CE888 Project Report

Author: Hector Palop
School of Computer Science and Electronic Engineering (CSEE)
University of Essex, UK
Data Science and Decision Making (CE888)

***Abstract*—This document explores the viability of autoencoders as feature extractors analysed through experimentation on 3 very different datasets, striving to combat the infamous curse of dimensionality. The project aims to train a Neural Network predictor on each dataset that learns from the encoded features previously compressed by an autoencoder in order to improve the performance of the predictor, and then study how the predictor behaves when we limit the amount of data available to train the autoencoder.**

***Index Terms*—Autoencoders, Artificial Neural Networks, Feature Selection, Neural Networks, Dimensionality Reduction**

I. Introduction

The curse of dimensionality [1] is often referred to as one of the greatest challenges to overcome in Machine Learning. It states that the number of observable configurations grows exponentially as the dimensionality of the feature space increases. By extension, this means that as the number of features in a dataset increases, the scope of the prediction greatly decreases. This is also reflected in the *Hughes Phenomenon*, that incurs when the accuracy of a predictor first rises and then decreases as the number of dimensions (features) used in the dataset increases [2].

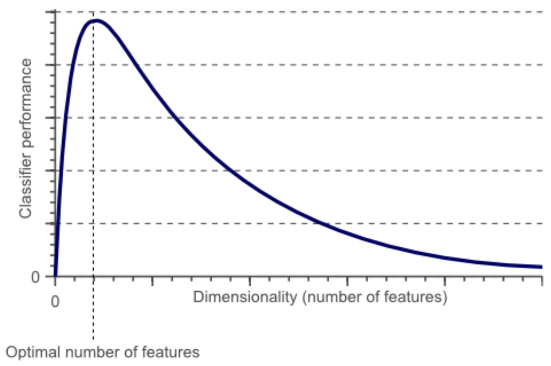

Fig. 1. Hugh Phenomenon representation [3].

Commonly, the variance of the model grows proportionally with the number of features present in the dataset, as the model is more likely to overfit to noise in a large dimensional space, which results in exhibiting a worse performance. For this reason, reducing the dimensionality of the dataset helps to remove potential sources of noise in the model while also saving up on computing resources, which is why we perform feature selection before training a model.

The question of which features are optimal is even more critical when dealing with small datasets, as every sample of data mistakenly discarded could result in a very heavy loss of accuracy of the predictor [4]. Although this task is usually achieved through a manual approach or through Principal Component Analysis (PCA), autoencoders can be used as feature extractors that provide a way to abstract the relevant information from the given data, in an attempt to automate and optimize the procedure of feature selection.

Therefore, the task at hand aims to evaluate effect of encoding all the usable features present in a dataset using an autoencoder before fitting a Neural Network model with the data. In order to confront this issue, 3 different datasets, with different sizes and natures have been selected to study the behaviour of autoencoders as feature extractors, specially on small datasets. The selected datasets are the following:

- **Dataset 1: Don't Overfit (II)** is a dataset retrieved from the *Kaggle* website, consisting of only 250 training abstract (with no physical meaning) samples and 200 features aiming to predict a binary output. The high dimensionality of the data was purposely favoured when selecting the dataset, in order to test the performance of the encoded in a high-dimensional space.
- **Dataset 2: Ships in Satellite Imagery** is an imagery dataset offering 4000 RGB images depicting bits of ships and background scenario. The target to be predicted is a binary label that indicates the presence of a ship appearing clearly in the image. The size of the dataset is large enough to not constitute as much of a complication as in the previous dataset. The intention behind this choice of data is the study of the use of the autoencoder in imagery as a data compressor rather than a feature extractor.
- **Dataset 3: North Korea Missile Test** is a fairly small dataset (only 117 samples) that shows the information gathered about the successful and unsuccessful missile tests performed by North Korea related to 19 independent features. The dataset was not built as a machine learning exercise (such as the previous ones), but rather as an informative panel that illustrates the western public knowledge of the military activities perpetrated by North Korea. Nevertheless, it allows us to perform an unprecedented classification task that predicts whether a missile test would be successful with the information available.

For each dataset, we will train an autoencoder and break it down into both the encoder and decoder sections. The features will then be encoded by feed-forwarding the data through the encoder. These encoded data can then be used to train and validate a Neural Network model with pre-encoded features to perform a supervised classification task.

