DEGREE PROJECT IN THE FIELD OF TECHNOLOGY
MEDIA TECHNOLOGY
AND THE MAIN FIELD OF STUDY
COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Uncovering Correlations Between Two UMAP Hyperparameters and the Input Dataset

FEDERICO JIMÉNEZ VILLALONGA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRAKT

Att lära sig små högdimensionella bilddatamängder kan vara en utmaning: modeller för djupinlärning är svåra att använda på grund av begränsade data och enklare modeller för maskininlärning kan vara långsamma på grund av det stora antalet egenskaper. UMAP är en metod för dimensionsminskning som skapar lågdimensionella representationer av dataset, som kan användas som input till enkla modeller, vilket minskar beräkningstiden. Att hitta de bästa hyperparametervärdena kan dock kräva många tester. Denna uppsats syftar till att minska antalet gissningar genom att avslöja möjliga korrelationer mellan datasetparametrarna och två hyperparametrar i UMAP. Nio dataset har bäddats in med olika kombinationer av hyperparametrar och KNN-precisionen har beräknats för att rangordna inbäddningskvaliteten. En korrelation mellan två hyperparametrar hittades. Datasets med färre egenskaper och/eller lägre siluetter följde närmast ett sådant mönster, med föredragna vårdområden för hyperparametrarna. Dessutom verkade inte träningsmängd spela någon roll för valet av hyperparametervärden.

Uncovering Correlations Between Two UMAP Hyperparameters and the Input Dataset

Federico Jiménez Villalonga
Royal Institute of Technology
Stockholm, Sweden
fjv@kth.se

ABSTRACT

Learning small high-dimensional image datasets can be challenging: while deep learning models struggle, because of the limited data, simpler machine learning models can be slow, due to the high number of features. UMAP is a dimensionality reduction method that creates low dimensional representations of the datasets, which can be used as input to simple models, reducing the computational time. However, finding the best hyperparameter values might require many tests. This thesis aims to reduce this guesswork by uncovering possible correlations between the dataset parameters and two UMAP hyperparameters. 9 datasets have been embedded with different hyperparameter combinations and the KNN accuracy was computed to rank the embedding quality. A correlation between two hyperparameters was found. Datasets with lower number of features and/or lower silhouette followed such pattern most closely, with preferred hyperparameter value ranges. Moreover, training size did not seem to play a role in the choice of the hyperparameter values.

INTRODUCTION

Image classification is a common computer-vision task in which a machine learning model is tasked to guess the label(s) of a given image, usually a photograph, from a group of previously learnt labels. These labels, or classes, refer to the most relevant features of the image. The relatively simple task lays the foundation for more complex ones, like object detection identification, which have many applications across various fields such as anomaly detection, video-tracking, surveillance, and image search, just to name a few.

Two variables that determine the successful implementation of a model are the quality and quantity of the training data. Therefore, much effort is being put into collecting many high-quality images, leading to the creation of massive datasets with easily thousands of features. Many algorithms, however, perform poorly when dealing with such datasets. While, the large training size increases the computational time, the high number of features causes the "Curse of Dimensionality" [30]: here, pairwise distances between the points tend to converge to the same average, causing distance and cluster based algorithms to underperform.

Therefore, new techniques have been developed for working with large high-dimensional datasets. Convolutional Neural Networks (CNNs), for example, are a type of deep learning network where the input is passed through a series of layers which modify it to extract common visual patterns, bypassing the pixel-by-pixel feature extraction. This process is faster than most other machine learning algorithms and leads to better results. CNNs have gained popularity and are now used by many companies worldwide. A drawback of using this technique is that a lot of data is needed in order to obtain acceptable results [2,6,20].

In some areas of research, unfortunately, data is scarce and so valuable that not many are keen to share it with the public. Here, data augmentation on real [24] or synthetic data [22] but also pre-training on bigger datasets, [18,27,29] can be applied. While these methods can boost the performance, they also come with some drawbacks: the augmented data is only as good as the source data [15], and, if too closely related to the original dataset it can introduce bias or cause the model to learn irrelevant features [24]. Transfer learning on the other hand, can be hard to implement: additional datasets might be hard to obtain, but most importantly, highly specialized models such as the ones used in medical imaging, are very sensitive to the pretraining bias [3]. Therefore, it is still unclear how to approach these cases. On one hand, the high dimensionality calls for the use of deep learning, while on the other, the small size of the training sets suggests simpler models could obtain better results.

Real-world image datasets, however, are not as high-dimensional as they may first appear. As hypothesized by Goodfellow et al. [9] and proved by Carlsson et al. [4], natural images tend to group in dense sparse clusters. This is the so-called Manifold Hypothesis, according to which, *"real-world high-dimensional image data lie on low-dimensional manifolds embedded within the high-dimensional space"*[1]. Indeed, real-world image datasets can only cover real things, which make up only a small part of all the possible pixel combinations. Therefore, there is a lower dimensional manifold on which the data is still well represented. The number of dimensions of this manifold is called the intrinsic dimension.

[1] https://deepai.org/machine-learning-glossary-and-terms/manifold-hypothesis

Dimensionality reduction methods are a group of algorithms capable of embedding high-dimensional data into a lower-dimensional space. Supported by the Manifold Hypothesis, a sub-group of this group has been created, called Manifold Learning. These methods can retain most of the complex correlations between the points of the high-dimensional dataset, therefore they are often adopted to solve the problem of learning small high-dimensional datasets, especially when dealing with distance or cluster-based models. Manifold Learning methods however are not easy to implement since they require many trial-runs to find the right hyperparameter values.

This thesis aims to streamline the process of choosing these values for the most recent Manifold Learning method, UMAP. More specifically, it will try to answer the question: *"Is there an empiric correlation between the dataset parameters and the UMAP's hyperparameter values?"*

BACKGROUND

The paragraphs will describe two commonly used dimensionality reduction methods, of which the second one is part of the Manifold Group.

Principal Component Analysis (PCA) [12] is a matrix factorization linear technique that works by compressing the variables of a dataset by means of correlation: variables that behave similarly are squished into a new dimension called a component. The objective of PCA is to find the minimum number of components that still describe the dataset well enough so as not to lose too much information. In the field of Data Science it is often utilized in the data pre-processing step: the lower-dimensionality dataset that PCA creates is used to train, for example, supervised algorithms like Support Vector Machines (SVM) [8,33].

Annie's work [8] is an early example of what PCA can do when paired with a simple model: here, the author compares two solutions to an anomaly classification problem. Both cases make use of an SVM model, but in the second one, PCA was used to reduce the dataset from 42 to 28 dimensions, which increased the precision and recall while also decreasing the execution time. The author brings forward that the embedding of the dataset improved the performance because the principal components of PCA, orthogonal between each other, cluster the features based on their variance, therefore helping the SVM model to rely more on the ones with less variance. This leads to more accurate results while also taking less time because SVM's search for the hyperplane is made easier by the reduced feature set.

