Brain Cells Phenotyping Via Unsupervised Machine Learning With Autoencoder and Clustering

Nikolay Pashov

under the direction of

Webster Guan PhD
Chung Labs
Massachusetts Institute of Technology

Research Science Institute
February 18, 2023

Abstract

In brain research, an important step after staining and imaging the different brain cells is their phenotyping. In this research, we work with patches of images, taken with CLARITY [1] and aim to develop an unsupervised neural network, consisting of autoencoder and clustering layers, to classify if the input image patch contains a cell or not. So far, the cell types which we have tested the algorithm on, are Microglia and Inhibitory neurons. We feed both the patches to an autoencoder with a small latent space, so that it can learn to extract the most fundamental information about the input, including if it contains a cell. We perform clustering on this compressed data and expect that the encoded vectors of "positive" and "negative" samples would differ from each other and thus form clusters in space. We run K-Means clustering on the latent space's output. The results so far do not prove that the compressed negative and positive samples form distinct clusters, however, the positive samples gather close to each other in space, which shows that the pictures with cells show similar features that the autoencoder is indeed learning. To make use of that and to reach a state where clusterization is possible, we are implementing and integrating other clustering algorithms and approaches, such as outliers removal, that could lead to an improvement of results.

Summary

To determine the type of each cell in the brain is of big importance for brain exploration, since this provides more detailed information about its structure and the processes that take place in it. The research is accelerated if this phenotyping is automated by a computer. In this paper we are implementing an unsupervised machine learning method for this task. The type of network we use is an autoencoder. Specific about this type of neural network is that it takes some data as input, compresses it to a very small representation of the original data and then tries to recreate the input, using only the compression. By doing that, the autoencoder is learning the most important information of the data. We use this feature to classify different type of cells. We rely on the fact, that the autoencoder will learn to determine if there is a cell on a given image or not. Then we use clustering to distinguish the compressed representations of images with and without cells. The advantage of this unsupervised method is that we do not need to label all the samples, thus saving a lot of time. While the results so far do not prove that the compressed data is distinguishable enough to be used for classification, we have grounds to believe that other algorithms we plan to implement will lead to success. These include training the autoencoder and the clustering simultaneously [2], which will adapt the clustering to the specifics of our data, or making the size of the compressed data smaller. This will force the autoencoder to learn only the most important features of the input.

1 Introduction

With the rapid development of science and technology, we are getting closer and closer to knowledge that has been previously considered unreachable. Such is the knowledge about the human brain. Discoveries in this field provide wider understanding of the human intelligence and allow researchers to find cures for brain diseases. After images of the brain and its cells are taken, it is important to know the phenotype of each cell to start doing research. This is why cell plays a key role in the field and also can save a lot of time for researchers if it is automated and unsupervised.

1.1 A cell phenotyping approach

A popular approach for brain research entails cutting the brain into small pieces and then investigating it. However, the slicing takes time, it damages cells, and also leads to loss of depth information about the brain [3], [4]. CLARITY [1] is a method for making the brain tissue transparent by removing the cell's lipids. It also preserves the cells and proteins and fixes them. This way, no cells are damaged and the depth information is sustained. In this current paper we develop a software that labels the different types of cells with pictures from CLARITY (Note: The pictures from Clarity are 3D, but the images we use are a cross-section of the XY-plane).

Markers are used to stain different cell types with different colors. An example of a marker is Syto-16, which marks cell nuclei in the brain. On the images we work with, cell nuclei are labeled as red. Other examples include PV, which marks every inhibitory neuron cell (green on the images in the paper) and IBA-1, which stains each microglia cell (green on the images in the paper). In order to label the cells, we take an image of a region of the brain with two excitation wavelengths, producing two image channels - one, where all nuclei are visible (marked with Syto-16 and in red) and another one, where a specific cell type (for

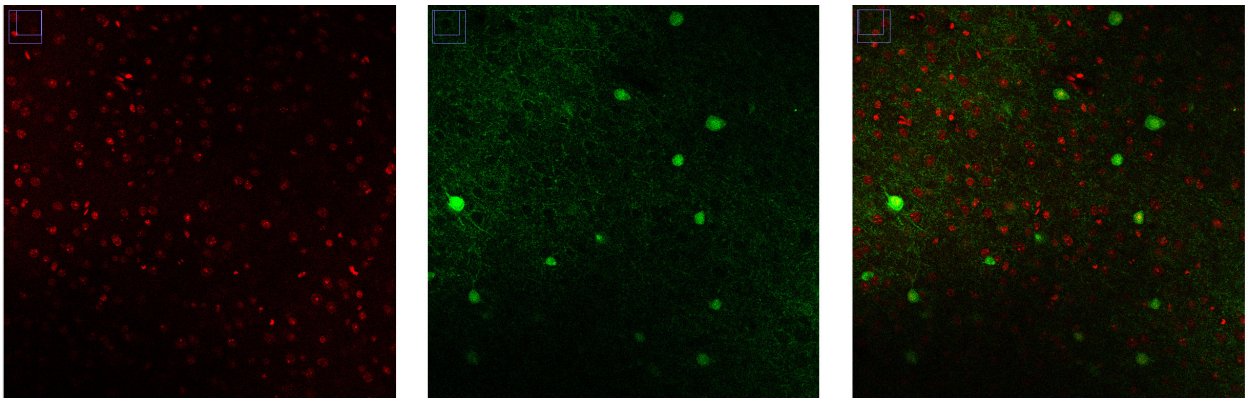

Figure 1: The first pictures contains all nuclei in the region, marked with Syto-16 and labeled red. The second one contains all the inhibitory neuron cells marked with PV and labeled green. On the third picture we have merged both and we observe that some red and green bodies are overlapping. A nuclei from the first picture that overlaps with a neuron from second is possibly is the nuclei of this neuron. Note: The original images are three dimensional, the current representation is a cross-section of the XY-plane

