Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Fisica "G. Occhialini"
Corso di Laurea Triennale in Fisica

Baseline implementation of deep clustering with convolutional autoencoders for medical imaging classification

Relatore: Prof. Marco Paganoni
Correlatrice: Dott. Elisabetta De Bernardi

Relazione della prova finale di:
Filippo Quarenghi
Matricola 802584

Anno Accademico 2020-2021

Acknowledgements

I want to thank prof. Marco Paganoni and Elisabetta De Bernardi for the opportunity they offered me. I thank particularly Elisabetta for her constant presence and guidance throughout the duration of this work. It has been a pleasure working with her. I also want to thank my colleagues at Orobix for continuously inspiring and supporting me, even outside the office.

Finally, I want to thank my family and, in particular, my parents, that constantly and unconditionally supported me throughout my journey.

Contents

Abstract

In this work we propose the baseline implementation of an unsupervised deep clustering approach as a tool for automated image classification. Two different algorithms are tested, DEC [9] and DCEC [6]. Both are constituted by a first stage, in which robust features are learned by training an autoencoder, and by a second stage, in which learned features are encouraged to be cluster-oriented by finetuning the encoder utilizing the prediction of k-means centers for initialization. The approach is tested on a small dataset composed of 2D medical images from 3 different sources (CT, MRI, PET). The methodology intends to be applied for the automated staging and classification of idiopathic pulmonary fibrosis (IPF) on high-resolution CT (HRCT), where early and accurate staging tools are needed. Various deep learning approaches have been proposed for the same task so far, all requiring image annotation for supervised training. Our approach aims to the same goal, but without any annotation or supervision.

Chapter 1

Introduction

In this chapter I'll introduce some core concepts that are necessary to understand the following thesis. From machine learning and deep learning, to convolutions, autoencoders and deep clustering.

1.1 Machine learning and deep learning

A machine learning algorithm is an algorithm that is able to learn from data. For this reason, machine learning allows us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings.

Machine learning tasks are usually described in terms of how the machine learning system should process an example. An example is a collection of features that have been quantitatively measured from some object or event that we want the machine learning system to process. In our case, the example is an image. The features of an image are usually the values of the pixels in the image.

One particular task related to images that interests us is classification. In this type of task, the computer program is asked to specify which of k categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function $f : \mathbf{R}^n \to \{1, ..., k\}$. When $y = f(x)$, the model assigns an input described by vector x to a category identified by numeric code y.

In order to evaluate the abilities of a machine learning algorithm, we must design a quantitative measure of its performance. Usually this performance measure is specific to the task being carried out by the system. For tasks such as classification,we often measure the accuracy of the model. Accuracy is just the proportion of examples for which the model produces the correct output.

Usually we are interested in how well the machine learning algorithm performs on data that it has not seen before, since this determines how well it will work when deployed in the real world. We therefore evaluate these performance measures using a test set of data that is separate from the data used for training the machine learning system.

Machine learning algorithms can be broadly categorized as *unsupervised* or *supervised*. Supervised learning algorithms experience a dataset containing features, but each example is also associated with a label or target. Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset.

In the context of deep learning, we usually want to learn the entire probability distribution that generated a dataset. We are interested in unsupervised learning algorithms that perform other roles, like *clustering*, which consists of dividing the dataset into clusters of similar examples.

Roughly speaking, unsupervised learning involves observing several examples of a random vector x, and attempting to implicitly or explicitly learn the probability distribution $\rho(x)$, or some interesting properties of that distribution.

Deep learning is a subset of machine learning. Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons (MLPs), are the quintessential deep learning models. The goal of a feedforward network is to approximate some function f^*. For example, for a classifier, $y = f^{(x)}$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

Feedforward neural networks are called networks because they are typically represented by composing together many different functions. For example, we might have three functions $f(1)$, $f(2)$, and $f(3)$ connected in a chain, to form $f(x) = f(3)(f(2)(f(1)(x)))$. These chain structures are the most commonly used structures of neural networks. In this case, $f(1)$ is called the first layer of the network, $f(2)$ is called the second layer, and so on. The overall length of the chain gives the depth of the model. It is from this terminology that the name "deep learning" arises. The final layer of a feedforward network is called the output layer. During neural network training, we drive $f(x)$ to match $f^{(x)}$.

The training data provides us with noisy, approximate examples of $f^{(x)}$ evaluated at different training points. Each example x is accompanied by a label $y \approx f^{(x)}$. The training examples specify directly what the output layer must do at each point x; it must produce a value that is close to y. The behavior of the other layers is not directly specified by the training data. The learning algorithm must decide how to use those layers to produce the desired

output, but the training data does not say what each individual layer should do. Instead, the learning algorithm must decide how to use these layers to best implement an approximation of f^*. Because the training data does not show the desired output for each of these layers, these layers are called hidden layers.

Finally, these networks are called neural because they are loosely inspired by neuroscience.

1.2 Convolutional neural networks and autoencoders

1.2.1 Convolutions and convolutional neural networks

Convolutional networks [7], also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include image data, which can be thought of as a 2D grid of pixels. The name "convolutional neural network" indicates that the network employs the mathematical operation convolution 1.1. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

$$s(t) = \int x(a)w(t-a)da \tag{1.1}$$

In convolutional network terminology, the first argument (in Equation 1.1, the function x) to the convolution is often referred to as the input and the second argument (in Equation 1.1, the function w) as the kernel. The output is sometimes referred to as the feature map.

Discrete convolution can be viewed as multiplication by a matrix. An example of 2D discrete convolutions can be seen in the Figure 1.1.

1.2.2 Autoencoders

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h, also called embedded layer, that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$.