However, PCA is based on the assumption that features are linearly correlated and can therefore only re-express the data as a linear combination of its features [28]. But when working with real-world datasets, nonlinear correlations are very common. This is the case of the famous Swiss Roll dataset. Here, the objective is to reduce a 3D roll down to two dimensions. Unfolding the roll, should result in a uniform gradient from the red data points to the blue ones.

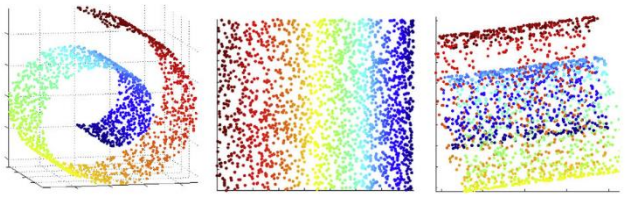

Figure 1. 3D Swiss Roll on the left. Ideal 2D unfolding in the centre and PCA result on the right.

As shown by [19], PCA is not able to correctly represent the non-linearity of this manifold because the distances between close points, called local distances, are warped by the roll and PCA struggles to correctly represent them in two dimensions. At the same time, the sum of distances between groups of far points, called global distances, are correctly represented. But reducing high dimensional non-linear manifolds it is also important to correctly represent the local distances, so to better separate the clusters.

Many new techniques have therefore been developed with a special attention in preserving nonlinear local variance. t-Distributed Stochastic Neighbour Embedding (T-SNE) [21], for example, is another popular algorithm used for data visualization. Here, the high-dimensional dataset is embedded in two dimensions by minimizing the difference between the two distances of two points in their high and low-dimensional graph representations. Repeating this process multiple times for every point, can generate a very clean visualization of the dataset. T-SNE however, has two main drawbacks. The first one being that it is a non-convex method, in which bad hyperparameter initializations can lead to poor local minima. This implies that many hyperparameter values may be tried before obtaining an acceptable result. The second drawback lies in the reason why T-SNE was developed. Its main goal is to display high-dimensional data in a clear and readable way. In particular, it tries to solve the "crowding problem", in which high-dimensional data points tend to become equidistant to each other when reducing them to a lower dimension, which in turn can cause them to overlap, forming very distinct but dense clusters. T-SNE solves the issue by artificially expanding the dense clusters and contracting the sparse ones, so to even out the cluster size. Although this makes for better visualizations, it does not truthfully represent the original data and should not be used as an input to a model.

Uniform Manifold Approximation and Projection

Taking inspiration from T-SNE, McInnes et al. developed the Uniform Manifold Approximation and Projection (UMAP) [23]. Just like T-SNE, it is a graph layout-based method, often used to visualize high dimensional data in a two-dimensional graph. However, UMAP can offer much more: since it is built on strong theoretical foundations, many researchers use it in the field of genomics for reducing the dimensions of genetic data. Despite being a novel algorithm, it has gained popularity in many application domains due to its ability to retain both global and local distances. UMAP

works in two steps. Firstly, it constructs two weighted k-neighbour graphs. One from the original high-dimensional dataset, and the other, from the same dataset embedded in the smaller dimension specified by the user with the *n_components* hyperparameter. Secondly, it minimizes the edgewise cross-entropy between the distances of the datapoints of these two graphs. To create the first weighted graph, it makes three assumptions on the data:

- The data is uniformly distributed on a Riemannian manifold.
- The manifold is locally connected.
- The Riemannian metric is locally constant.

Let $X = \{x_1,\ldots,x_N\} \in \mathbb{R}^M$ be the input dataset and $Y = \{y_1,\ldots,y_N\} \in \mathbb{R}^K$ the embedded dataset such that $K < M$. Two parameters, ρ_i and σ_i, are computed for each point x_i. ρ_i is the distance between the point x_i and its nearest neighbour.

$$\rho_i = \min\left\{d\left(x_i, x_{i_j}\right) \,\middle|\, 1 \le j \le k, d\left(x_i, x_{i_j}\right) > 0\right\}$$

Where d is the hyperparameter *min_dist*, the metric used to calculate distances between points and k is the hyperparameter *n_neighbors* which represents the neighbourhood size. This ensures that the manifold is locally connected, since every point is connected to at least another one. Then the normalization factor σ_i is calculated from:

$$\sum_{j=1}^{k} \exp\left(\frac{-\max\left(0,\ d\left(x_i, x_{i_j}\right) - \rho_i\right)}{\sigma_i}\right) = \log_2(k)$$

σ_i ensures that the local manifold of x_i is uniformly distributed and that the metric is locally constant. Essentially, ρ_i and σ_i are calculated so to find a custom metric for each x_i that respects the three assumptions. For every x_i an induced weighted directed graph of its k-nearest neighbours is obtained with the formula $\vec{G}_i = (V_i, E_i, \omega)$, where V_i is the set of x_i plus its neighbours, E_i is the set of the directed edges pointing towards x_i and ω their corresponding weights, found by the formula:

$$\omega\left(x_i, x_{i_j}\right) = \exp\left(\frac{-\max\left(0, d\left(x_i, x_{i_j}\right) - \rho_i\right)}{\sigma_i}\right)$$

In short, ρ and σ are used to find the relationships between any given point x_i to its k-nearest neighbours from which a graph is then extracted. To obtain a representation of the whole dataset, these graphs must be combined into a uniform undirected graph. Given any two points x_i and x_j, there exists a maximum of two edges connecting them, one from x_i's induced graph and one from x_j's induced graph. These are combined with: $e_{ij} = \overrightarrow{e_{ij}} + \overleftarrow{e_{ij}} - \overrightarrow{e_{ij}} * \overleftarrow{e_{ij}}$, to create the undirected graph G = (V, E, ω) of the complete dataset. A similar procedure is then repeated to obtain the graph representation H of the embedded dataset. The second step consists in optimizing this representation so to minimize the structural difference between the two graphs. To do so, two artificial forces are applied to each pair of points. One attractive force that brings them closer together and a repulsive one that pushes them further away from each other. The total difference between the two graphs can be seen as the sum of these forces at every edge.

$$\sum_{e\,\epsilon\,E} \omega_h(e) \log\left(\frac{\omega_h(e)}{\omega_l(e)}\right) + \left(1 - \omega_h(e)\right) \log\left(\frac{1 - \omega_h(e)}{1 - \omega_l(e)}\right)$$

Where $\omega_h(e)$, is the weight of the edge in the high-dimensional graph representation while $\omega_l(e)$ is the weight of the edge in the low-dimensional graph representation. The first term is the attractive force: when bringing the points closer together, $\omega_l(e)$ increases and the term tends to 0 therefore decreasing the difference between the graphs. The second term is the repulsive force because separating the points will lower the $\omega_l(e)$ value, also reducing the difference. UMAP iteratively tweaks the positions of the lower-dimensional points to minimize the sum of differences until a balance is found and a local minimum is reached.