II. Background

A. About feature selection

When working in a classic supervised learning task, we are usually given a set of labelled feature vectors that are used to shape the classifying algorithm. Although we can intuitively imagine that the more of these different features contributing towards the solution, the better the predictor will perform, this is usually not the case. When a model uses more terms than necessary to estimate a target variable, it exhibits an under-performing behaviour that is popularly known as overfitting [5]. In addition to this, having fewer data to process also reduces the training time and the necessity of a complex model [6], which can be critical in some situations.

Feature selection is the art of isolating the best features to be processed by a predictor in order to avoid over-fitting during the training, caused by feeding unnecessary information to the algorithm. This procedure has frequently been the object of study in Machine Learning applications [7] , as it plays a huge transformative role in the nature of the data that is used to obtain a predicting solution.

This process is usually carried out by analysing the level of importance presented by each feature according to the amount of information contributed towards the validity of the prediction. This feature importance can by estimated using many different methods [8] that ultimately rely on the amount of information that is derived from the well-known concept of information entropy (S), depicted in the following equation (2), where P_i is the probability value [9]:

$$S \;=\; -\sum P_i\; logP_i \tag{1}$$

A great deal of effort has been involved in perfecting the best feature selection techniques, which include mRmR, RELIEF, CMIM, Correlation Coefficient, BW-ratio, Interact, GA, SVM-REF and component analysis. Traditionally, most of the feature selection methods were split into either 'filters' (very fast but underperforming) or ' wrappers' (slower but bringing out better results). However, most of the modern methods previously mentioned end up combining both approaches to be able to deal with highly dimensional data without sacrifizing too much computing time. [10].

B. Artificial Neural Networks

Artificial Neural Networks have progressively gained their throne in numerous fields of Machine Learning, given their simplicity of usage as black boxes and their property as universal approximators [11].Neural Networks are massively distributed processors that have a natural propensity for storing experimental knowledge and making it available for use [12]. The neurons inside a neural network are distributed in arranged and inter-connected neuronal layers (Fig. 2), notably an input (X_i), output (Y_i)and a hidden (H_i) layer. This system resembles the brain in its faculty to store the knowledge acquired during the learning process in what we call inter-neuron synaptic weights.

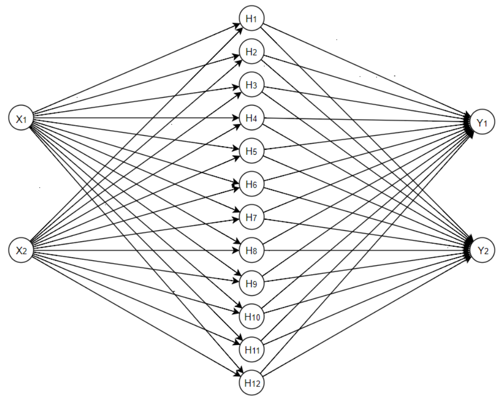

Fig. 2. Traditional neural network representation.

The software implementation of a traditional neural network is performed through artificial elements that go by the name of 'Perceptrons'(Fig. 3), giving the model the popular name 'Multi-Layer Perceptrons'(MLP). These serve as an algoritmic unit that is used to compute parallel linear combinations of neurons from the same layer, which is followed by activating the value through a pre-defined activation function ($\varphi(v)$), which is usually whether a linear, a logistic or a stem function. [13]

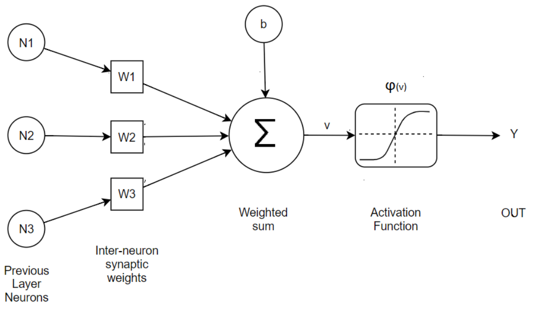

Fig. 3. Diagram of a perceptron.