example the inhibitory neurons, stained with PV) is marked. An example for that could be to have inhibitory neurons as the second channel. If a nuclei from the first channel (stained with Syto-16) has similar coordinates on the image as an inhibitory neuron from the second channel, it possibly means that this is the neuron's nuclei (Fig. 1).

The complexity of this task comes from the fact that often more than one nuclei overlaps with the same neuron, this making it difficult to determine which one is the neuron's nuclei. This problem can be easier to see when working with other type of brain cells, such as the microglia. Its shape is more complex and often overlaps with more than one nuclei from the other channel (Fig. 2). Fig. 3 contains examples of more nuclei overlapping with the same cell.

1.2 Previous work

Both a trivial computer vision and a supervised machine learning approach are known for dealing with the problem. Theoretically, the images can be processed with traditional computer vision algorithms like thresholding [5], connected components analysis [6], blob

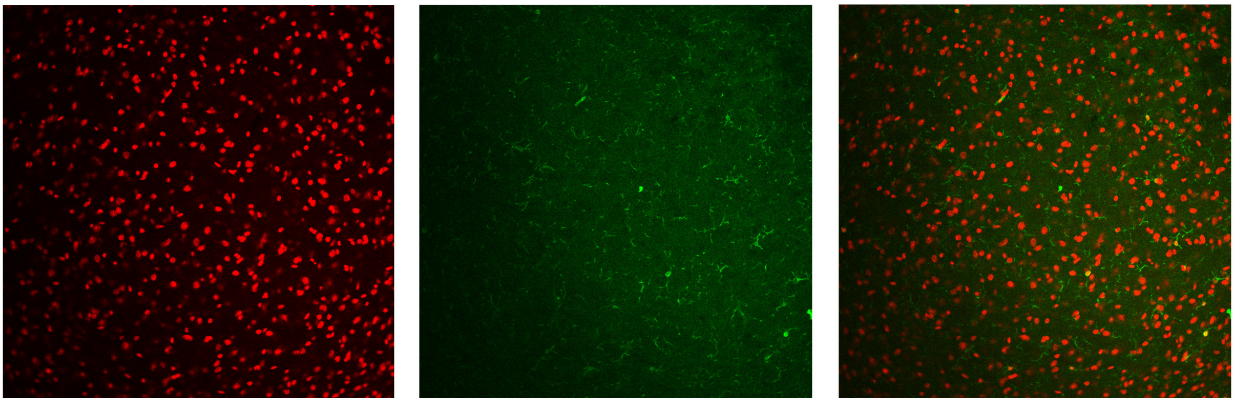

Figure 2: The first picture contains all nuclei in the region, marked red. The second one contains all the microglia cells, marked green. On the third picture we have merged both and we observe that some red and green bodies are overlapping. A nuclei from the first picture that overlaps with one from the second is possibly a microglia cell. Note: The original images are three dimensional, the current representation is a cross-section of the XY-plane

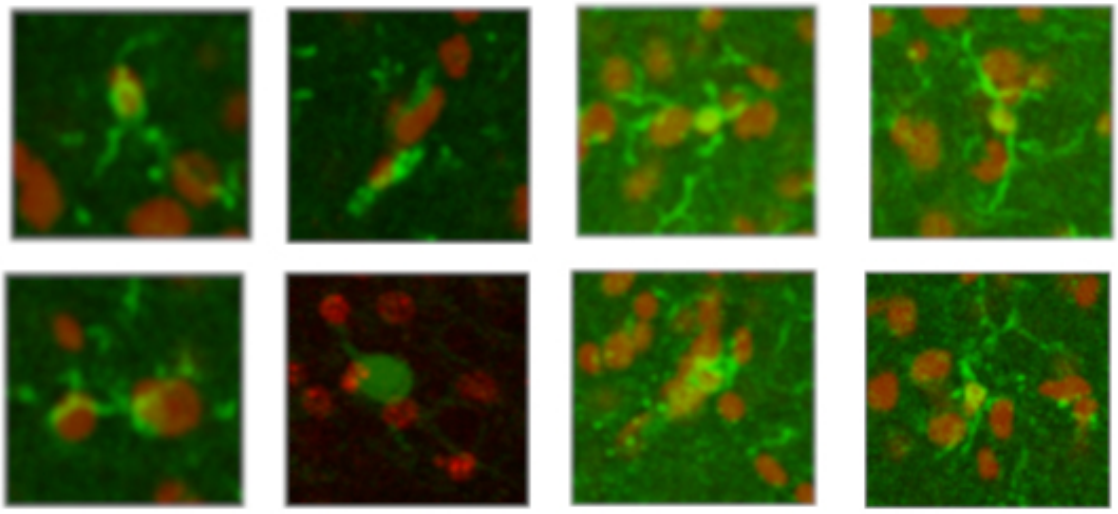

Figure 3: Sample patches of pictures of the two channels, where a specific cell type overlaps with more than one nuclei. The cells types are inhibitory neurons and microglia, marked in green; the nuclei are marked in red.