If an autoencoder succeeds in simply learning to set $g(f(x)) = x$ everywhere, then it is not especially useful. Instead, autoencoders are designed to be unable to learn to copy perfectly. Usually they are restricted in ways

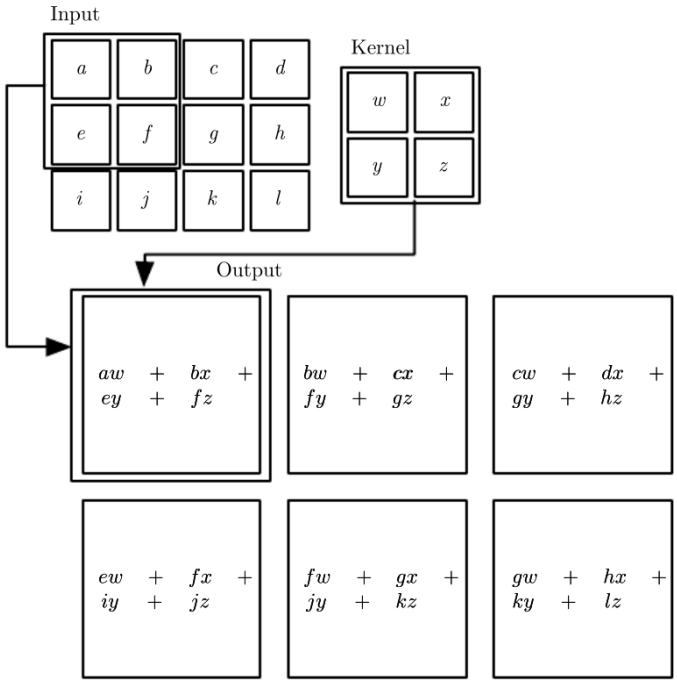

Figure 1.1: Example of 2D convolutions.

that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.

Our model is a convolutional autoencoder, meaning that it follows the structure of an autoencoder (encoder, embedded, decoder) where almost all layers are convolutional layers.

1.3 Deep clustering

Deep clustering algorithms can be broken down into three essential components: deep neural network, network loss, and clustering loss.

The deep neural network is the representation learning component of deep clustering algorithms. They are employed to learn low dimensional non-linear data representations from the dataset. Most widely used architectures are autoencoder based (mostly convolutional).

The objective functions of deep clustering algorithms are generally a linear combination of an unsupervised representation learning loss, here referred

to as network loss L_r and a clustering oriented loss L_c. The total loss is formulated as

$$L = L_r + \gamma L_c \tag{1.2}$$

where $\gamma > 0$ controls the clustering loss. The neural network loss refer to the reconstruction loss of the autoencoder. Some models like DEC [9] discard the network loss altogether in favour of just using clustering loss to guide both representation learning and clustering.

In deep clustering literature, we see the regular use of the following two evaluation metrics:

- **Unsupervised Clustering Accuracy (ACC)**.
  ACC is the unsupervised equivalent of classification accuracy. ACC differs from the usual accuracy metric since it uses a mapping function m to find the best mapping between the cluster assignment output c of the algorithm with the ground truth y. This mapping is required because an unsupervised algorithm may use a different label than the actual ground truth label to represent the same cluster. In particular, the unsupervised clustering accuracy is defined as

  $$ACC = \max_m \frac{\sum_{i=1}^{n} \mathbf{1}\{y_i = m(c_i)\}}{n} \tag{1.3}$$

  where l_i is the ground-truth label, c_i is the cluster assignment produced by the algorithm, and m ranges over all possible one-to-one mapping between clusters and labels.

- **Normalized Mutual Information (NMI)**.
  NMI is an information theoretic metric that measures the mutual information between the cluster assignments and the ground truth labels. It is normalized by the average of entropy of both ground labels and the cluster assignments. It's defined as

  $$NMI(Y, C) = \frac{2 \times I(Y; C)}{H(Y) + H(C)} \tag{1.4}$$

  where Y are the class labels, C are the cluster labels, H is the entropy and $I(Y; C)$ is the mutual information between Y and C.

1.4 Applications in medical physics and previous works

As one of the most popular approaches in artificial intelligence, deep learning has attracted a lot of attention in the medical physics field over the past

few years. Embracing the current big data era, medical physicists equipped with these state-of-the-art tools aim to solve pressing problems in modern radiology.

An example of this is Idiopathic pulmonary fibrosis (IPF), as one the most common type of pulmonary fibrosis. Machine learning and deep learning techniques are applied to High Resolution Computed Tomography (HRCT) for essentially two purposes. The first is to aid the radiologist in the diagnosis of the IPF. This is because the IPF has a specific texture pattern of the lung parenchyma (pattern UIP) that often is very similar to the texture pattern of other interstitial lung tissue diseases. When the diagnosis through images is not certain then it is necessary to resort to a biopsy, which is very invasive for the patient. Some works, like [1], use ML techniques to analyse the HRCT images of interstitial lung diseases to automatically classify the IPF patterns.

A second objective for the application of machine learning is the staging and prognosis of the IPF. Normally this step is done by functional tests, like a spirometry test. [2] tried to created a clinic/radiology model for staging and prognosis that includes parameters extracted from HRCT images. Most of the effort for staging and prognosis is concentrated on semantic segmentation of the lung tissue. The goal is to segment different patterns and feeding feature extracted from these patterns to a machine learning model. Most recently, the semantic segmentation is done by CNN models like [3]. Also, [8] uses a CNN directly for classification.

Our work falls into the second group where we try to approach in an unsupervised way the staging and prognosis of IPF utilising CNN to extract deep radiomics features from HRCT.

Chapter 2

Methods

2.1 Convolutional autoencoders

As mentioned in Chapter 2, an autoencoder is generally composed of two sections, corresponding to the encoder $f_W(\cdot)$ and the decoder $g_U(\cdot)$ respectively. It aims to fit an input sample by minimizing the mean square error (MSE) between its input and output over all samples, i.e.

$$\min_{W,U} \frac{1}{n} \sum_{i=1}^{n} ||g_U(f_W(x_i)) - x_i||_2^2 \tag{2.1}$$

Convolutional AutoEncoders (CAE) are formed by convolutional layers stacked on the input images. These layers extract hierarchical features from the input. The features extracted from the last convolutional layer are then flattened to form a vector, followed by a fully connected layer with only a small number of units (see Chapter 4 for details). This central layer is called the embedded layer. The input 2D image is thus transformed into an a vector of a lower dimension feature space. This vector is then transformed back into a 2D image via a symmetrical model where the convolutional layers are being replaced by convolutional transpose layers.