The induced local field that results from the linear combination of the values in the input neurons can be interpreted as the following, with the values corresponding to the ones depicted in Fig. 3.

$$v \;=\; \sum W_i \cdot N_i \;+\; b \tag{2}$$

The algorithm that is used for training neural networks is called 'Back-Propagation', and it performs a stochastic gradient descent on a hyper surface based on the deviation (error) measured from the target variable. This can be interpreted as a sort of strength space, where the height of the curved surface represents the performance error [14]. If we consider η as the learning rate, E as the target error, and α as the momentum, the coupling strengths v of the neural network are updated with Back-Propagation through the derivative of the error (E) with a learning rate (η) and adding momentum (α):

$$\Delta v_{s+1} = -\eta \sum \frac{\delta E}{\delta v} + \alpha \Delta v_s \tag{3}$$

However, as efficient as this method has proven to be, the stochastic gradient descent does not guarantee to converge in the global minimum. The most effective and common ways of dealing with this issue are adding momentum (as depicted in Eq. 3) and a patience criteria that limits the training when no improvement is observed.

C. Autoencoders

Autoencoders are Neural Network architectures that are trained to attempt to reconstruct the given input in the output of the network after the compression of the input information. This can be visualized in Fig. 2 assuming $Y_i = X_i$. The network of an autoencoder (Fig. 4) can be broken down in two modular elements: an encoder function ($h = f(x)$) that compresses the input information and a decoder ($r = g(h)$) that attempts to reconstruct it. [15]

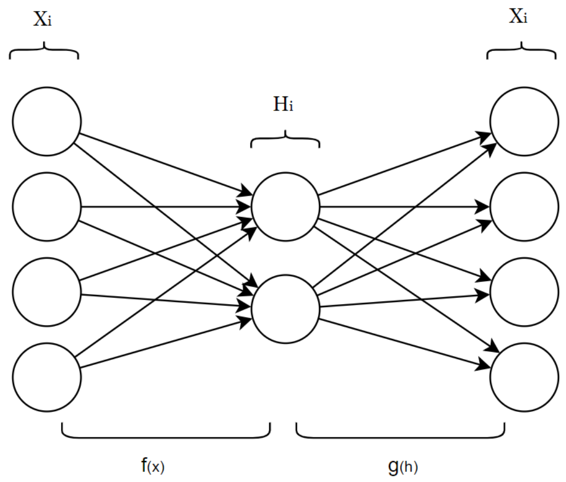

Fig. 4. Autoencoder network.

Note that the bottleneck present in the network forces a reduction on the dimensionality of the data ($dim(H_i) < dim(X_i)$), which constrains the amount of information that reaches the output. The utility of autoencoders lies in two fundamental applications:

- Dimensionality reduction and feature learning.
- Generative modelling.

When seeking to perform the former, only the first half is truly needed after fitting an autoencoder, since the encoder will be trained to be able to generate artificial features with less dimensionality than the original features. This turns out in theory to be a more powerful dimensionality reduction tool than Principal Component Analysis (PCA) [16], given that autoencoders are able of learning nonlinear manifolds.

D. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural network architectures that in the last few years have exhibited an unmatched success in image classification, as proven in the annual ImageNet competition [17]. They comprised of the following three main types of layers [18]:

- **Convolutional layers with ReLu** that apply a convolution kernel on every pixel of the input image by performing scalar products between the weights of the kernel and the inputs that are distributed across channels. The result is then activated through a ReLu function, thus providing activation maps of the image.
- **Pooling layers** that apply non-linear downsampling on the activation maps .
- **Fully connected layers** that form of a regular neural network that maps the extracted visual features into the target outputs.

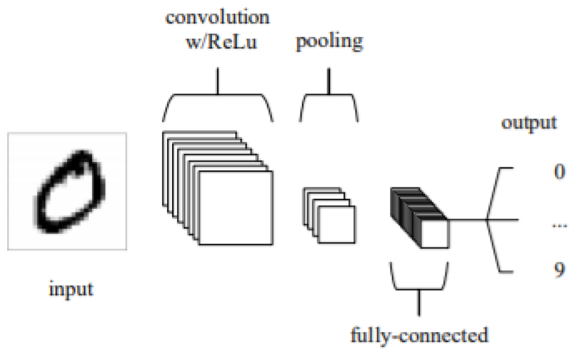

Fig. 5. Convolutional neural network. [18]

Both the convolutional and the pooling layers are usually found iterated several times before meeting the fully-connected layer at the end of the network architecture. Inside the convolution layer. Overall, we can find three main hyperparameters that can be tuned to optimize the performance of the convolutional network [19]:

- **The 'depth'** of the output volume set through the number of neurons present in the layer.
- **The 'stride'** in which the depth is set around the spatial dimensionality of the input.
- **The 'Zero-padding'** that controls the volume of the output by padding the border of the input.