detection [7] and morphological operations [8]. However, these methods require a lot of parameter tuning, such as choosing the threshold values, the size of the windows by some adaptive thresholding methods, and the size of the filters by morphological transformation. These are slow and often inefficient, since using one setting of the parameters is inappropriate for a wide range of pictures, where the shapes, sizes, colors and locations of the objects are different.

Instead, the solution can be automated using machine learning. A supervised convolutional deep network has been previously trained and designed for this specific task. The results are promising, but a disadvantage of supervised learning is that it takes a lot of time to prepare proper ground truth data.

Unsupervised learning allows us to avoid these problems; It does not require any data labeling for the training, since it is learning how to recreate the given input. We attempt to use unsupervised learning to make the process of cell phenotyping more automated.

1.3 Autoencoders and clustering

Autoencoders are unsupervised neural networks which consist of an *encoder* and a *decoder*. The encoder aims to extract important information from the input as it reduces the data's dimensions and feeds it through a low-dimensional layer, called the latent space. The decoder then takes the latent space's output and its goal is to recreate the original data, using only the compressed representation in the latent space (Fig. 4).

Our goal is to train the network to compress all the important data, so that we can use the latent space's output to run clustering algorithm. We expect that the compressed information of a positive sample will appear different than the compressed information of a negative sample. If true, the encoded positive samples should appear close to each other in a multi-dimensional space. The negative samples should behave similarly. Moreover, the two groups should form clusters relatively away from each other. We then use a clustering algorithm to

identify and later to classify new inputs as we measure closer to which of the clusters the new input appears. For the clustering we use K-means clustering [9] and we use a custom loss function, proposed by Bert De Brabandere et al. [10], as well as a new approach for training the clustering and the autoencoder simultaneously, as proposed by Bo Yang et al. [2].

1.4 Implemented methods

By now, in this work we have tried two different patterns for the layers in the autoencoder. Both of the patterns include a traditional sequence of layers, commonly used in convolutional networks for image classification. It consists of using sequences of convolutional layers with activation functions and layers for changing the dimensionality. We used this sequence and tried two types of pairs of dimensionality changing layers, which are described in the methods section, as well as the parameters tuning. We then use the data from the autoencoder's latent space as an input for K-Means clustering.

2 Methods

As a preprocessing step, all the nuclei from the first channel (the Syto-16 channel) are detected with a blob detection algorithm. Then from both channels (the nuclei channel and the channel with the cells) we make a 32x32 image patch, the center of which is a detected nucleus' center. We define the patch as "positive" if, when applied on the second layer, it contains a cell of a specific type, and as "negative" otherwise. A positive sample would mean that the phenotype of the nucleus is the cell type, which is present on the second channel. To investigate this, we perform classification on the image patch, using autoencoder and clusterization. As input to the autoencoder, we use either both channels of the patch (3D input), or just the channel with the specific cell types (2D input). The described work is

	PV	IBA-1
positive	396	296
negative	13,174	9835

Table 1: Number of samples for each specific cell type. The marker PV stains ihnibitory neuron cells, IBA-1 stains for microglia.

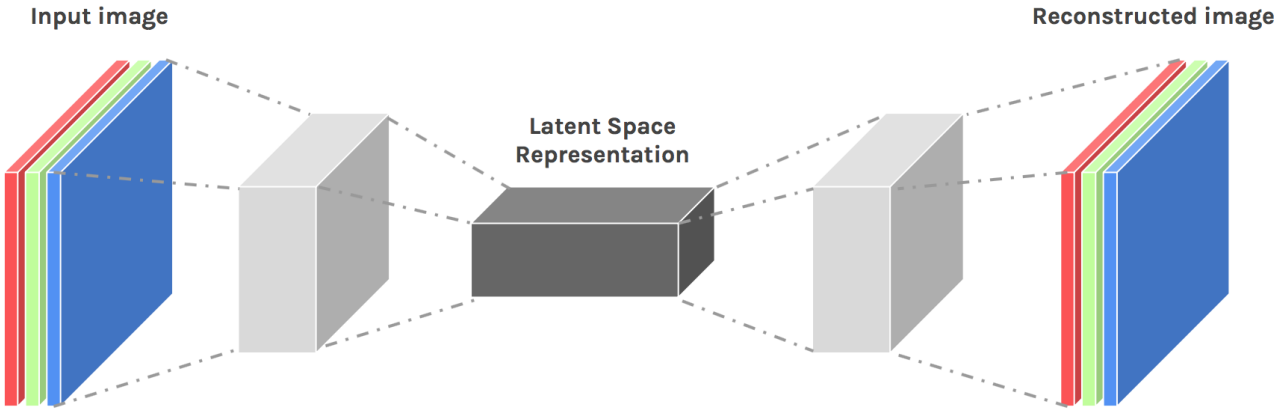

Figure 4: A convolutional autoencoder. Operations are applied on the layers to reduce dimensions in the encoder, leading to a compressed latent space and operations to increase dimensions are applied in the decoder. [12]

performed, while using the 2D input. The samples we train and test with are shown on Table 2.