The parameters of the encoder $h = F_w(x)$ and decoder $x' = G_{w'}(h)$ are updated by minimizing the reconstruction error, defined as

$$L_r = \frac{1}{n} \sum_{i=1}^{n} ||G_{w'}(F_w(x_i)) - x_i||_2^2 \tag{2.2}$$

where n is the number of images in the dataset and $x_i \in R^2$ is the ith image.

The key factor in this implementation is the reduced dimension of the embedded layer. In fact, if the embedded layer is large enough, the network may be able to copy its input to output, leading to the learning of useless

features. Such constraint forces the model to capture only the most salient features of the data. The idea here is to create an autoencoder where the dimension of the embedding is close to the number of clusters that we hope to identify.

This way the network can be trained in an end-to-end manner without the need of any classification labels. The learned embedded representations are proved to be cluster-oriented.

In this baseline implementation of the method we don't utilize any regularization layer such as BatchNormalization or Dropout layers.

2.2 Deep embedded clustering (DEC)

We consider now the problem of clustering a set of n points $x_i \in X_{i=1}^{n}$ into k clusters, each represented by a centroid μ_j, $j = 1, ..., k$. Instead of clustering directly the data space X, the considered method proposes first to transform the data with a non-linear mapping $f_\theta : X \to Z$, where θ are the learnable parameters and Z is the latent *feature space*. In our case the non-linear mapping f_θ is the encoder section of the CAE and the dimension of the feature space is defined by the dimension of the embedded layer.

The considered algorithm (DEC) [9] clusters data by simultaneously learning a set of k clusters centers $\mu_j \in Z_{j=1}^{k}$ in the feature space Z and the parameters θ if the encoder that maps the 2D images into Z. DEC is composed of two phases: firstly we initialize the weights training the CAE, next we optimize the parameters of the encoder (i.e. clustering), where we iterate between computing an auxiliary target distribution and minimizing the Kullback-Leibler (KL) divergence to it.

2.2.1 Clustering with KL divergence

As mentioned above the proposed method aims to improve the clustering using an unsupervised algorithm that alternates between two steps. In the first step, we compute a soft assignment between the embedded points and the cluster centroids. In the second step, we update the encoder mapping f_θ and refine the cluster centroids by learning from current high confidence assignments using an auxiliary target distribution. This process is carried on until convergence is reached.

Soft assignment

The method uses Student's t-distribution as a way to measure the similarity between embedded points z_i and centroid μ_j:

$$q_{ij} = \frac{(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}} \tag{2.3}$$

where $z_i = f(\theta) \in Z$ corresponds to $x_i \in X$ after embedding, α are the degrees of freedom of the Student's t-distribution and q_{ij} cab be interpreted as the probability o assigning sample i to the cluster j (i.e. soft assignment). In all experiments α is set to 1.

KL divergence minimization

The model is trained by matching the soft assignment to the target distribution. The method defines as its objective the minimization of the KL divergence loss between the soft assignments q_i and the auxiliary distribution p_i as follows:

$$L_c = KL(P|Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{2.4}$$

The choice of target distributions P is crucial for DEC's performance.

In our experiments, we compute p_i by first raising q_i to the second power and then normalizing by frequency per cluster:

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij}^2/f_j} \tag{2.5}$$

where $f_j = \sum_i q_{ij}$ are soft cluster frequencies. In particular, this target distribution has the following proprieties: (1) strengthen predictions (i.e., improve cluster purity), (2) puts more emphasis on data points assigned with high confidence, and (3) normalize loss contribution of each centroid to prevent large clusters from distorting the hidden feature space. Please refer to [9] for in depth discussions.

Initialization and optimization

We first pretrain the parameters of CAE to get meaningful target distribution. After pretraining, the cluster centers are initialized with k-means on the embedded features of all images to obtain k initial centroids. Then we discard the decoder section and we jointly optimize the cluster centers μ_j and the parameters of the encoder using Adam. The gradients of L with respect

to the feature space embedding of each data point $\partial L/\partial z_i$ and each cluster centroid $\partial L/\partial \mu_j$ are computed and then passed down to the encoder and used in standard back-propagation to compute the encoder's parameter gradient $\partial L/\partial \theta$. We used an early stopping mechanism to stop the training once less than $tol\%$ of points change cluster assignment between two consecutive iterations.

2.3 Deep Convolutional Embedded Clustering (DCEC)

A possible issue with the previously described approach is that the embedded feature space in DEC may be distorted by only using clustering oriented loss 2.4. To this end, the method described in [6] proposes to add the reconstruction loss of autoencoders to the objective and so optimize 2.2 along with clustering loss 2.4 simultaneously. The autoencoders should preserve the local structure of data generating distribution, avoiding the corruption of feature space.

2.3.1 Structure of Deep Convolutional Embedded Clustering

The DCEC structure is composed of CAE with a clustering layer connected to the embedded layer of the autoencoder. The clustering layer then maps each embedded point z_i coming from the input image x_i into a soft label. The clustering loss L_c is defined as 2.4 between the distribution of soft labels and the predefined target distribution. This way the CAE is used to learn embedded features and the clustering loss guides the embedded features to be prone to forming clusters.

The objective of DCEC is then:

$$L = L_r + \gamma L_c \tag{2.6}$$

where $\gamma > 0$ is a coefficient that controls the degree of distorting embedded space. When $\gamma = 1$ and $L_r \equiv 0$, 2.6 reduces to the objective of DEC [9].

2.3.2 Reconstruction Loss for Local Structure Preservation

As described in Section 3.2.1, DEC [9] abandons the decoder and finetunes the encoder using only the clustering loss 2.2. It's possible however that this

kind of finetuning could distort the embedded space, weakening the representativeness of the embedded features extracted and in so doing hurting the clustering performance. Keeping the decoder and simultaneously optimizing the two objectives should prevent this issue.