Hozumi et al. [13] introduced a clustering strategy based on UMAP and K-means to study the SARS-CoV-2 genome sequences. In their work, they firstly compared the performance of PCA, T-SNE and UMAP on 4 datasets and with different reduction ratios (from 2 dimensions to half of the total dimensions) both assisting a supervised method, KNN, and an unsupervised one, K-means. Their results show that UMAP does not outperform PCA and T-SNE every time. However, due to its consistency and lower computational time, they decided to use it for working with the SARS-CoV-2 dataset. PCA, performed ng best when paired with KNN but it failed to reduce the datasets with large sample size, when paired with K-means. T-SNE often led to inconsistent results and overall performed worse than UMAP with longer computational times. As explained before, this could be caused by T-SNE's focus on preserving local distances at the cost of meaningless global distances.

Another successful implementation of UMAP comes from Allaoui et. al paper [1]. Here, the authors embedded the input data to four clustering algorithms to see whether it could improve the accuracy on five high-dimensional image datasets. It was found that UMAP increased the accuracy in every test they ran, up to 60% from its initial value. They attributed this improvement to the fact that clustering algorithms suffer from the curse of dimensionality.

It is then clear that UMAP pre-processing can increase the performance of machine learning models. Just like its predecessor T-SNE however, it still suffers from being a non-convex optimization problem: bad hyperparameters initialization can lead to poor local minima and therefore to a loss in performance. This thesis aims to investigate the existence of possible correlations between the dataset characteristics and the UMAP hyperparameters. These correlations could be used to the user's advantage and reduce the guesswork made on hyperparameters values while consistently leading to the best possible results.

UMAP Hyperparameters

A UMAP object can be implemented through the python *umap-learn* module. It can take around 20 hyperparameters but, as explained in the UMAP's official documentation[2], it is possible to achieve excellent embeddings by adjusting the three basic ones and leaving the rest to their default value.

- *n_neighbors* is the neighbourhood size of the two k-nearest neighbour graphs. Its recommended range is 2 to 50. It balances the trade-off between local and global structure definition. Small values capture well the local manifolds but may fail when summarizing their interaction, therefore losing definition of the overall structure. With larger values, on the other hand, the global information is retained but the local variance will be smoothed out.
- *min_dist* sets the minimum distance at which the points must be in the embedded graph. Low values bring the points closer together, highlighting the clusters. Higher values improve the visualization by spreading the points apart. Even though it has been designed by McInnes as an *"aesthetic parameter"* for visualizations, it has influence on the output and Allaoui et al. [1] have indeed reported that a *min_dist* of 0 can improve the performance on clustering problems. This is because denser cluster are more easily classified.
- *n_components* represents the number of embedded dimensions. When using UMAP for visualizations, it is normally set to 1, 2 or 3. For data embedding however, a higher number of dimensions might yield better results. Hozumi et al. [13] indeed tried different ratios (from 2 dimensions to half of the total dimensions), but from their tests, it appears that a low value of *n_components* such as 2 or 3 works just as well if not better than higher ones. Moreover, higher values increase the computational time both of UMAP algorithm and of the model.

METHOD

When uncovering these correlations, it must be considered that they should be valid not only for a few datasets, rather, they should be held as guidelines, applicable to any dataset. This in turn, implies that many datasets must be tested, so to only extract the universal patterns that are common to all of them. Due to the limited amount of time and resources, collecting and testing many datasets is out of the scope of this work, which instead, defines a working proof of concept, to which more datasets could be added at a later stage.

Algorithm 1 illustrates how the testing is carried out: each dataset is firstly retrieved, either from an online source or from a local save. Then, for each test, it is embedded through a UMAP instance. The UMAP output is then scaled and fed to a machine learning algorithm which returns the predictions on the test set. This process is repeated with varying UMAP hyperparameters. The results are then saved and later analysed to discover the possible correlations.

Algorithm 1: Running the tests

```
for dataset in datasets:
    retrieve and pre-process dataset
    compute dataset parameters
    for test in tests:
        embed the dataset with UMAP
        scale the embedded dataset
        fit the model on the embedded training set
        predict the labels of the test set
        compute the accuracy of the model
    save the accuracy score along with other test values
```

Dataset Parameters

The dataset parameters are a set of characteristics that can summarise it while distinguishing it from others. Ideally, one set should identify only one dataset. There are many other variables that could describe a dataset. The following ones have been chosen because of three reasons. Firstly, they are relatively easy to obtain. Secondly, some are thought to impact the model's performance [26] and therefore, they could lead to the optimal UMAP hyperparameters values. Thirdly, this specific set appropriately describes the dataset from a quantitative and qualitative point of view.

- The value of the intrinsic dimensionality, since it has been found by Pope et al. [26] that it has an effect on the model's generalization from training to test data: datasets with lower values are better generalized that their counterparts Moreover, it is used as one of the tested values for the number of dimensions of the embedded dataset, since, in theory, it should provide an almost perfect representation of the dataset. There are many types of intrinsic dimension estimators [32] and exploring them all is out of the scope of this work. Therefore, a simple but fast global estimator has been chosen. It is based on the PCA algorithm and has been implemented with the *scikit-dimension* module[3].
- The number of instances quantitatively describes the dataset. As seen in [13], UMAP scales well with this parameter, but struggles when it is too low. It would be interesting to find a lower-bound under which UMAP is not worth using.

[2] https://umap-learn.readthedocs.io/en/latest/parameters.html

[3] https://scikit-dimension.readthedocs.io/en/latest/index.html

Datasets	Intrinsic Dimensionality	Number of Instances	Number of Classes	Number of Features	Silhouette
MNIST	38	60,000	10	784	0.04
Fashion-MNIST	8	60,000	10	784	0.04
CIFAR-10	9	50,000	10	3072	-0.05
CIFAR-100	9	50,000	100	3072	-0.11
COIL-20	13	1008	20	16384	0.16
Intel Image Classification	9	11885	6	67500	-0.01
UMIST	14	402	20	10304	0.08
USPS	22	7291	10	256	0.11
ShipsNet	7	2800	2	19200	0.01

Table 1. List of the datasets and the value of their parameters.

- The number of features or dimensions provides a measure of quality of the instances. This parameter might also be correlated with the number of embedded dimensions: datasets with thousands of dimensions might store more information than those with just hundreds, therefore needing a higher *n_components* to be properly represented.
- The silhouette describes the quality of the clustering. It varies between -1 and +1, -1 being the worst case, in which the existing clustering is wrong, while +1 is the best case in which each point is assigned to the right cluster. The *silhouette_score* from *sklearn* has been used to calculate its value.

The number of classes might also be correlated: a high number of labels might suggest the presence of an also high number of clusters, which in turn, might influence the *n_neighbors* hyperparameter. However, as shown in Table 1, almost all the tested datasets have a similar value. Therefore, the analysis on this parameter is omitted.

Datasets

There are many high-dimensional image dataset that can be freely accessed and used. The following one have been selected because they range in very different parameter values. This way, different cases are explored, and the final guidelines are less prone to being biased towards a subset of values. Moreover, most of these datasets are very often used in this domain field for testing purposes, are easily accessible and downloadable and they do not require complicated pre-processing or data augmentation steps. Each dataset has its own getter function, which returns the training and testing data, along with their labels.