We want to achieve a well trained encoder, which reduces the dimensionality of the input as much as possible for the latent space, thus easing the clustering. The reasons why this makes the clustering easier are explained in the work of Steinbach et al. [11].

2.1 Designing the autoencoder

2.1.1 MaxPooling and UpSampling

A common layer pattern for convolutional networks is

$$INPUT \rightarrow [[CONV \rightarrow RELU] * N \rightarrow POOL?] * M$$

[13], where:

- *CONV* is a convolutional layer. Its input is the output of a convolution, applied on the previous layer. The amount of filters, used in the convolution and their sizes are adjusted manually when designing the network. Since we want the dimensionality reduction to be performed by the POOL layer, we apply padding around the matrix to manipulate the size of the output matrix.

- *RELU* is a fully-connected convolutional layer with the ReLU activation function [14]. This activation is one of the most commonly used. It is calculated as:

  $$f(x) = x^+ = \max(0, x)$$

  where x is a neuron's output.

- *POOL* is a Pooling layer. It applies a function to reduce the dimensionality of the input. The function we used is MaxPooling, which applies the max function in a neighbourhood of elements.

- $*$ indicates repetition

- *POOL*? indicates an optional pooling layer

- $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$

For the Decoder we used the pattern

$$CONV \rightarrow RELU \rightarrow UPSAMPLING$$

.

The POOL layer is replaced with UPSAMPLING layer, that aims to increase the dimensionality of the input by copying the value of an element and creating a set of elements with that value.

2.1.2 Convolution with Strides

Unfortunately, this pattern didn't perform well on recreating the input images (See section 3 for explanation). The second network we designed consisted of the layer pattern

$$INPUT \rightarrow [[CONV \rightarrow RELU] * N \rightarrow CONV\_WITH\_STRIDES?] * M$$

for the encoder and

$$INPUT \rightarrow [[CONV \rightarrow RELU] * N \rightarrow TRANSPOSED\_CONVOLUTION?] * M$$

for the decoder, where:

- $CONV\_WITH\_STRIDES$ is a normal $CONV$ layer, but we also apply strides (see Fig. **??**), so that some data is taken out. By doing so, we decrease the dimensionality and force the network to learn with reduced amount of parameters.
- $TRANSPOSED\_CONVOLUTION$ performs, again, a normal convolution, but paddings are added and the convolution filters are applied on the values from the paddings, thus increasing the dimensionality.

2.1.3 Adjustment of layer dimensions

In order to compensate for the loss of information from the dimensionality reduction layers, and still have a low-dimensional latent space, we increase the depth of the matrix, when we reduce the width and height. In other words, when we apply convolution with strides or pooling to reduce the matrix size, we balance the information loss by using more filters. This way, we effectively reduce the size of the sample and also provide a way for the network to learn more about the compressed data. Table 2 outlines the architecture we have used. Important to notice is that by the encoder (the layers before the latent space), we try to put the dimensionality reduction layers at the beginning of the network where not that much information from the sample has been lost. When we reach the decoder (the part after the latent space), we put the dimensionality increasing layers at the top, where the depth of

Layer type	# of filters	Filter size	Activation	Padding	Strides	Shape
Conv	2	2,2	relu	same	no	32,32,2
Conv	16	2,2	relu	same	no	32,32,16
Conv	16	2,2	relu	no	2,2	16,16,16
Conv	32	2,2	relu	same	no	16,16,32
Conv	64	2,2	relu	mo	2,2	8,8,64
Conv	64	2,2	relu	same	no	8,8,64
Conv	64	2,2	relu	no	2,2	4,4,64
Conv	64	2,2	relu	same	no	4,4,64
Conv	256	2,2	relu	no	4,4	1,1,256
Latent Space; Conv	64	2,2	relu	same	no	1,1,256
Transponsed_Conv	256	4,4	relu	no	no	4,4,256
Conv	128	2,2	relu	same	same	no,128
Transponsed_Conv	128	2,2	relu	no	2,2	8,8,128
Conv	64	2,2	relu	same	no	8,8,64
Transponsed_Conv	64	2,2	relu	no	2,2	16,16,64
Conv	64	2,2	relu	same	no	16,16,64
Transponsed_Conv	64	2,2	relu	no	2,2	32,32,64
Conv	64	2,2	relu	same	no	32,32,64
Conv	32	2,2	relu	same	no	32,32,32
Conv	16	2,2	relu	same	no	32,32,16
Conv	2	2,2	relu	same	no	32,32,2

Table 2: An example of the used architecture. "Same" for padding means that there is enough padding to keep the width and height of the matrix the same after the convolution.

the layers is big so that the most information is encoded. This way, by slowly decreasing the sample size and simultaneously increasing the dimensions, we can make the dimensionality reduction smoother, and give the encoder more time to learn before it compresses the data for the latent space.

