product of their density and their distance to the nearest element with higher density. The k cluster centers will be the first k elements.

Although Density Peaks is not a hierarchical clustering, the above criterion to detect the cluster centers transform it in a divisive top-down clustering: passing from k clusters to $k+1$, one cluster is split in two.

In figure 4.1 a comparison of the running-time of the clustering algorithms studied in the thesis is presented. In every datasets analyzed the above implementation of Density Peaks is faster than all the other agglomerative clustering.

4.3 Scoring the results

How to score the similarity of two clustering is a widely studied problem and numerous measurements are available in literature. Clearly, the focus of these measures is not on the labels provided by an algorithm but on the partition of the data it induces. Indeed a permutation of the labels in a clustering solution does not change the structure of the clusters. That is why, in this section, the terms clustering and partition will be used interchangeably.

A scoring coefficient that seems particularly suited for the task of this chapter is the Normalized Mutual Information. Informally, the normalized mutual information measures the quantity of information that two partitions share. The formal definition follows.

4.3.1 Normalized Mutual Information

The normalized mutual information coefficient is based on the notion of entropy. Let X be a set of objects and $\mathcal{C} = \{C_1, \dots, C_k\}$ a partition of X, the *entropy* of $\mathcal{C}$ is defined as

$$\mathcal{H}(\mathcal{C}) = -\sum_{i=1}^{k} P(C_i) \cdot \log_2(P(C_i)), \quad \text{where } P(C_i) = \frac{|C_i|}{|X|}.$$

Intuitively, the clustering entropy measures the uncertainty of the cluster assignation of an element $x \in X$. For example, consider the case where $\mathcal{C}$ is composed by k evenly balanced cluster. In this simple example the entropy is $\log_2(k)$ thus it is zero when there is just one cluster (no uncertainty) and it increases with the number of clusters.

Consider now a second partition $\mathcal{C}' = \{C'_1, \dots, C'_l\}$ of X. In order to measure the similarity of the two partitions one can ask how much of the entropy of $\mathcal{C}'$ can be explained by the entropy of $\mathcal{C}$. Formally, the *mutual information* is defined as

$$\mathcal{I}(\mathcal{C}, \mathcal{C}') = \mathcal{H}(\mathcal{C}, \mathcal{C}') - \mathcal{H}(\mathcal{C}|\mathcal{C}') - \mathcal{H}(\mathcal{C}'|\mathcal{C}), \tag{4.1}$$

where $\mathcal{H}(\mathcal{C}, \mathcal{C}')$ is the entropy of the partition obtained considering the intersections of the clusters in $\mathcal{C}$ and $\mathcal{C}'$, and $\mathcal{H}(\mathcal{C}'|\mathcal{C})$ is a weighted average of the entropy of the partitions induced by $\mathcal{C}'$ on each cluster of $\mathcal{C}$. By computing each term, equation 4.1 can be rewritten in a more common form

$$\mathcal{I}(\mathcal{C}, \mathcal{C}') = \sum_{i=1}^{k} \sum_{j=1}^{l} P(C_i, C'_j) \cdot \log_2\left(\frac{P(C_i, C'_j)}{P(C_i) \cdot P(C'_j)}\right),$$

where $P(C_i, C'_j)$ is the joint probability of C_i and C'_j.

A flaw of the mutual information is that, in general, the score is higher for partitions with a larger number of clusters independently of the information they share. In order to standardize the score, a normalization is usually considered. The score used in the thesis is obtained normalizing $\mathcal{I}$ by the geometric average of the entropy of the two partitions. That is,

$$\mathrm{NMI}(\mathcal{C}, \mathcal{C}') = \frac{\mathcal{I}(\mathcal{C}, \mathcal{C}')}{\sqrt{\mathcal{H}(\mathcal{C}) \cdot \mathcal{H}(\mathcal{C}')}}.$$

To conclude, it is worth to recap some of the properties that make the normalized mutual information score a good measure for comparing clustering solutions.