It's important to notice that the parameter γ is crucial as it prevents the clustering loss L_c to cause distortion. It's key that on the first iteration of the finetuning stage both the L_c and the L_r contribute with the same intensity to the update of the network weights.

2.3.3 Optimization

As for DEC we first pretrain the CAE by setting $\gamma = 0$ to achieve a meaningful target distribution. As in DEC, the cluster centers are then initialized with k-means on the embedded features of all images. Then γ is set to a value of 0.01 and the weights of the autoencoder, cluster centers and target distribution P as follows: $\partial L/\partial z_i$ and $\partial L/\partial \mu_j$ are computed and the weights and centers are updated by backpropagation and Adam; the target distribution P serves as ground truth soft label but also depends on predicted soft label. Therefore, to avoid instability, P should not be updated at each iteration. In practice we update target distribution using all embedded points every T iterations. As with DEC, the training process terminates when the change of label assignments between two consecutive updates for target distribution is less than a threshold tolerance.

Chapter 3

Experiments

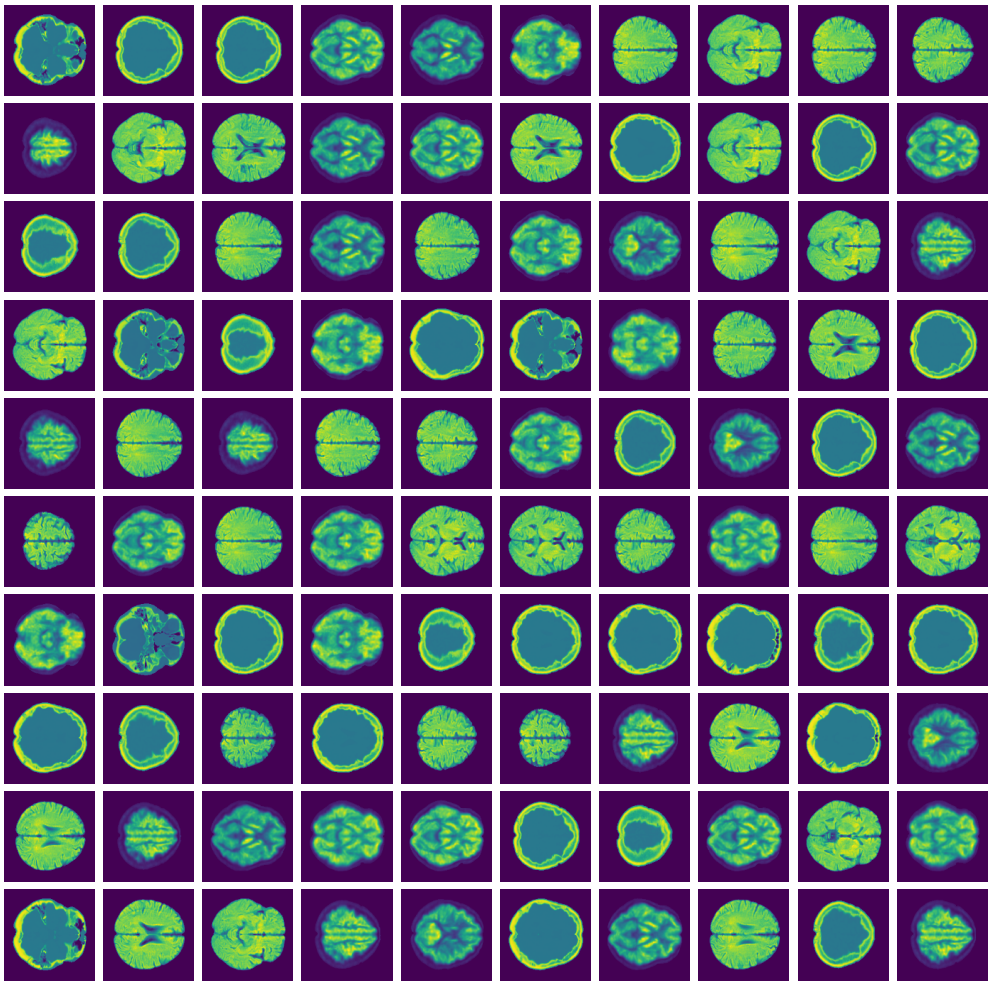

Figure 3.1: Examples of some images from the dataset.

3.1 Dataset

The proposed DEC and DCEC methods are evaluated on a custom dataset composed of images from three different medical imaging source: CT, MRI and PET. These are 2D gray-scale images of the head of dimension 128x128 pixels, see 3.1 for some examples. The dataset is being divided into training, validation and testing datasets with 70:10:20 ratio. The number of images composing the dataset is reported in Table 3.1.

Images	**Training**	**Validation**	**Testing**	**Total**
CT	213	35	106	354
MRI	213	35	106	354
PET	198	29	56	283

Table 3.1: Number of images in the dataset.

3.1.1 Data preparation

All the images were coregistered to a common atlas in order to reduce variability and help training. Also, all images have been normalized in intensity. No data augmentation has been used during our experiments.

3.2 Experiments setup

We investigated the performance of DEC and DCEC methods and compared them to a basic k-means on the features extracted after pretraining the CAE. The CAE is symmetrical and the encoder part is composed of three 2D convolutional layers, one flatten layer and the embedded fully connected layer. The decoder part reflects the encoder and the transpose convolutional layers substitute the convolutional ones. We utilize convolutional layers with stride instead of convolutional layers followed by pooling layers in the encoder, and convolutional transpose layers with stride in the decoder. This because the convolutional (transpose) layer with stride allow the network to learn spacial subsampling (upsampling) from data, leading to higher transformation capability. After preliminary testing (not shown) we choose the following architecture, see the 3.2 for all the details.