- MNIST [17] is made of 70,000 28 by 28 pixel grayscale pictures of the handwritten digits, of which 10,000 are used for the test set.
- Fashion-MNIST [31] shares the same properties of MNIST but contains pictures of 10 different types of clothing instead. It was made to replace MNIST, which is considered too easy for modern models.
- CIFAR-10 [16] consists of 60,000 32 by 32 pixels colour pictures of 10 classes representing 7 animals and 3 objects. the test set takes 10,000 instances.
- CIFAR-100 [16] is similar to CIFAR-10 but it has 100 different classes.

These four abovementioned datasets are obtained through the *keras* module, which firstly loads them in memory and then divides them into the training and testing set. Afterwards, every image is reshaped to be a one-dimensional array and scaled to values between zero and one. The remaining datasets are downloaded from their respective online source and saved to a local folder, from which they can be loaded into memory with the *Pillow* module. As showed in Algorithm 2, the rest of the steps is similar to what was done with the *keras* datasets.

- COIL-20 [25] contains 1,440 128 by 128 pixels grayscale pictures of 20 common objects photographed 72 times each, while rotating on a turntable (1 picture every 5 degrees).
- Intel Image Classification [34] is made of 24,335 150 by 150 colour pictures of 6 classes (buildings, forests, glaciers, mountains, sea, streets). 7,301 of these are not labelled and have been discarded.

- UMIST [10] consists of 575 220 by 220 grayscale pictures of close-ups from 20 different people, at various angles between frontal and lateral views.
- USPS [14] also represents the 10 handwritten digits, coming from the U.S. Postal Service. Every image is made of 16 by 16 grayscale pixels.
- ShipsNet [11] consists of 4,000 80 by 80 colour satellite images of which 1,000 include a ship and the remaining 3,000 do not.

Algorithm 2: Local datasets pre-processing

```
for image in folder:
    open image with Pillow
    transform image to array
    reshape array to one dimension
scale the dataset
retrieve labels
split dataset in 70% for training 30% for testing
```

K-Nearest Neighbours

K-Nearest Neighbours [7] has been chosen to make the predictions on the embedded dataset due to it being a simple non-parametric algorithm. Indeed, many others require their parameters to be tuned for each dataset and repeating this step for all of them would greatly increase the computing time and more importantly, introduce unwanted variables in the tests. KNN, however, does not make any assumption on the underlying data and, has previously determined in [5], its performance is consistent across different embeddings. The hyperparameter K has been left to its default value of 5. This resilience towards changes to the input dataset ensures that any difference in the performance is solely caused by the choice of UMAP hyperparameter values. As discussed in [13], KNN does not always benefit from embedding the input data, rather, their results show that the baseline score calculated on the high-dimensional dataset, sometimes outperform the scores on the embedded datasets. It is important to note, however, that the objective of this work is not to improve the performance of an existing algorithm, but rather to look at the relative variations in performance, caused using different hyperparameter values. For this reason, the baseline score is not calculated.

There are two main disadvantages when using KNN, but they are not an issue in this use case. Firstly, it is space expensive, because, to make the predictions on the testing data, it must load all the training data into memory. To avoid this, the tests were run on a machine with 16 Gigabytes of RAM and the datasets were loaded one at a time. Python garbage collector, then, ensured the removal from memory of the already tested datasets. Secondly, because the algorithm works by comparing the Euclidean distances between points, all the input features must be scaled to the same range. Since UMAP's output is not scaled, the *MinMaxScaler* method from *sklearn* has been applied to the embedded dataset.

Tests

As illustrated in Algorithm 1, after a dataset has been loaded in memory, a series of tests are run on it. Each test output a row that is appended to a *pandas* DataFrame object. When the tests for a certain dataset are finished, the DataFrame is saved to a csv file. Each test is a different combination of values of the *n_neighbors* and *n_components* hyperparameters. For *n_neighbors* the values 2, 5, 10, 20, 50 are tested and for *n_components*, the values between 1 and two times the value of the intrinsic dimension are tested. The value of *min_dist* has also been set to 0 for all tests. At the beginning of each test, a timer is started, to keep count of its execution time. Afterwards, a new UMAP instance is initialized, called reducer, with the chosen hyperparameter values. The reducer then fits the training data and embeds both the training and testing data. The lower-dimensional output is then transformed with the *MinMaxScaler* method, which rescales each feature individually in the zero-one range. Assuming x to be the feature value of an instance, its scaled counterpart is calculated with the following formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Where x_{min} and x_{max} are the minimum and maximum values of that feature. Just like the reducer, the scaler first fits the training data then scales the training and test data.

Now, the embedded scaled training set can be used, along with its labels, as an input to the KNN instance, which is also implemented with the *sklearn* module. After the instance has fitted the training set, it predicts the labels of the test set. The predictions are outputted as one-dimensional array and saved to a local variable. The execution timer is now stopped. Finally, the accuracy score of the KNN instance is calculated, by comparing the prediction array and the true label array of the test set. The output of each test that is appended to the DataFrame contains 9 values. Namely, the 4 analysed dataset parameters, the 2 UMAP hyperparameters, the dataset name, the execution time, and the accuracy score. When the tests for a certain dataset are finished, each *n_components* value has been tested with the 5 different *n_neighbors* values. Since *n_components* varies between 1 and 2 times the value of the intrinsic dimension, the final shape of the saved DataFrame is of 9 columns and 10*intrinsic dimension rows.

A total of 1,290 tests have been executed, which took slightly less than 165 hours. To ensure that the execution times were accurate, the computer has not been used whilst the tests were being performed. Because of the lengthy computational time, the datasets have been tested one at a time. The tests for the MNIST dataset have furtherly been divided into 6 smaller tests (i.e., a first test batch from 1 to 12 components, then from 13 to 19 etc.), because of its high intrinsic dimensionality, which led to the total execution time for this dataset to be of 66 hours.

RESULTS

The *pandas*, *matplotlib* and *seaborn* modules have been utilized to handle the DataFrames and create the graphs. The following 4 chapters analyse the results from the point of each dataset parameter.

Intrinsic Dimension

The behaviour of the accuracy score with regards to the number of embedded dimensions varies when comparing different datasets. What is common amongst all cases, is that setting *n_components* to the intrinsic dimensionality does not necessarily improve the performance of the model. Let us consider the results on the COIL-20 dataset.