2.2 Integration of clustering algorithm

A working autoencoder is an important step towards reaching our goal, but it alone cannot do classification. To complete the classification we use a clustering algorithm. It takes as input the compressed data from the latent space and segments the data points (Fig. 5), based on the distances between them in the high-dimensional space, which in the

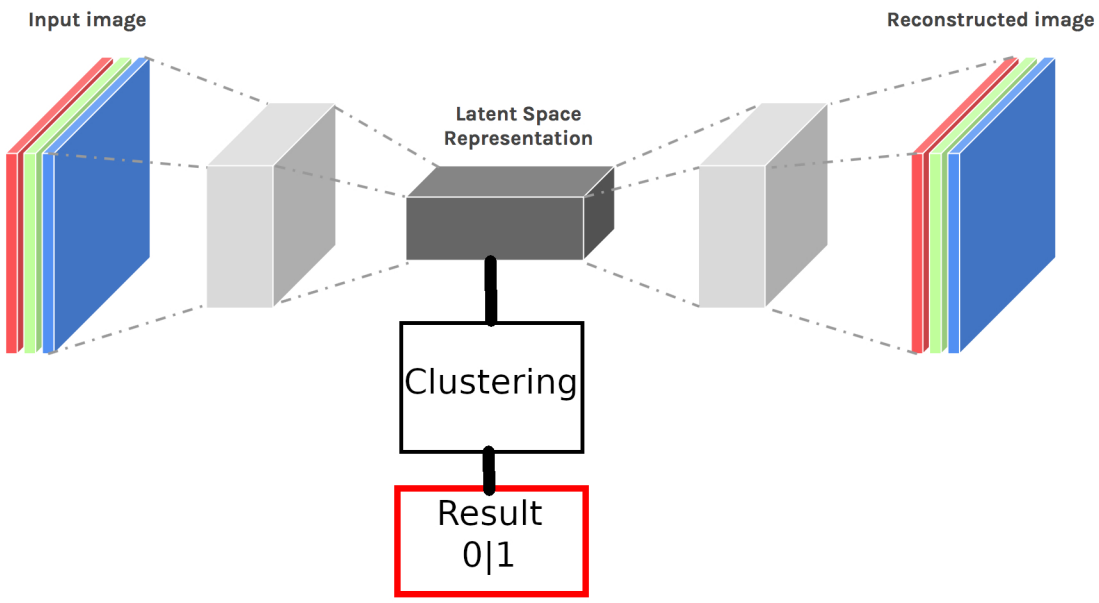

Figure 5: The clustering algorithm runs on the latent space's output, and classifies the data. It relies on the fact that the positive samples' compressed representation should differ strongly from the negative samples' and therefore form a cluster on a different place in the multidimensional space, where the positive samples' data points are close to each other.

case is the dimensionality of the latent space. It relies on the fact that whether a sample is negative or positive should be encoded in the compressed representation in the latent space. Therefore, positive samples should appear close to each other in the high-dimensional space and far from the negative samples. If these conditions are satisfied, the clustering algorithm should succeed. We used K-means clustering without any initialized data points.

2.3 Removing Data Outliers

One of the characteristics of our dataset that could disturb the learning process is the small ratio of positive and negative samples. To eliminate this, we have started implementing a method for automatically removing some of the negative samples. It depends on the fact that after a successful training, the positive samples form a dense group of data points (fig. 6). Using Gaussian Kernel Density Estimation on the distribution of data from the latent

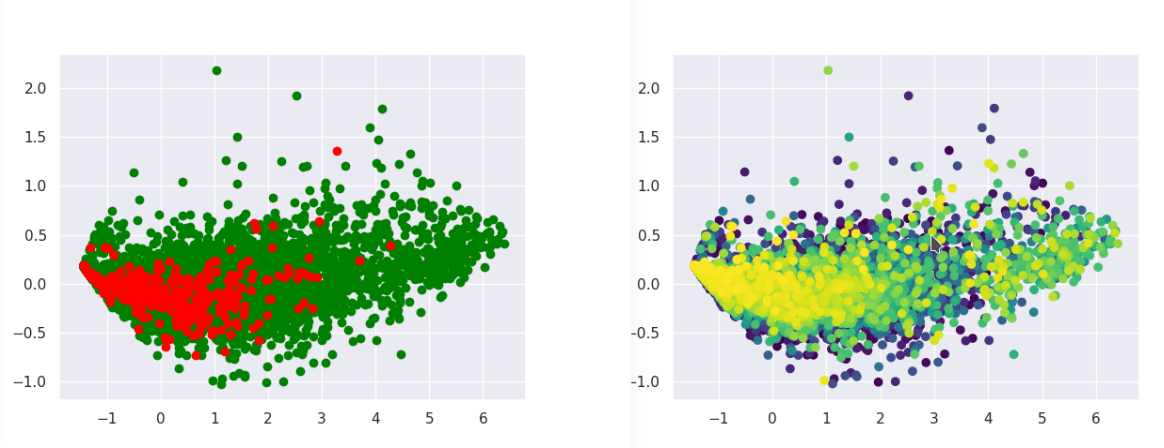

Figure 6: Distribution of the latent space's output (left), where red points are positive samples and green are negative. The densest part of the scatter plot is where the positive samples gather, therefore, the less-dense regions can be discarded.

space's output, we set a threshold for the density of each data point and remove the samples that fall under the threshold. Then we train the autoencoder anew with the filtered dataset.