Layer	Dimensions	Filters	Kernel size	Stride	Params
Input	128 x 128	1	None	None	0
Conv1	32 x 32	16	5 x 5	4	416
Conv2	8 x 8	32	5 x 5	4	12832
Conv3	4 x 4	32	3 x 3	2	9248
Flatten	512	None	None	None	0
Embedded	3	None	None	None	1539
Linear	512	None	None	None	2048
Reshape	4 x 4	32	None	None	0
Deconv3	8 x 8	32	3 x 3	2	9248
Deconv2	32 x 32	16	5 x 5	4	12816
Deconv1	128 x 128	1	5 x 5	4	401

Table 3.2: Convolutional AutoEncoder (CAE) architecture.

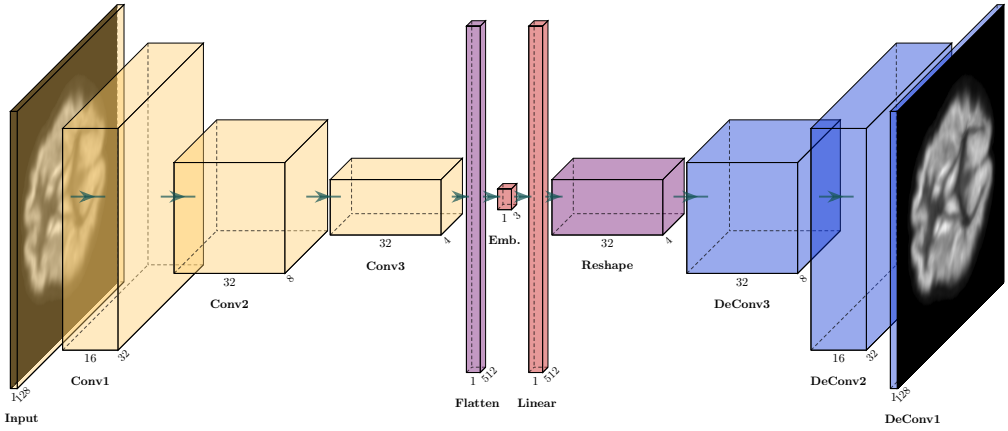

Figure 3.2: CAE architecture.

The embedding layer dimension of the CAE for both DEC and DCEC is initially set at 3. The architecture for DEC is then described in table 3.3. The architecture of DCEC is the same as CAE with the clustering layer connected to the Embedded layer. During our experiments we also tried to change the dimension of the embedded layer. See below for the results.

Layer	Dimensions	Filters	Kernel size	Stride	Params
Input	128 x 128	1	None	None	0
Conv1	32 x 32	16	5 x 5	4	416
Conv2	8 x 8	32	5 x 5	4	12832
Conv3	4 x 4	32	3 x 3	2	9248
Flatten	512	None	None	None	0
Embedded	3	None	None	None	1539
Clustering	3	None	None	None	3

Table 3.3: DEC architecture.

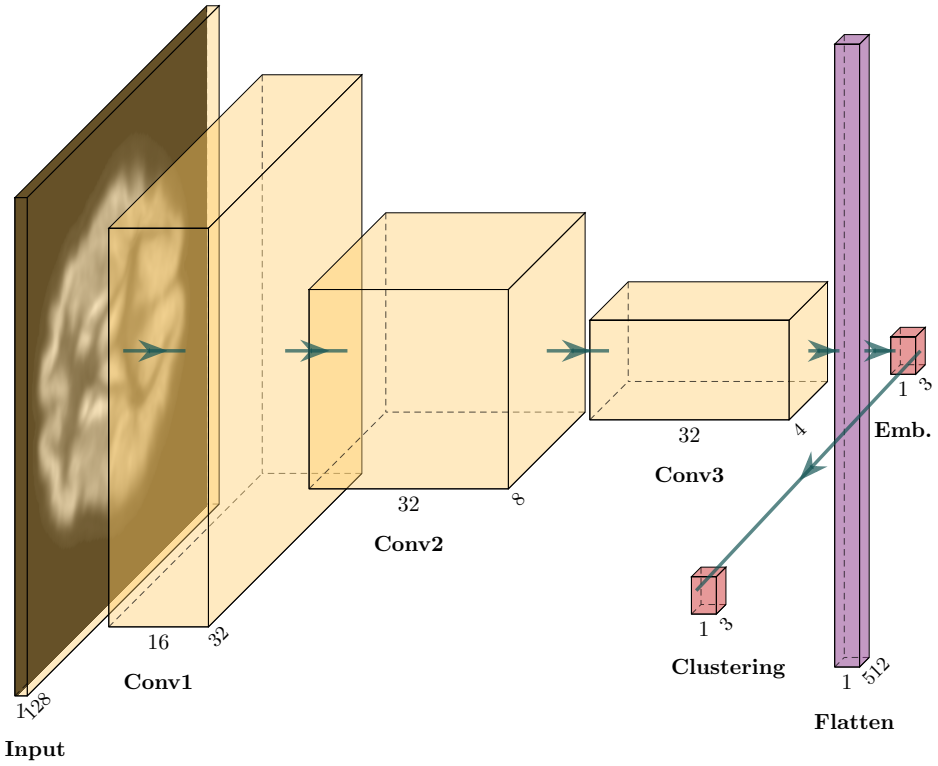

Figure 3.3: DEC architecture.

Evaluation metrics

All clustering methods are evaluated by Unsupervised Clustering Accuracy (ACC) and Normalized Mutual Information (NMI) which are widely used in unsupervised learning scenarios, as previously mentioned in Section 1.3.

3.3 Results

3.3.1 Pretraining CAE

First we report the results from the pretraining. The autoencoder, even with such a small dimension for the embedded layer, was able to reconstruct, albeit not perfectly, the input image. Below, in Figure 3.5 we can see on the left of each pair the original image and on the right the one reconstructed. Also, looking at the training and validation error during training, reported in the Figure 3.4, we are confident that there was no overfitting on the training data.