- It is symmetric, i.e. $\mathrm{NMI}(\mathcal{C}, \mathcal{C}') = \mathrm{NMI}(\mathcal{C}', \mathcal{C})$.
- It is bounded $0 \leq \mathrm{NMI}(\mathcal{C}, \mathcal{C}') \leq 1$ and, higher is the score more information are shared by the partitions considered:
    1. $\mathrm{NMI}(\mathcal{C}, \mathcal{C}') = 0$ if, and only if, for every $C_i \in \mathcal{C}$ and $C'_j \in \mathcal{C}'$, C_i and C'_j are statistically independent;
    2. $\mathrm{NMI}(\mathcal{C}, \mathcal{C}') = 1$ if, and only if, $\mathcal{C} = \mathcal{C}'$.

4.3.2 Examples

To understand properly how the NMI score works it is better to apply it to a simple case. Consider an ordered set of 100 elements and define, for each $1 \leq k \leq 100$, a partition $\mathcal{C}_k$ assigning progressively k labels to the elements. Thus, each $\mathcal{C}_k$ evenly split the set in k clusters and, up to a permutation of the labels, the array representing the assignation of each element is,

$$\mathrm{labels}(\mathcal{C}_k) = [0, 1, \ldots, k-1, 0, 1, \ldots, k-1, \ldots].$$

Figure 4.2 represents the 10×10 matrices obtained reshaping in a row-wise fashion the above array for 4 different values of k and assigning to each label a different color. Let the partition $\mathcal{C}_{10}$ (represented in the figure by (A)) be the ground truth to evaluate the other clusterings against. In this sense, a partition having a good score against the ground truth would correspond to a matrix where the color of a cell provides information about its column. The partitions $\mathcal{C}_5$ and $\mathcal{C}_{20}$, represented in figure 4.2 by (B) and (C), are obtained respectively by merging and by splitting the ground truth clusters. Thus their NMI score against $\mathcal{C}_{10}$ is quite high. On the other hand, each cluster of

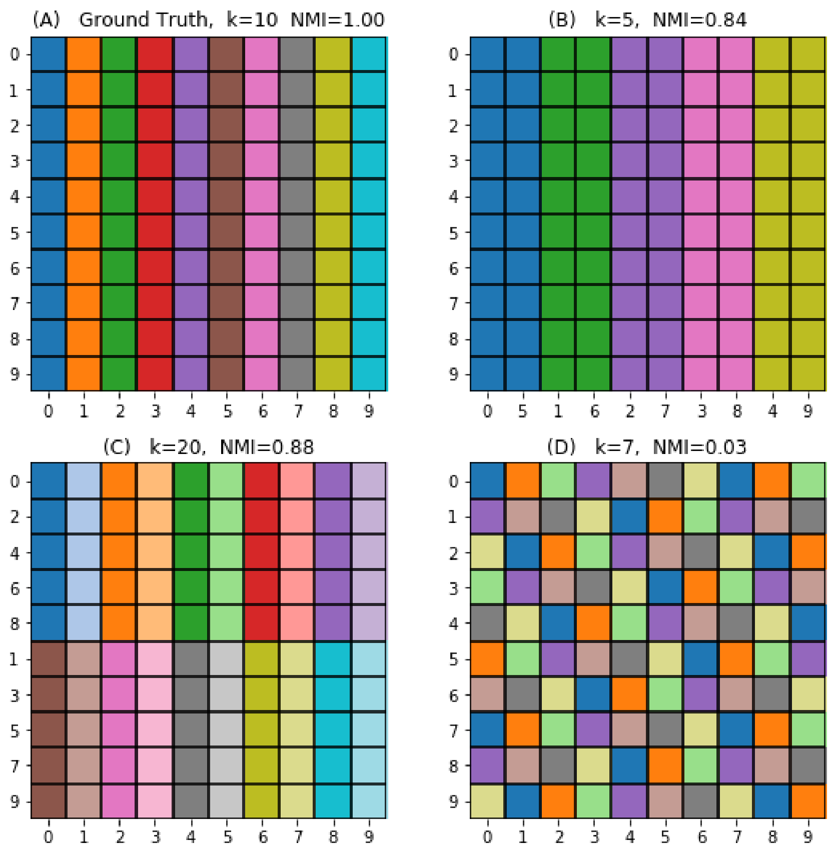

Figure 4.2: NMI scores of four $\mathcal{C}_k$ partitions of a 10×10 square against the column-wise clustering (A). In order to ease the visual identification of the clusters, (B) and (C) present respectively a permutation of the columns and a permutation of the rows.

the partition $\mathcal{C}_7$ (see matrix (D) in the figure) is split on all the columns and on each column all the colors appear. No information on the column can be deduced by the color of a cell, therefore $\text{NMI}(\mathcal{C}_{10}, \mathcal{C}_7)$ is close to zero.