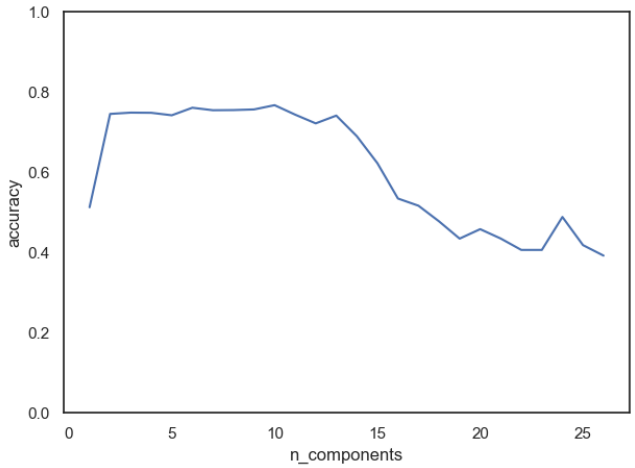

Figure 2. Mean accuracy over n_components in COIL-20

This graph shows on the x-axis the tested *n_components* and on the y-axis, the mean of the 5 accuracies found for each n_*neighbors*. The intrinsic dimension of COIL-20 is 13, at which the accuracy score is 74%, but the highest accuracy is reached when *n_components* is 10, with 76%. More importantly, the mean accuracy drops almost vertically, right after the intrinsic dimension is reached. This behaviour is also common to other datasets. In USPS, for example, the drop happens even before the intrinsic dimensionality value.

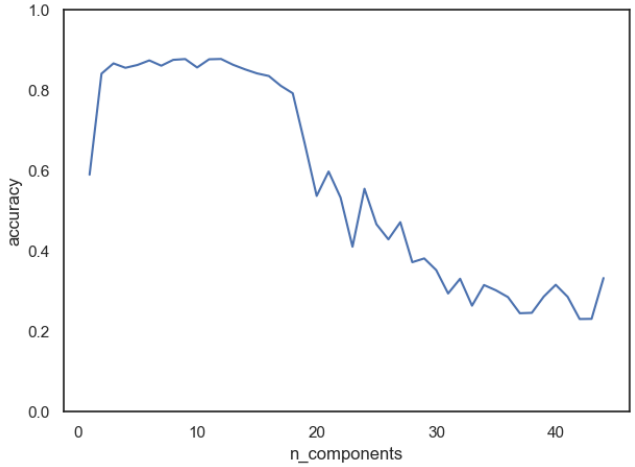

Figure 3. Mean accuracy over n_components in USPS

Some datasets, like CIFAR-10, do not present a drop in performance around the intrinsic dimension, while others, like UMIST, have a shallower decline.

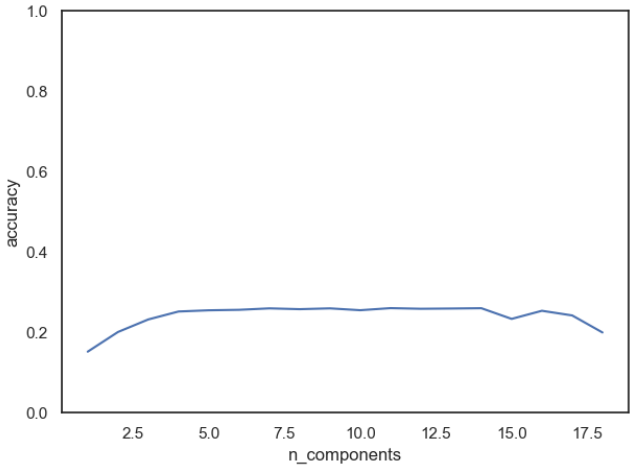

Figure 4. Mean accuracy over n_components in CIFAR-10

Here, the mean accuracy remains consistent throughout the tests. Since the CIFAR-10 accuracies are very low, it could be argued that the model never truly learned this dataset.

When considering the maximum accuracy, rather than the mean, no decrease in accuracy is seen.

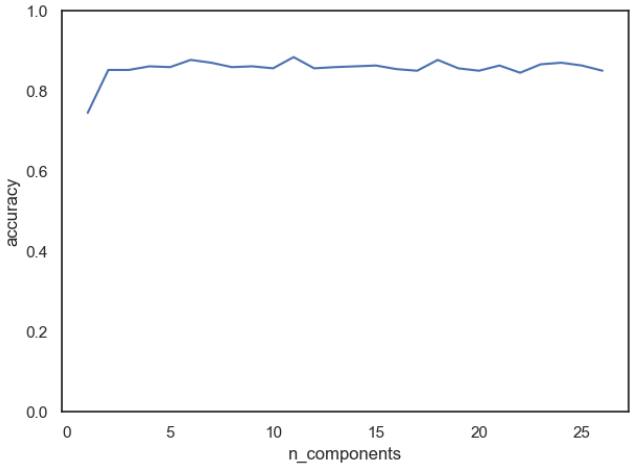

Figure 5. Max accuracy over n_components in COIL-20

For COIL-20 and almost every other dataset, there is either no loss in accuracy score, or a very shallow one, which also happens around the intrinsic dimensionality.

When considering all the data points at once, instead of the mean, it can be seen that *n_neighbors* is also tied to the accuracy score. Figure 6 shows another common behaviour, in which, for small *n_components*, *n_neighbors* does not seem to have a significant impact on the accuracy. When increasing *n_components* however, higher values of *n_neighbors*, like 20 and 50, significantly decrease the performance. This is what ultimately causes the steep drops in the mean accuracy of some datasets.

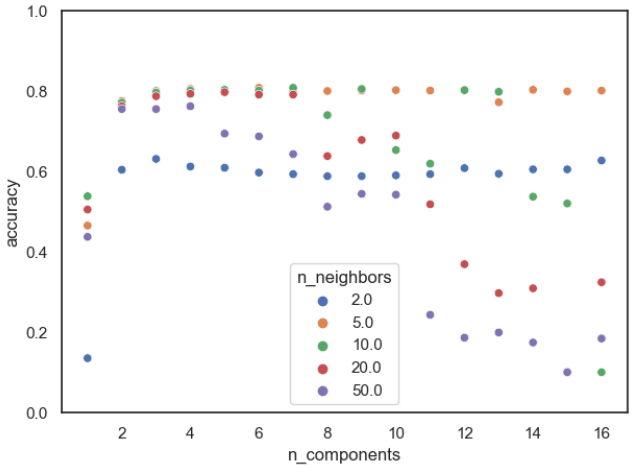

Figure 6. Accuracies over n_components in Fashion MNIST

A better example of this behaviour can be seen in the results from the USPS dataset.

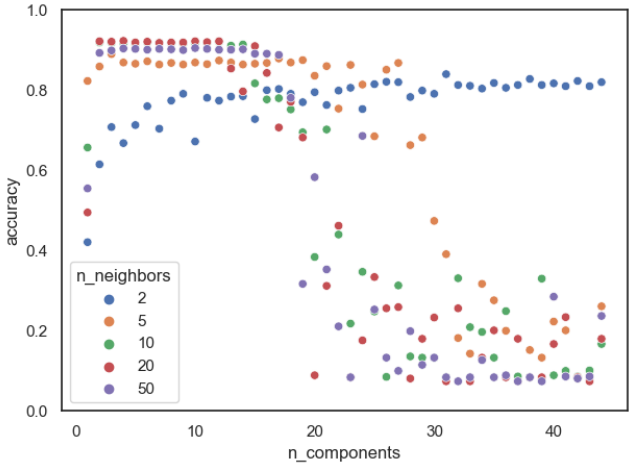

Figure 7. Accuracies over n_components in USPS

In the first 15 dimensions, the *n_neighbors* values that yield the highest accuracies are 20 and 50. When increasing *n_components*, however, the accuracies calculated with these values start to decrease. A similar phenomenon is observed when *n_neighbors* is 5, but its decrease happens for higher values of *n_components*. The cases where *n_neighbors* is 2, seem to behave inversely: when *n_components* is low, they yield a low accuracy, but they gradually keep climbing, until the last dimensions, in which setting *n_neighbors* to 2 is the only option to have a high accuracy score. In other datasets, such as MNIST and COIL-20, a similar pattern can be seen. More specifically, setting *n_neighbors* to low values like 2 or 5 also yields the highest accuracies in the last *n_components*. Another pattern common to many datasets is that the highest accuracy is reached in the first dimensions, specifically between dimension 3 and half of the value of the intrinsic dimension. In general, it can be said that some datasets seem to be best fitted with high values of *n_neighbors* when *n_components* is low and vice versa.