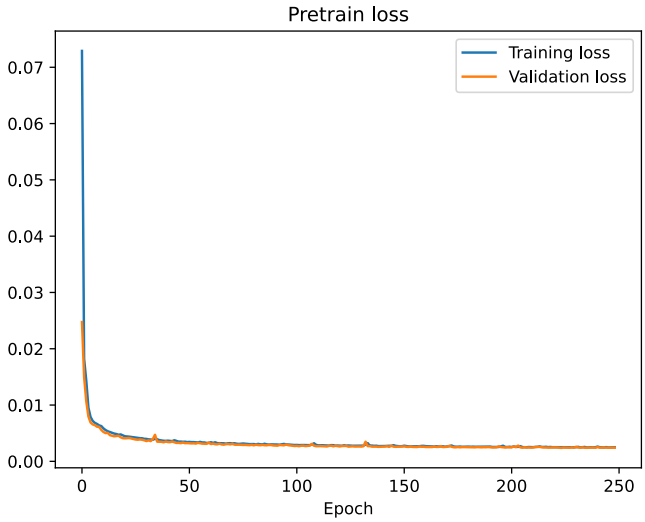

Figure 3.4: Pretraining MSE error for the autoencoder with embedding space dimension 3. Lowest values: `train loss:0.0025`, `val loss:0.0024`

We report then in Table 3.4 the values of the two metrics we are considering to evaluate the results. That is, we performed a k-means initialization on the features from the embedded layer and calculated the ACC and NMI metrics.

Method	ACC	NMI	MSE error
k-means+CAE	0.622	0.392	0.0024

Table 3.4: Results after pretraining. Embedded dimension 3.

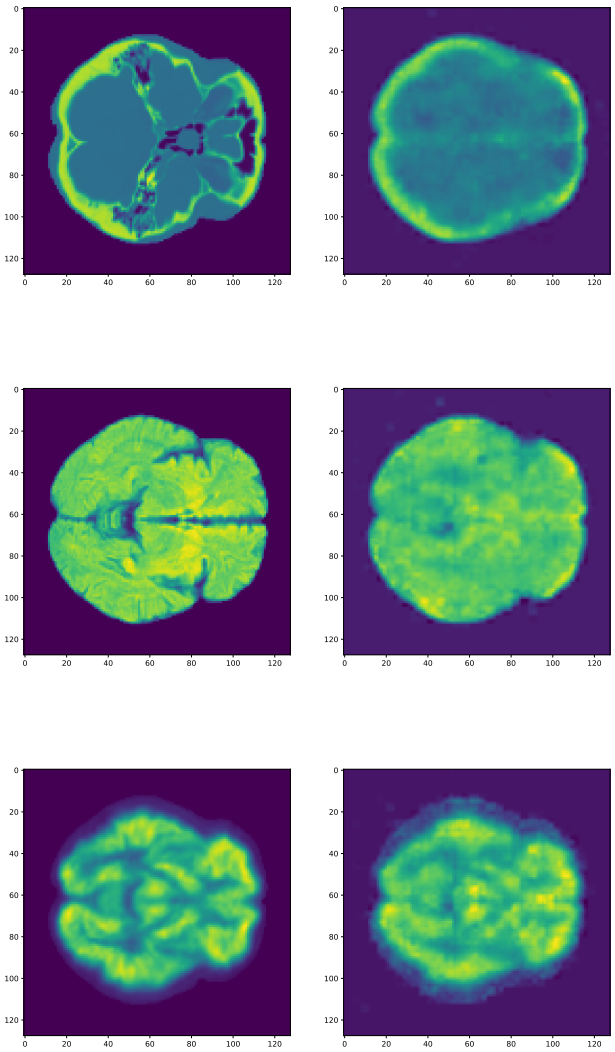

Figure 3.5: On the left the original image, on the right the image predicted by the CAE.

We also plotted in Figure 3.6 the results from the TSNE and UMAP projection of the embedding space onto a 2D plane to judge the quality of the clustering. While from the TSNE we can see that the clustering is not perfect but the three classes are clearly separated in different regions of the space, the UMAP plot shows how the points are not perfectly divided into the Voronoi cells. This is expected from an accuracy around 50%.

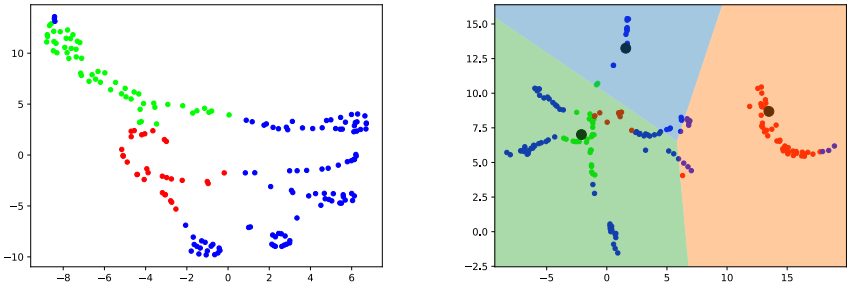

Figure 3.6: UMAP with Voronoi cells and TSNE projections from 3-dimensional features from the *pretrained autoencoder* embedded layer.

Finetuning DEC and DCEC

As mentioned before we tested two different finetuning methods, DEC and DCEC. The results from these tests are shown in Table 3.5. We can clearly see that DEC achieves the best *accuracy* and *NMI*. The DCEC doesn't improve upon the results from the CAE. This is probably due to the fact that the reconstruction of the image is not of enough quality for the algorithm to be able to perform flawlessly. Nether the less, the reconstructed images are not worst than the one produced after pretraining. The prediction from DCEC are shown in Figure 3.8.

Method	ACC	NMI	MSE error	KL div
DEC	**0.805**	**0.613**		0.03694
DCEC	0.638	0.412	0.00171	0.11211

Table 3.5: Comparison of results of all methods. Embedded dimension 3.

Regarding the quality of the clustering, in Figure 3.7 we can see the results of TSNE and UMAP projections for both the DEC and DCEC. These clustering plots reflects the results from table 3.5.