Similarly to traditional neural networks, CNNs can also be trained to reconstruct an input forming convolutional autoencoders, generating feature maps that represent the presence of particular patterns (convolutional kernels) within the input image [20].

E. Similar work

Using autoencoders as feature extractors is not an uncommon practice in Machine learning applications. Dimensionality reduction is one of the most recurrent challenges in deep learning applications, which has led to various deep learning dimensionality reduction methods being proposed [21] as an alternative to the more popular PCA. In other recent studies [22], autoencoders have proven to clearly outperform PCA and other unconventional methods in the task of feature extraction. Their advantage is likely due to their non-linear learning capabilities that was previously mentioned.

The use of autoencoders is especially encouraged when it comes to dealing with imagery instead of numeric data, as the learned features are usually non-linear and too abstract for other methods. In addition to dimensionality reduction, they are also used to achieve image coloring, feature variation and denoising images. [23]

III. Methodology

A. Objective of the experiments

The main goal that is proposed for this project is to train a neural network model on the compressed features extracted by an autoencoder to perform a supervised classification task, and then comparing how the performance of the network changes when we modify the size of the subset used to train the autoencoder.

B. Overall process

The following top level diagram (Fig. 6) represents the main idea behind the overall process that will be performed in all three datasets, assuming a previous stage where the data is imported, reshaped and cleaned.

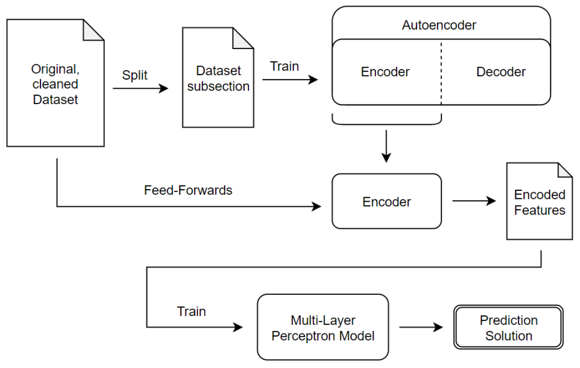

Fig. 6. Modelling process of the project.

The experiments performed on the three selected datasets build on the following generalized steps:

1) **Reading the data:** Whether the data is stored as a .csv file or as a folder with imagery and labels in a .json file, it needs to be first loaded in a *pandas* dataframe, assigning the target label to be predicted.
2) **Splitting the data:** In order to perform cross-validation, we need to split the data into train and validation subsets.
3) **Interpreting the data:** Before manipulating the data in any way it is always regarded as a good practice exploring the way the data is shaped and distributed, and locate any features that need any sort of 'treatment'.
4) **Cleaning the data:** It involves getting rid of any unusable feature, One-Hot-Encoding the categorical features, properly filling the empty cells and adapting all the features with an 'odd format' (such as timestamps).
5) **Tuning and training an autoencoder:** Once the data is prepared to be processed by a model, simply running it through a pre-made autoencoder is not enough. The autoencoder's hyperparameters (notably the number of layers and neurons in the autoencoder) need to be properly tuned to ensure that the features are efficiently compressed. When working with imagery, a convolutional autoencoder will be used instead.
6) **Feed-forwarding the data through the encoder:** The autoencoder must be broken down into its encoder and decoder sub-parts so that the features in the dataset can be compressed though the encoder.
7) **Tuning and training a discriminative neural network:** Once the features have been encoded, they are ready to be learned by a classification neural network model that also needs to be properly tuned for each dataset. Similarly to the autoencoder, when dealing with an imagery dataset, a convolutional neural network will be utilized to perform the classification task on the learned features.
8) **Resizing the training set used for the autoencoder:** Once the previous steps are subsequently built, we can modify the size of the training set used for fitting the autoencoder and annotate the changes in its performance.