Number of Instances

The analysis of the results regarding a potential correlation between the hyperparameters and the number of instances starts with dividing the datasets into two groups: small sized datasets (COIL-20, Intel Image Classification, UMIST, USPS, ShipsNet) and relatively large datasets (the rest). This has been done because the number of tested datasets is small and does not allow to make an educated guess on the type of correlation. Grouping the datasets, on the other hand. highlights common patterns to a group, when compared against the other. Moreover, there is a large gap between Intel Image Classification with 11,885 instances, and CIFAR-10 with 50,000.

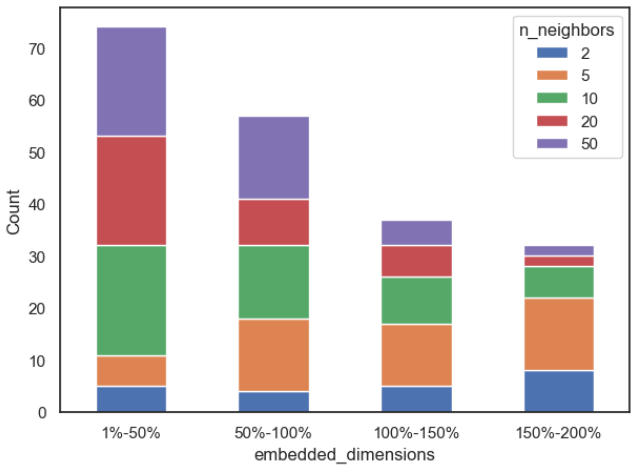

Figure 8. 50 best accuracies of large datasets

Figure 8 and Figure 9 show the 50 best accuracy scores from each of the big sized datasets and each of the small datasets respectively. On the x-axis, they are divided into 4 groups, depending on the value of the *n_components* used with regards to the intrinsic dimension: the first group spans from *n_components* equals one embedded dimension, to 50% of the value of the intrinsic dimension, and so on. The y-axis keeps the counts of instances per group.

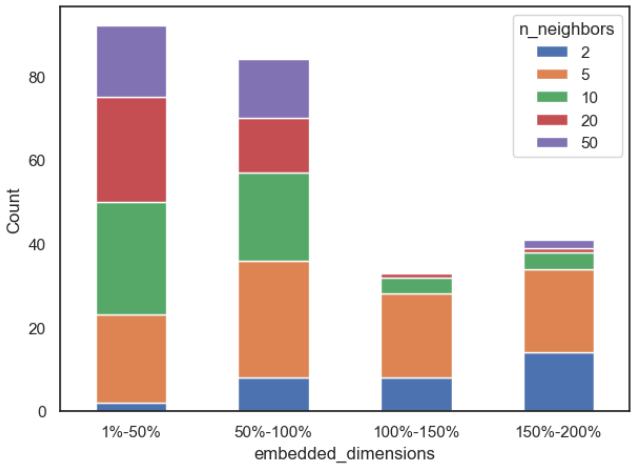

Figure 9. 50 best accuracies of small datasets

Datasets with large number of instances are better fitted with a large *n_neighbors* and an *n_components* smaller than the number of intrinsic dimensions. When *n_components* is large, a smaller *n_neighbors* will results in the higher accuracy. As shown in Figure 9 small sized datasets behave similarly. The only difference, being that *n_neighbors* of 5 is a good choice for almost any *n_components*. These results show that UMAP is capable of properly embedding both small and larger datasets, therefore, no lower bound has been found, under which UMAP is not capable of properly embedding a dataset. Moreover, its hyperparameters do not seem to be greatly influenced by the size of the dataset.

Number of Features

Much like the previous analysis on the number of instances, the datasets are now divided into a high-dimensional group, made of the MNISTs, the CIFARs and USPS, and a very high-dimensional group, containing the rest of the datasets.

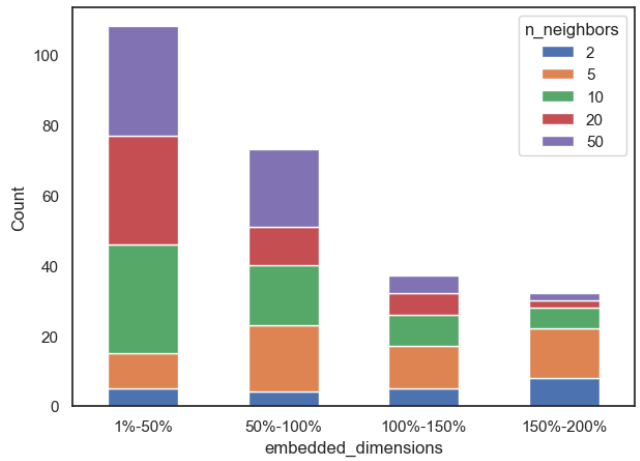

Figure 10. 50 best accuracies of high-dimensional datasets

The high-dimensional group follows the previously explained correlation between the two hyperparameters.

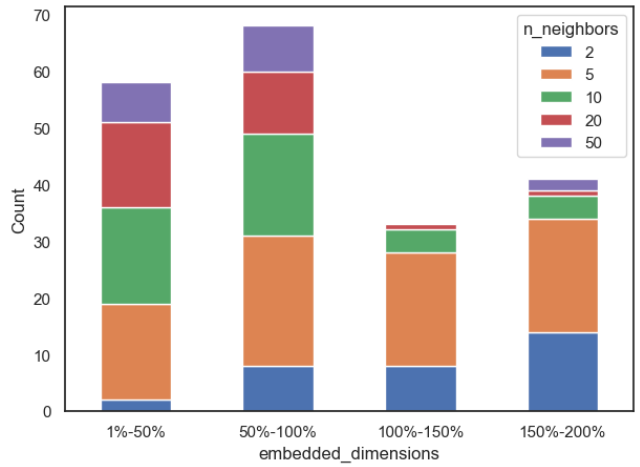

Figure 11. 50 best accuracies of very high-dim. datasets

The very high-dimensional group, however, does not seem to follow it as much. Instead, an *n_neighbors* of 5 seems to be preferred across the 4 columns. The preferred *n_components* are also not the ones in the first group, rather the ones in the second, closer to the value of the intrinsic dimension. This can be caused by two reasons. Maybe, higher-dimensional datasets need even higher *n_components*, to be correctly embedded, suggesting that the correlation between *n_components* and the intrinsic dimension is not linear but exponential. On the other hand, there could also be a certain percentage of error in the estimation of the intrinsic dimension, and the true intrinsic dimension is higher than the estimation.