The fact that the partitions considered above have different number of clusters should not surprise. Indeed, in what follows, the classification in 10 categories will be compared to hierarchical clustering and thus the number of clusters should be considered as a variable.

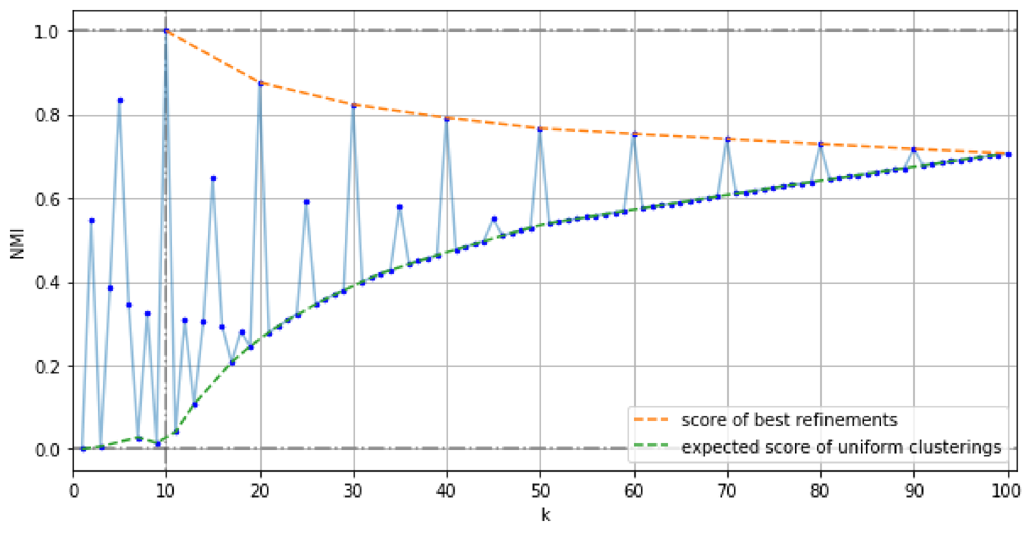

Figure 4.3: The blue points in the graph correspond to $\text{NMI}(\mathcal{C}_{10}, \mathcal{C}_k)$ varying k. These are bounded above by the score obtained in the best refinement with k clusters and below by the expected score of clustering derived by a uniform distribution of the k labels.

Forcing a little bit this purely artificial example, figure 4.3 shows how $\text{NMI}(\mathcal{C}_{10}, \mathcal{C}_k)$ varies with k. The higher NMI scores are obtained when k is a multiple of 10; that is because these case correspond to matrices where the columns are perfectly split in even parts (like in figure 4.2 (C)). The orange curve connecting these points has a smooth decreasing behaviour. On the other hand, when k is prime to 10, the score of $\mathcal{C}_k$ against $\mathcal{C}_{10}$ coincides with the expected score of a uniform distribution of the k labels over the 100 objects. Thus, the green curve should be considered as a baseline; every result above it entails the presence of some meaningful information shared between a partition and the ground truth. Clearly, the orange curve decreases while the green curve increases as, for $k = 100$, they both degenerate in the trivial partition, i.e. when every element forms a singleton cluster.

The important fact highlighted by the above example is that, when comparing two partitions with different number of clusters, the the NMI score should not be considered bounded by 0 and 1 as these bounds are shrunk.

A similar analysis can be done on the dataset `1u-2u`. In this case, the ground truth is provided by the classification in 10 categories derived from `SEM_Dataset`. As reported in table 2.5, `1u-2u` counts 7557 labelled data and only the labels assigned to these images is considered when scoring a clustering on `1u-2u`.