3 Results

3.1 Results from the autoencoder

It is difficult to test the autoencoder separately, since it cannot do classification. We can check how accurate it is at recreating the input, but that doesn't necessarily give us information about how successful can the latent space's data be clustered. The layer sizes and parameters of the first network mentioned (with max pooling and upsampling) are very similar to the ones showed in Table 2. Comparison between its input and the output it produces can be seen on Fig. 7.

The network presented in Table 2 performs differently from the first one. The main difference between the two networks are the dimensionality reduction layers - in the second network they are convolution with strides, without padding, and transposed convolution. The recreated input of the second network can be observed in Fig. 8.

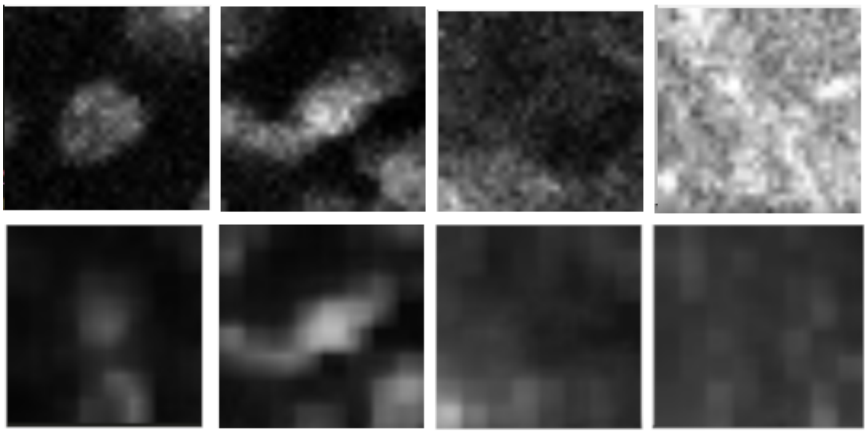

Figure 7: Comparison of the input (first row) and the corresponding output (second row) from the network with pooling and upsampling layers.

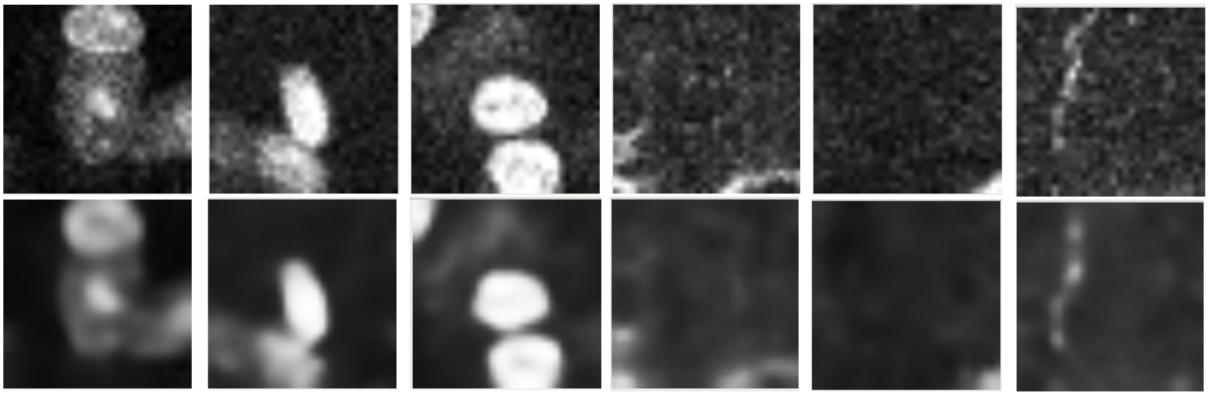

Figure 8: Comparison of the input (first row) and the output (second row) from the network with transposed convolution and convolution with strides and no padding. The output is very similar to the input. However, this doesn't mean that the compressed data in the latent space can be clustered correctly.