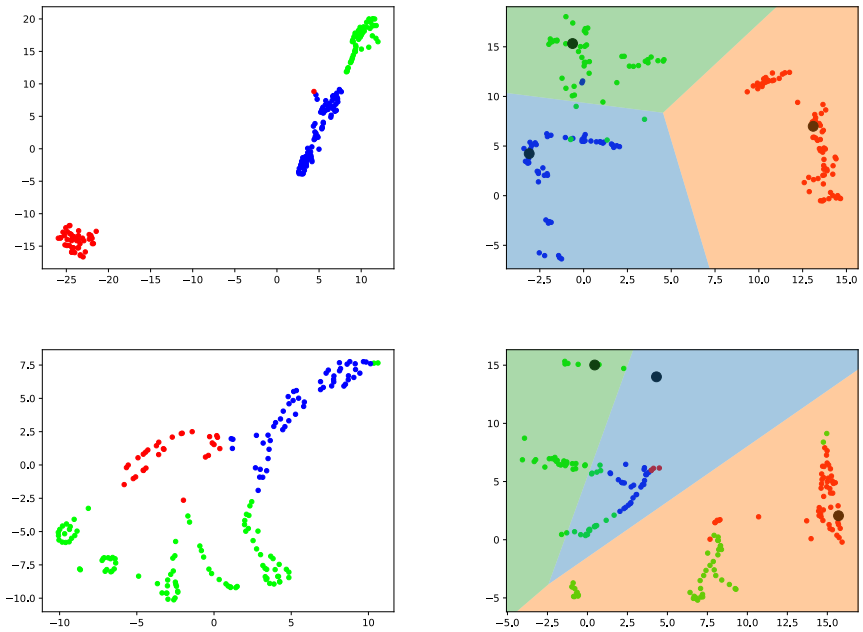

Figure 3.7: UMAP with Voronoi cells and TSNE projections from 3-dimensional features from the *finetuned autoencoder* embedded layer. DEC on the first row, DCEC on the second one.

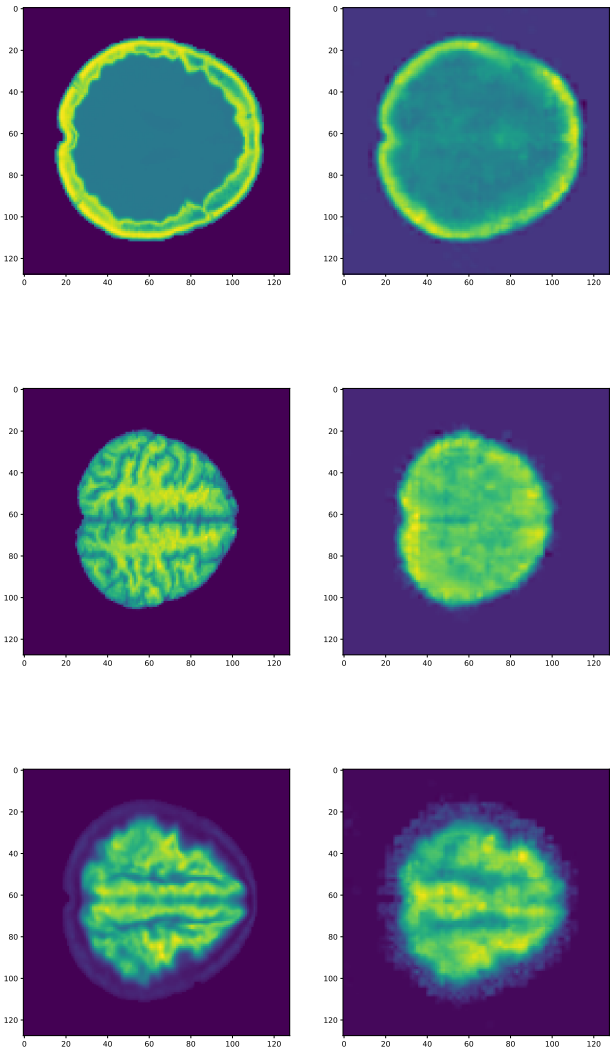

Figure 3.8: On the left the original image, on the right the image predicted by the DCEC.

3.3.2 Different architectures for the autoencoder

We then performed a series of tests varying the dimension of the embedding space, in particular we tested autoencoders with 10 and 30 dimensions for the embedded layer, leaving the network architecture unchanged. The Table 3.6 shows the results we obtained.

Method	ACC	NMI	MSE error	KL div	Embedded dim
k-means+CAE	0.656	0.491	0.00170		
DEC	0.659	**0.548**		0.06359	10
DCEC	**0.665**	0.498	0.00311	0.09366	
k-means+CAE	0.578	0.414	0.00130		
DEC	**0.654**	**0.530**		0.06445	30
DCEC	0.605	0.458	0.02312	0.08776	

Table 3.6: Results from test with different embedding space dimension.

Pretraining

Comparing the results from the pretraining phase, we can see how the accuracy increases when we go from 3 to 10 for the dimension of the autoencoder, while there's a drop from 10 to 30. The better performance with dimension 10 is clearly explainable by the fact that with this architecture the autoencoder can more easily reconstruct the image. The lower performance from the architecture with dimension 30 can be explained remembering that if the embedded layer is too large the autoencoder can learn feature that are not characteristic but that could be related to the noise in the images. In particular, for our autoencoder an embedded layer with dimension 30 is too large for this particular dataset. This can be seen in the plot in Figure 3.10. Also, the MSE error values confirm these results.