Silhouette coefficient

The silhouette coefficient of the datasets varies between -0.11 and +0.16. Similar to the previous analysis, the datasets have been divided into two groups, those with a small silhouette and those with a big silhouette. The first group does not seem to follow any rule. The preferred *n_components* group is the second one and no particular *n_neighbors* is preferred.

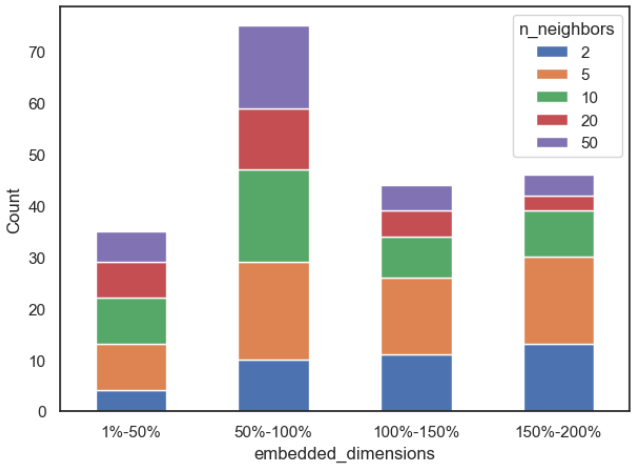

Figure 12. 50 best accuracies of datasets with small silhouette

Datasets with a high silhouette coefficient, closely follow the patterns previously found in datasets with relatively low number of features.

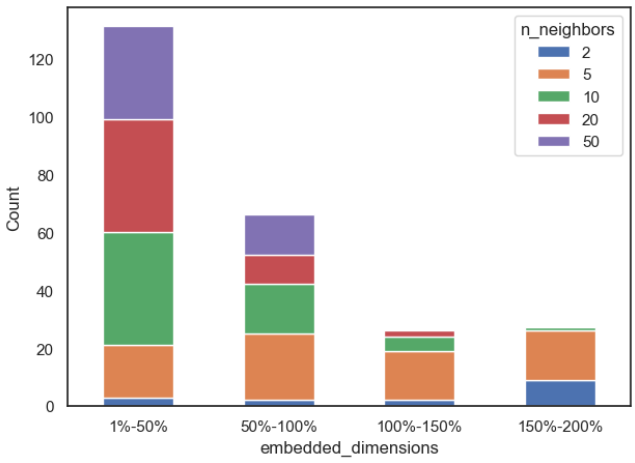

Figure 13. 50 best accuracies of datasets with big silhouette

The difference between these two groups could be due to the fact that datasets with a high silhouette coefficient are better clustered than those with a smaller silhouette and therefore easier to embed for UMAP: while small silhouette dataset embeddings might be highly case-dependant on the particular cluster structures and positions, therefore leaving no trace of correlation, high-silhouette datasets might be correctly embedded by UMAP in any *n_neighbors* *n_components* combination, leaving exposed the inner weaknesses of UMAP itself, like the difficulty to get good embeddings for high *n_components* and so on.

In summary, the two studied hyperparameter were found to be correlated to the value of the intrinsic dimension, the number of features and the value of the silhouette and uncorrelated to the number of training instances.

DISCUSSION

This work explores the existence of a possible correlation between the parameters of a dataset and the two most important UMAP hyperparameters, namely *n_components* and *n_neighbors*. To do so, 9 publicly available datasets, having very different parameter values, have been embedded into a lower dimensional space with a UMAP object initialized with different hyperparameter values. On each embedded dataset, K-Nearest Neighbours has then been trained and tested: its accuracy score has been used to determine the quality of the performed embedding. Every accuracy has then been saved to a file along with the corresponding hyperparameter combination used in the embedding and the dataset parameters. This process has been repeated with many hyperparameter combinations. In particular, each dataset has been embedded between 1 and two times the value of its intrinsic dimension and every embedding dimension has been tested for 2, 5, 10, 20, 50 *n_neighbors*. The value of *min_dist* has been set to 0 for every test, to get denser clusters.

From the results, there does not seem to be an advantage in setting *n_components* to be the value of the intrinsic dimension. Rather, most datasets are better embedded with an *n_components* in the range 3 to half of the value of the intrinsic dimension. This could be due to two reasons. On one side, UMAP does not seem to perform evenly across the various *n_components* test. Specifically, a low dimensional embedding will almost always outperform a higher dimensional embedding, maybe exposing that the UMAP algorithm is more prone to fail, when used with a high *n_components*. On the other side, the value of the intrinsic dimension used in this work is just an estimation of the true value and a possible error in the estimation might be the cause of this behaviour. More tests should be carried out with different estimators to properly assess that the number of components should not be set to the value of the intrinsic dimension. Moreover, the accuracy extracted from the KNN algorithm is just one of the many ways the embedding quality can be ranked. Future studies should also investigate adopting different distance or cluster-based algorithms.

A previously unknown correlation has however been found between the two hyperparameters. When UMAP manages to properly embed a dataset, the best possible results are obtained when *n_components* is relatively low (as explained before, between 3 and half of the intrinsic dimension) and *n_neighbors* is high (20 or 50). When increasing *n_components* however, the accuracy will overall decrease and *n_neighbors* will play a more important role in determining the accuracy: in this case, very low *n_neighbors* values such as 2 or 5 will lead to better results while the higher values will decrease the accuracy, up to 50%. This could imply that, when increasing *n_components*, UMAP tends to better preserve the local manifolds rather than the overall structure, therefore yielding the best results with a lower *n_neighbors*. Another study should be conducted by calculating the accuracy for more *n_neighbors* values.

Regarding the number of instances, there does not seem to be a significant difference in how UMAP embeds the dataset. The previously discovered correlation between *n_components* and *n_neighbors* is valid both for small and larger datasets. Therefore, no lower bound has been found, under which UMAP struggles to embed the dataset. The same cannot be said for the number of features: highly dimensional datasets (10,000+ dimensions) prefer an *n_components* between half and the full value of the intrinsic dimension, which suggests that the correlation between the intrinsic dimension and *n_components* might not be linear but exponential. Finally, datasets with high silhouette coefficient, which are in theory easier to embed, very closely follow the previously described *n_components* – *n_neighbors* pattern, while datasets with a lower silhouette coefficient are harder to predict and their best hyperparameter combinations are more case dependent.

CONCLUSION

In short, it has been found that datasets with relatively low number of features and/or high silhouette have a clear predictable preferred hyperparameter range of low *n_components* and high *n_neighbors*, while datasets with very high number of features and/or low silhouette are more case dependent, and the user should try out different hyperparameter combinations. Moreover, training size does not seem to impact the embedding and *n_components* should not bet set to the value of the intrinsic dimension.