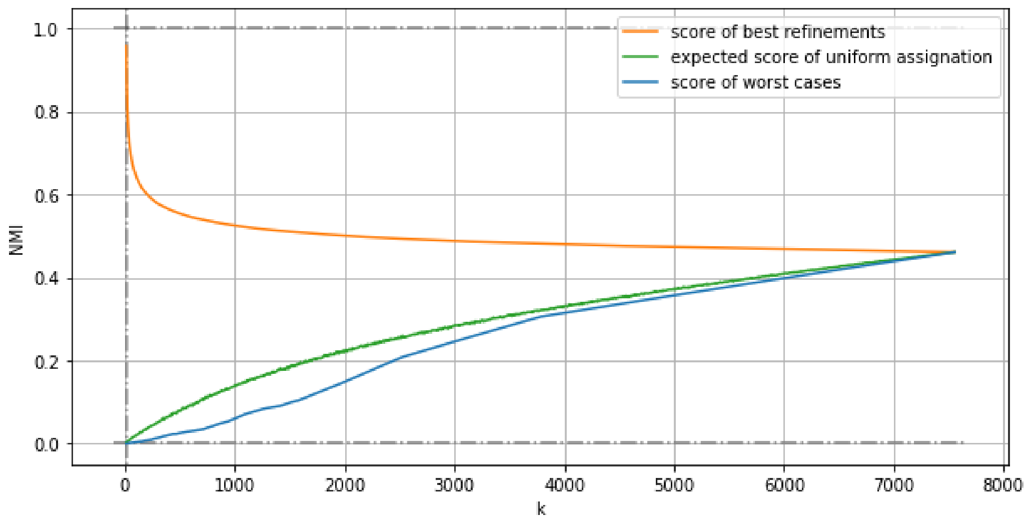

Figure 4.4: Reference curves for the NMI score against the classification in 10 categories on the `1u-2u` dataset computed using artificial partitions.

The labels of the 10 categories entail a partition in 10 clusters of the 7557 images. In a divisive way, artificial refinements of this partition were created for each $k > 10$; they are constructed recursively following the idea that a good scenario happens when at each step the bigger cluster of a partition is evenly split into two clusters. The orange curve in figure 4.4 reports the NMI scored by these refinements. On the other hand, the expected score of partitions created by uniform assignations of the labels is drawn by the green curve. It is important to highlight that those two curves do not represent the bounds of the NMI score but they should be considered just as references. One can easily consider partitions whose scores are represented outside the area within the two curves. For example, the blue curve represents the score achieved by the worst scenario considered; that is an orthogonal [1] assignation of the k labels with respect to the 10 categories.

[1]Here the term orthogonal is used referring to the previous example. Indeed the lowest NMI score against the column-wise clustering of the matrix in figure 4.2 (A) is achieved assigning the labels row-wise.

4.4 Scores on `1u-2u`

In this section the scores obtained by applying the five clustering algorithms on `1u-2u` are presented. This is done considering one by one the four distances defined in section 3.4.