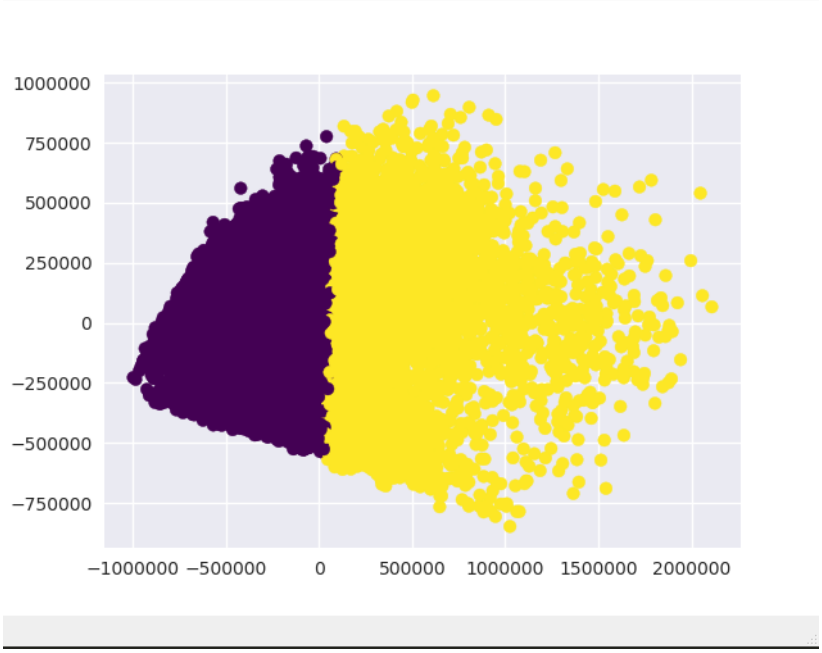

Figure 9: Visualization of the two clusters. Purple data points belong to the cluster, collecting the samples, that are classified as positive and yellow - to the one collecting the samples classified as negative. In the ideal case, the two clusters should be further from each other and contain only the ground truth positive and negative samples.

3.2 Results from the clustering

Since the data points are high-dimensional vectors (64-dimensional in the network from 2), we have to reduce the dimensions in order to visualize them. For this purpose we have used Principal Component Analysis (PCA) to reduce the dimensions to only 2. Fig. 9 shows how are the two clusters fit to the data. Ideally, one cluster should contain all the positive samples (nuclei with its cell of a certain type) and the other cluster - all the negative samples. Fig. 10 visualizes how the ground truth positive samples are positioned in relation to the negative samples when using a bigger latent space (Dimensions: 1x256). Fig. 11 shows the same relation when using a smaller latent space (Dimensions: 2x2). It can be observed that, although the positive data points do not form a separate cluster, they do appear close to each other in space. This means that the autoencoder has learned specific features for the cells.

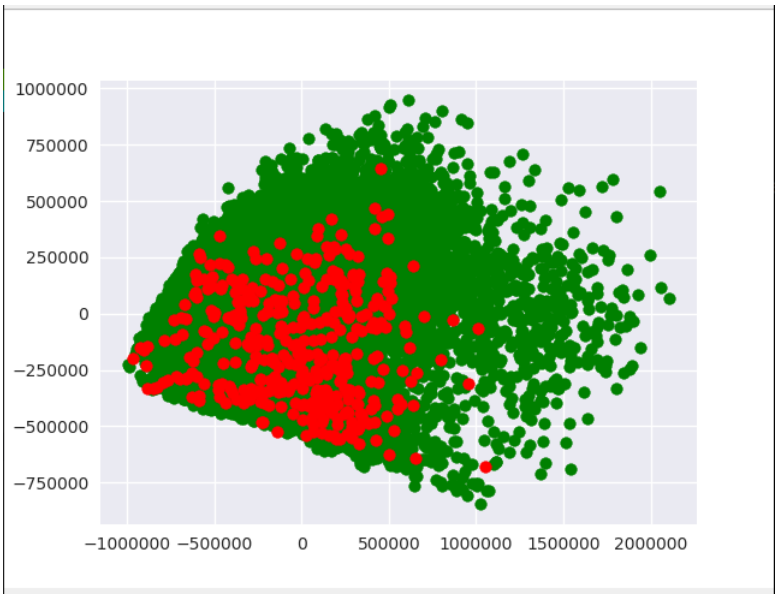

Figure 10: Visualization of the distribution of the truly positive and negative samples with latent space dimensions $1x256$. Red data points are the positive samples. Green data points are the negative. In the ideal case, the red and the green data points should be separated and form two clusters.