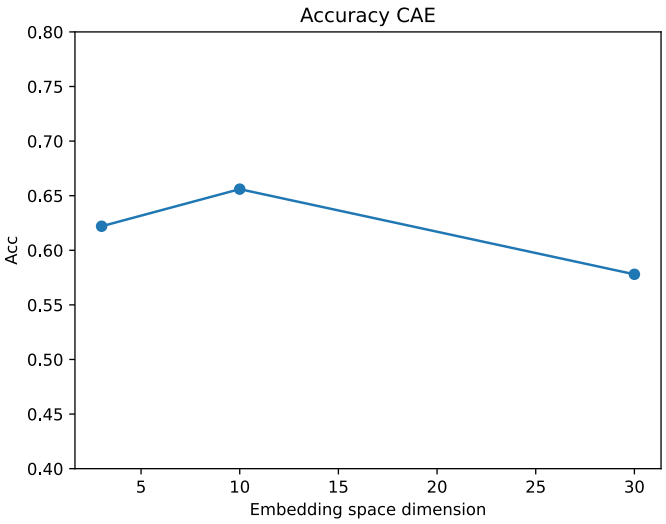

Figure 3.9: Accuracy of CAE for different dimensions of embedded layer.

Finetuning

Regarding the results from the finetuning stages, the best results for DEC are achieved in the first experiment with the embedded with dimension 3. This is also reported in [9], where it's shown that the embedded with dimension equal to the number of clusters is the one that performs the best.

The results for DCEC show that the best performance is achieved with embedding dimension equal to 10. This is probably due to the fact that the reconstruction is helped by the larger number of nodes. The performance for the autoencoder with embedding space dimension 30 is lower than the one with dimension 3 or 10 and this is, as mentioned earlier, explained by the fact that the autoencoder has mapped to the embedding space some features that are not representative for the input image.

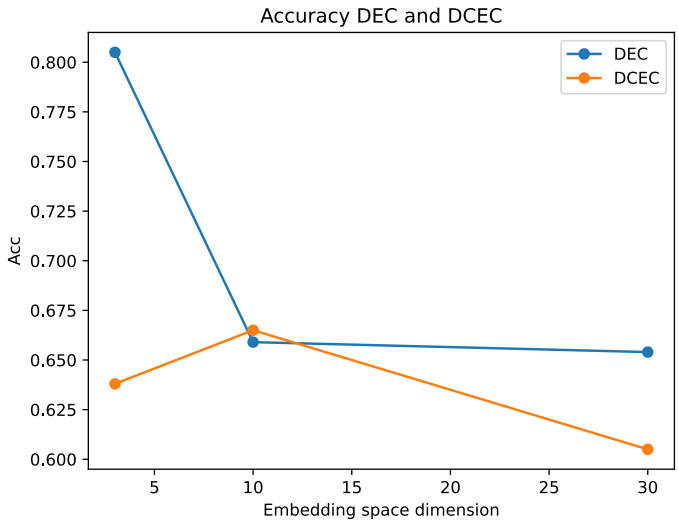

Figure 3.10: Accuracy of DEC and DCEC for different dimensions of embedded layer.

The NMI metric follows the same trend as the accuracy both for the DEC and DCEC finetuning stages.

Chapter 4

Conclusions

In this thesis we tested two methods, DEC [9] and DCEC [6], as clustering algorithms to classify in an unsupervised way images from three different classes (CT, MRI and PET).

Firstly, we did not report the results, but we carried on extensive experiments to find the optimal autoencoder architecture that would allow us to both reconstruct the images and keep the smallest possible dimension for the embedded layer. Also, the chosen autoencoder is the best because of its depth. In fact, a deeper autoencoder performed worst because the feature we are interested come from the texture of the image, not the complex structures that a deep autoencoder might learn. Moreover, a shallower autoencoder could not map effectively the images into the embedding space.

Regarding the finetuning stage, DEC has proven to be the best approach for our case. The key to the success of this method is in the finetuning stage where it discards completely the decoder and focuses solely on the clustering metric. DCEC didn't reach the same performance and we think that this is due to the fact that the small number of dimensions for the embedded layer drastically compromises the approach. This is the main fault for this approach: the limited number of classes impedes the achievement of better performances.

A future development for this experiments would be to test more approaches related to DEC [9]. We are currently testing the APCS-DA method [5] from X. Guo et al., where data augmentation and self-pacing strategies are used in order to achieve better performances for the clustering accuracy. An other further step would be to start implement the current state of the art in self-supervised learning with contrastive learning, such as [4].

Bibliography

[1] Andreas et al. "Computer-Aided Diagnosis of Pulmonary Fibrosis Using Deep Learning and CT Images, Investigative Radiology". In: Volume 54 (2019), pp. 627–632.

[2] Lee et al. "Prediction of survival by texture-based automated quantitative assessment of regional disease patterns on CT in idiopathic pulmonary fibrosis." In: *Eur Radiol* 28 (2018), pp. 1293–1300.

[3] Marios Anthimopoulos et al. "Semantic Segmentation of Pathological Lung Tissue With Dilated Fully Convolutional Networks". In: *IEEE Journal of Biomedical and Health Informatics* 23.2 (2019), pp. 714–722. DOI: 10.1109/JBHI.2018.2818620.

[4] Jean-Bastien Grill et al. *Bootstrap your own latent: A new approach to self-supervised Learning*. 2020. arXiv: 2006.07733 [cs.LG].

[5] Xifeng Guo et al. "Adaptive Self-Paced Deep Clustering with Data Augmentation". In: *IEEE Transactions on Knowledge and Data Engineering* 32 (2020), pp. 1680–1693.

[6] Xifeng Guo et al. "Deep Clustering with Convolutional Autoencoders". In: *Neural Information Processing*. Ed. by Derong Liu et al. Cham: Springer International Publishing, 2017, pp. 373–382. ISBN: 978-3-319-70096-0.

[7] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[8] Simon L F Walsh et al. "Deep learning for classifying fibrotic lung disease on high-resolution computed tomography: a case-cohort study". In: *The Lancet Respiratory Medicine* 6.11 (2018), pp. 837–845. ISSN: 2213-2600. DOI: https://doi.org/10.1016/S2213-2600(18)30286-8. URL: https://www.sciencedirect.com/science/article/pii/S2213260018302868.

[9] Junyuan Xie, Ross Girshick, and Ali Farhadi. *Unsupervised Deep Embedding for Clustering Analysis*. 2016. arXiv: 1511.06335 [cs.LG].