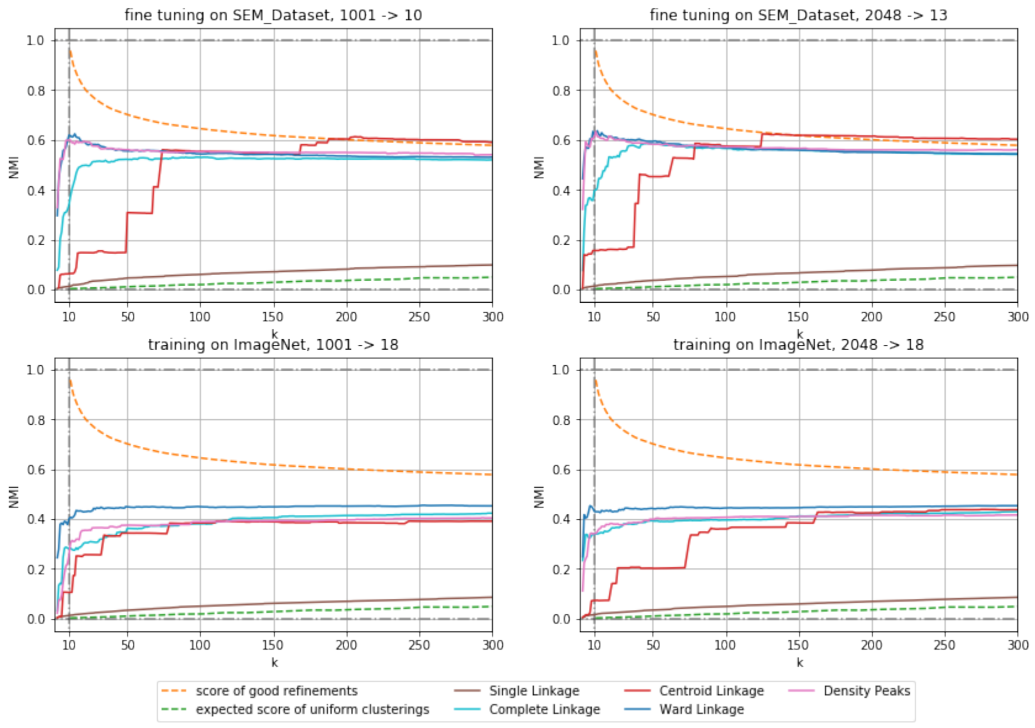

Figure 4.5: NMI scores of the clustering obtained by the five hierarchical algorithms considered as a function of k, the number of clusters. Each of the four distances defined in section 3.4 is considered in a distinct frame.

Figure 4.5 displays in each frame the results obtained by considering a particular distance. The distances analyzed in the frames in the first row are defined via the CNN fine tuned on `SEM_Dataset`, while in the second row via the network trained to distinguish more general features on `ImageNet`. In accordance with the analysis of the correlation with the discrete distance done in section 3.4, the results in the top frames present a higher NMI score than the ones in the lower frames.

Comparing the clustering algorithms within the same frame, one can immediately spot some patterns. The worst scores are always obtained by the single linkage (brown curve). An analysis of the clusters cardinalities shows that this algorithm produces a big cluster and, increasing k, small clusters or singletons are dropped form it. On the other hand, the complete and Ward linkages (represented respectively by the cyan and red curves) as well as Density Peaks (pink curve) behave similarly and present good results. Actually, the last two produce almost identical scores in the first row. In this case the distances are strongly biased towards the ten categories and both algorithms show a peak in the score around 10. On the contrary, when the distances are dataset independent the Ward linkage is the one performing the best. In particular, when a lower level of abstraction is considered (bottom right frame), a peak in the score can be spotted just before 10, although there is no bias towards the 10 categories. Without any doubt, the most impressive results in the first row are obtained by the centroid linkage (red curve). Even thought for a small number of cluster ($k < 70$) its scores are quite poor, they rapidly increase outperforming the results obtained by the artificial refinements used as reference of good scores.

Chapter 5

Conclusions

In this thesis different strategies to uncover intrinsic structures among the images captured by SEM microscopes were analyzed. In particular, the approaches considered can be distinguished according to the images features they are focusing on:

1. the 10 features corresponding to the categories of `SEM_Dataset`, learned by training on SEM images;

2. the 1001 general features learned by training on the `ImageNet`dataset.

Moreover, in order to reduce the bias introduced in the supervised learning phase, each approach was conducted observing not only the highest level of abstraction but also a slightly lower one.

Clearly, the nature of the features analyzed influences heavily the results. In the first case the structure of the classification in 10 classes is tangible among the data (see the distances heatmaps illustrated in figure 3.9). On the contrary, the second approach leaves other hidden patterns to emerge, also if they are not consistent with the previous hand made classification.

Despite the huge differences between those two approaches, the thesis depicted a common procedure to pass from the features observed via supervised learning to a clustering of the `1u-2u` dataset. This fact is principally motivated by exposition coherency. If the focus is set on the first approach only, to perform a good unsupervised feature learning there would be no need to compute the intrinsic dimension of the dataset in the features space and to train an autoencoder. Indeed, in this case, a Principal Component Analysis would produce even better results. On the contrary, as illustrated in figure 3.2, in the second approach a linear method for dimensional reduction is not sufficient.