Although this study was conducted on a very low number of datasets, some correlations between the hyperparameters and the dataset parameters appear to exist, however many more datasets should be tested to confirm them. Such data would also identify the precise type of correlation between each hyperparameter each dataset parameter singularly, allowing users to skip the search for the best embedding.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Jarmo Laaksolahti and my fellow classmates whose advice and support helped me throughout the making of this thesis.

REFERENCES

1. Mebarka Allaoui, Mohammed Lamine Kherfi, and Abdelhakim Cheriet. 2020. Considerably improving clustering algorithms using umap dimensionality reduction technique: A comparative study. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 317–325. https://doi.org/10.1007/978-3-030-51935-3_34

2. Jayme Garcia Arnal Barbedo. 2018. Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification. *Computers and Electronics in Agriculture* 153. https://doi.org/10.1016/j.compag.2018.08.013

3. Bjorn Barz and Joachim Denzler. 2020. Deep learning on small datasets without pre-training using cosine loss. *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*: 1360–1369. https://doi.org/10.1109/WACV45572.2020.9093286

4. Gunnar Carlsson, Tigran Ishkhanov, Vin De Silva, and Afra Zomorodian. 2008. On the local behavior of spaces of natural images. *International Journal of Computer Vision* 76, 1. https://doi.org/10.1007/s11263-007-0056-x

5. Jiachen Chen and W. Kenneth Jenkins. 2017. Facial recognition with PCA and machine learning methods. *Midwest Symposium on Circuits and Systems* 2017-Augus: 973–976. https://doi.org/10.1109/MWSCAS.2017.8053088

6. Sandeep Dutta and Eric Gros. 2018. Evaluation of the impact of deep learning architectural components selection and dataset size on a medical imaging task. 1057911, March 2018: 36. https://doi.org/10.1117/12.2293395

7. Evelyn Fix and J. L. Hodges. 1989. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique* 57, 3. https://doi.org/10.2307/1403797

8. Annie George. 2012. Anomaly Detection based on Machine Learning Dimensionality Reduction using PCA and Classification using SVM. *International Journal of Computer Applications* 47, 21: 5–8. https://doi.org/10.5120/7470-0475

9. Courville Aaron Goodfellow lan, Bengio Yoshua. 2016. *Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books*. MIT Press. Retrieved from http://www.deeplearningbook.org

10. Daniel B Graham and Nigel M Allinson. 1998. *Characterising virtual eigen signatures for general purpose face recognition*. Springer, Heidelberg.

11. R. Hammel. Ships in Satellite Imagery. Retrieved from https://www.kaggle.com/rhammell/ships-in-satellite-imagery

12. H. Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology* 24, 6. https://doi.org/10.1037/h0071325

13. Yuta Hozumi, Rui Wang, Changchuan Yin, and Guo Wei Wei. 2021. UMAP-assisted K-means clustering of large-scale SARS-CoV-2 mutation datasets. *Computers in Biology and Medicine* 131, February: 104264. https://doi.org/10.1016/j.compbiomed.2021.104264

14. Jonathan J. Hull. 1994. A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16, 5: 550–554. https://doi.org/10.1109/34.291440

15. Bettina Hüttenrauch, geb. Krämer. 2016. *Targeting Using Augmented Data in Database Marketing*. https://doi.org/10.1007/978-3-658-14577-4

16. Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. *... Science Department, University of Toronto, Tech. ...*. https://doi.org/10.1.1.222.9220

17. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11. https://doi.org/10.1109/5.726791

18. Hyunkwang Lee, Sehyo Yune, Mohammad Mansouri, Myeongchan Kim, Shahein H. Tajmir, Claude E. Guerrier, Sarah A. Ebert, Stuart R. Pomerantz, Javier M. Romero, Shahmir Kamalian, Ramon G. Gonzalez, Michael H. Lev, and Synho Do. 2019. An explainable deep-learning algorithm for the detection of acute intracranial haemorrhage from small datasets. *Nature Biomedical Engineering* 3, 3: 173–182. https://doi.org/10.1038/s41551-018-0324-9

19. Thomas Lefèvre, Patrick Chariot, and Pierre Chauvin. 2016. Multivariate methods for the analysis of complex and big data in forensic sciences. Application to age estimation in living persons. *Forensic Science International* 266: 581.e1-581.e9. https://doi.org/10.1016/j.forsciint.2016.05.014

20. Trond Linjordet and Krisztian Balog. 2019. *Impact of training dataset size on neural answer selection models*. Springer International Publishing. https://doi.org/10.1007/978-3-030-15712-8

21. Laurens Van Der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9: 2579–2625.

22. Faisal Mahmood, Richard Chen, and Nicholas J. Durr. 2018. Unsupervised Reverse Domain Adaptation for Synthetic Medical Images via Adversarial Training. *IEEE Transactions on Medical Imaging* 37, 12: 2572–2581. https://doi.org/10.1109/TMI.2018.2842767

23. Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv*.

24. Jakub Nalepa, Michal Marcinkiewicz, and Michal Kawulok. 2019. Data Augmentation for Brain-Tumor Segmentation: A Review. *Frontiers in Computational Neuroscience 13*. https://doi.org/10.3389/fncom.2019.00083

25. S Nene, S Nayar, and H Murase. 1996. Columbia Object Image Library (Coil-20). *Technical Report* 95.

26. Phillip Pope, Chen Zhu, Ahmed Abdelkader, Micah Goldblum, and Tom Goldstein. 2010. The Intrinsic Dimension of Images and its Impact on Learning. May: 1–16. Retrieved from https://arxiv.org/pdf/2104.08894.pdf

27. Miguel Romero, Yannet Interian, Timothy Solberg, and Gilmer Valdes. 2019. Training Deep Learning models with small datasets. *arXiv*.

28. Jonathon Shlens. 2014. A Tutorial on Principal Component Analysis. Retrieved from http://arxiv.org/abs/1404.1100

29. Mengying Shu. 2019. Deep learning for image classification on very small datasets using transfer learning. *Creative Components*: 14–21.

30. Michel Verleysen and Damien François. 2005. The curse of dimensionality in data mining and time series prediction. In *Lecture Notes in Computer Science*. https://doi.org/10.1007/11494669_93

31. Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. 1–6. Retrieved from http://arxiv.org/abs/1708.07747

32. Xin She Yang, Sanghyuk Lee, Sangmin Lee, and Nipon Theera-Umpon. 2015. Information Analysis of High-Dimensional Data and Applications. *Mathematical Problems in Engineering 2015*. https://doi.org/10.1155/2015/126740

33. Huanhuan Yu, Rongda Chen, and Guoping Zhang. 2014. A SVM stock selection model within PCA. In *Procedia Computer Science*. https://doi.org/10.1016/j.procs.2014.05.284

34. Intel Image Classification. Retrieved from https://www.kaggle.com/puneet6060/intel-image-classification