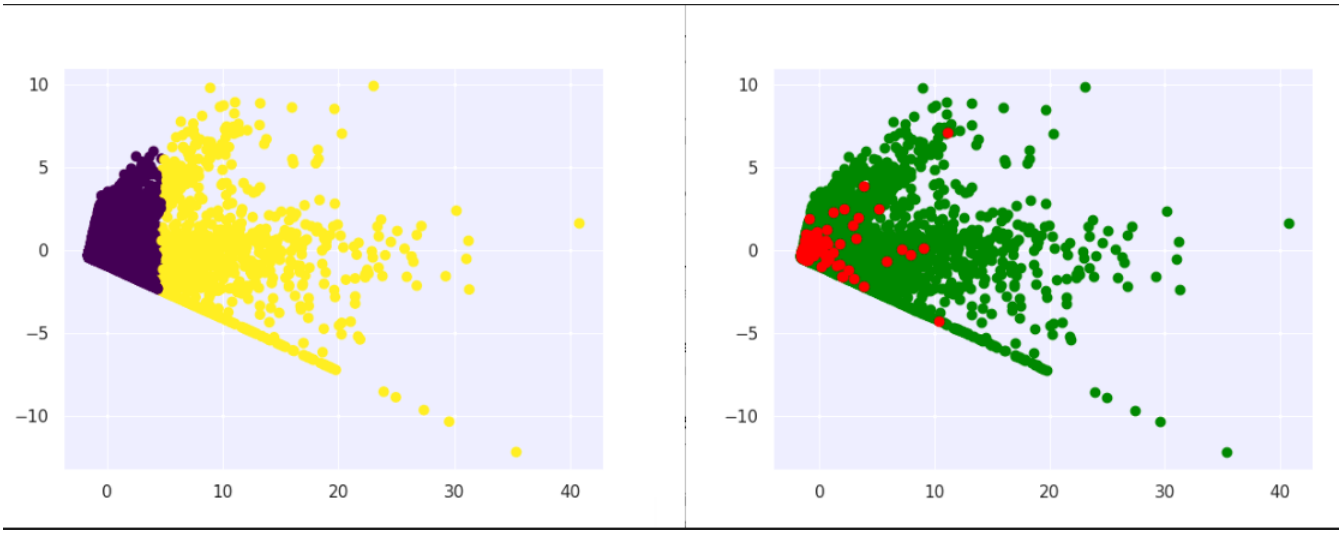

Figure 11: Visualization of the distribution of the truly positive and negative samples with latent space dimensions $2x2$. Red data points are the positive samples. Green data points are the negative. In the ideal case, the red and the green data points should be separated and form two clusters.

4 Discussion

4.1 Results from the autoencoders

At first sight, both networks manage to recreate the input successfully. The obvious difference is that the representation of the network, showed in Table. 2, is more blurred. The outputs from the first architecture are more pixelated. Therefore, we suspect it creates less detailed representation of the input. However, it still contain the cells and the nuclei on them. The pixelation might be a result of a not so well trained decoder, which does not directly affect out results, since we only use the latent space's output. We require further testing to establish which archictecture provides better classification. Overall, it seems like the autoencoder performs good enough on its part.

4.2 Results from the clustering

In the ideal case, the data points should be gathered in distinguishable clusters, since we expect the positive samples to be encoded in a different way from the negative. The reasons for our false clustering may be due to either inaccurate work from the encoder or unefficient clustering method. However, the results from experimenting with smaller latent space (2x2) 11 show that the encodings of the positive samples do show similar features. The problematic clusterization comes from the fact that they aren't distinct enough from the negative samples. An image patch that does not contain a cell may contain different kind of noise and it may be difficult for the autoencoder to extract similar features from such patches. For this reason, an outlier removal algorithm may be useful to erase some of the negative samples.

5 Conclusion

In this research we tried to create an unsupervised machine learning approach for brain cell phenotyping. We developed an autoencoder using different architectures that try to learn the most important information about the cell pictures we feed into it. We expect that this compressed data can be used to distinguish if a nucleus is of a certain cell type or not. For this classification we used the K-means clustering algorithm. The results so far do not support that this task can be fulfilled with these approaches, since the data does not form distinct clusters. However, the improved results with the reduced dimensionality of the latent space prove our hypothesis that the neural network is indeed learning important features about the cells. Given this, and the methods we are about to implement for improving the synergy between the autoencoder and the clustering, and also the data processing, we are optimistic that classification will soon be possible.

6 Further Development

Even though the current results aren't conclusive, we have grounds to believe that new methods may improve performance. These include:

- Trying more types of autoencoders. Other types may compress the data in a different way and provide us the functionality we need.
- Trying to reduce the dimensionality of the latent space. Reducing the dimensionality of the vectors that are to be clustered may enforce the learning of specific features of the input and will also make the clustering process. easier
- Trying other types of clustering and also implementing our own.
- Implementing simultaneous training of the clustering and the autoencoders.

References

[1] Kwanghun Chung and Karl Deisseroth. Clarity for mapping the nervous system. *Nature methods*, 10(6):508, 2013.

[2] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *arXiv preprint arXiv:1610.04794*, 2016.

[3] Jin-Wu Tsai, K Helen Bremner, and Richard B Vallee. Dual subcellular roles for lis1 and dynein in radial neuronal migration in live brain tissue. *Nature neuroscience*, 10(8):970, 2007.

[4] Jun B Ding, Kevin T Takasaki, and Bernardo L Sabatini. Supraresolution imaging in brain slices using stimulated-emission depletion two-photon laser scanning microscopy. *Neuron*, 63(4):429–437, 2009.

[5] Francis HY Chan, Francis K Lam, and Hui Zhu. Adaptive thresholding by variational method. *IEEE Transactions on Image Processing*, 7(3):468–473, 1998.

[6] Omar Gameel Salem. Connected component labeling algorithm. 2014.

[7] T. Lindeberg. Scale Invariant Feature Transform. *Scholarpedia*, 7(5):10491, 2012. revision #153939.

[8] Nick Efford. Morphological image processing. 2000.

[9] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *arXiv preprint arXiv:1610.04794*, 2016.

[10] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. *arXiv preprint arXiv:1708.02551*, 2017.

[11] Michael Steinbach, Levent Ertöz, and Vipin Kumar. The challenges of clustering high dimensional data. In *New directions in statistical physics*, pages 273–309. Springer, 2004.

[12] Julien Despois. Autoencoders - deep learning bits 1.

[13] Cs231n convolutional neural networks for visual recognition.

[14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.