The results exposed in the last chapter, in particular figure 4.5, should not be considered as mere evaluations of how the various clustering algorithms perform. Indeed there are two major observations that are worth highlighting. The first one is a rough remark on the classification of `SEM_Dataset`: although there is no direct relation between the general features provided by `ImageNet` and its 10 features, the structures they induce on `1u-2u` somehow correlate. The second observation is specific to the results obtained in the framework of the first approach: the centroid linkage achieves impressive scores when the number of clusters is around 200. This result surely deserves to be further analyzed in the future, as it is likely to produce a first ready-to-use refinement of the classification in 10 categories.

The last remark increases the confidence that future developments of this project can reach the goal of classifying the SEM images in a tree structure of sub-categories without any further labelling effort. A possible direction to follow is to repeat the above procedure (feature learning and subsequent clustering) restricted to images belonging to the same category. In this case, the number of the labelled images should be increased using the predictions realized by the CNN in [1] (even though this model achieves a worse overall accuracy than the one used in the thesis, its confusion matrix reports a better accuracy on the less represented categories). Another aspect that should be further studied in order to gain more insight about the hidden structures in the dataset, is to extend the spectrum of different clustering algorithms analyzed, for example considering hierarchical DBSCAN and spectral clustering.

Bibliography

[1] M. Modarres, R. Aversa, S. Cozzini, R. Ciancio, A. Leto, and G. Brandino, "Neural network for nanoscience scanning electron microscope image recognition.," *Scientific Data*, vol. 7(1), no. 13282, 2017.

[2] R. Aversa, "Scientific image processing within the nffa-europe data repository," Master's thesis, MHPC, 2016.

[3] C. De Nobili, "Deep learning for nanoscience scanning electron microscope image recognition," Master's thesis, MHPC, 2017.

[4] R. Aversa, M. Modarres, S. Cozzini, R. Ciancio, and A. Chiusole, "The first annotated set of scanning electron microscopy images for nanoscience," *Scientific Data*, vol. 5, no. 180172, 2018.

[5] NFFA-EUROPE Project, R. Aversa and M.H. Modarres and S. Cozzini and R. Ciancio, "`SEM Dataset`," 2018. https://b2share.eudat.eu/records/19cc2afd23e34b92b36a1dfd0113a89f.

[6] NFFA-EUROPE Project, R. Aversa and M.H. Modarres and S. Cozzini and R. Ciancio, "`Hierarchical SEM Dataset`," 2018. https://b2share.eudat.eu/records/b9abc4a997f8452aa6de4f4b7335e582.

[7] NFFA-EUROPE Project, R. Aversa and M.H. Modarres and S. Cozzini and R. Ciancio, "`Majority SEM Dataset`," 2018. https://b2share.eudat.eu/records/e344a8afef08463a855ada08aadbf352.

[8] NFFA-EUROPE Project, R. Aversa and M.H. Modarres and S. Cozzini and R. Ciancio, "`100% SEM Dataset`," 2018. https://b2share.eudat.eu/records/f1aa0f5ad38c456eaf7b04d47a65af53.

[9] M. P. Sampat, Z. Wang, S. Gupta, A. C. Bovik, and M. K. Markey, "Complex wavelet structural similarity: A new image similarity index," *IEEE Transactions on Image Processing*, vol. 18, pp. 2385–2401, Nov 2009.

[10] L. Wang, Y. Zhang, and J. Feng, "On the euclidean distance of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, Aug 2005.

[11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.

[12] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, pp. 1–127, Jan. 2009.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[14] S. G. N. Silberman, "Tensorflow-slim image classification model library," 2016. https://github.com/tensorflow/models/tree/master/research/slim.

[15] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative review," *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[16] D. Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'REILY, 2017.

[17] E. Facco, M. d'Errico, A. Laio, and A. Rodriguez, "Estimating the intrinsic dimension of datasets by a minimal neighborhood information," *SCIENTIFIC REPORTS*, vol. 7, no. 1, pp. 1–8, 2017.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[20] M. d'Errico, E. Facco, A. Laio, and A. Rodriguez, "Automatic topography of high-dimensional data sets by non-parametric Density Peak clustering," *ArXiv e-prints*, Feb. 2018.