C. Metric used for evaluation

Given the fact that all three datasets have sufficiently balanced classes, the accuracy of the prediction is representative enough to compare the model performances. If the datasets were heavily affected by data skew, other metrics such as *Recall* and *AUC* (area under the *ROC* curve) would have been more appropriate.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

The accuracy (Eq. 4), where FN = False Negatives, FP = False Positives, TN = True Negatives, and TP = True Positives, measures the number of correct predictions performed across the total number of predictions.

In another note, overfitting will definitely be one of the big things to look for and avoid given the small size of the datasets. To work our way around it, we will validate our models through cross-validation with a train-validation split of 75% and 25% respectively. Again, because the data is not heavily skewed, there is no need of performing stratified sampling to ensure that the data split is optimal.

D. Dataset 1: Don't Overfit (II)

This first dataset is retrieved from the *Kaggle* website, posing a very technical challenge against overfitting, given the dimensionality of the data. The dataset contains 250 training samples and 200 different features, with a binary output target to predict. The data was artificially produced and has no real or physical meaning.

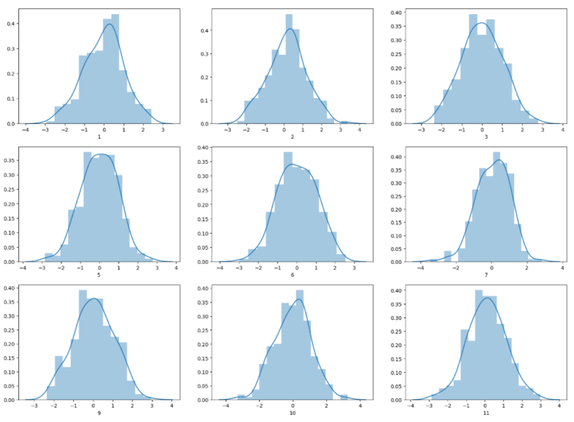
Fig. 7. First 9 features distributions.

Upon inspecting it (Fig. 7), it becomes clear that the features all follow a normal distribution with mean 0. Additionally, the data fortunately presents no missing values.

E. *Dataset 2: Ships in Satellite Imagery*

This dataset also originates from the Kaggle website. It contains a set of imagery samples decomposed from the grid of 8 separate scene landscapes collected over the San Francisco Bay and San Pedro Bay areas of California, USA. Out of the 4000 total samples, 3000 of those images are labelled with a Ships not present' (Fig. 8.A) label while the remaining 1000 are labelled with 'Ships present' (Fig. 8.B).

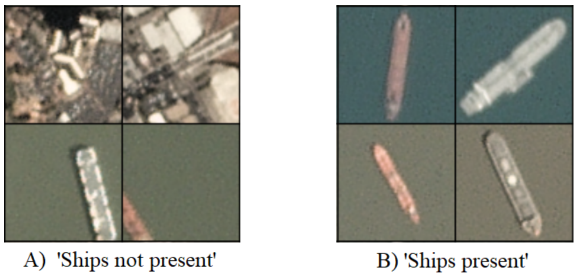

Fig. 8. Examples of labelled imagery samples.

Despite its name, this second label only contemplates the cases when a ship appears centered and uncut within the image. Furthermore, each of the images has a shape of 80×80 pixels whith 3 channels corresponding to the RGB color spectrums.

F. *Dataset 3: North Korea Missile Test*

This third dataset was provided by the James Martin Center for Nonproliferation Studies, and it gathers the record of all missile flight tests performed by North Korea that are capable of delivering payloads of at least 500 kilograms across a minimum of 300 kilometers, down to the first missile test documented in 1984.

The size of the dataset is substantially small, with only 117 samples. It contains information regarding the date and launch time of the test, the type of the missile, the facility location, the apogee and distance travelled by the missile and the outcome of the test, amongst others. This information is distributed across a total of 19 different features. Because the data was never gathered with the intention of being used to train a predictor, the features come on a variety of shapes (including many empty cells across different features) that require a previous 'data cleaning' stage before they can be useful to the objective of the project.

The target of the classification will be located in the binary *Test Outcome* feature, which is labelled as *Success* in the 75% of the dataset and as *Failure* in the remaining 25%.

IV. Experiments

The following sections present the operation run through all three datasets, following the steps described in the "Methodology" section of the report, and that are implemented in the *ipython* notebooks.

A. *Importing and preparing the data*

The first steps of the implementation involve importing the necessary libraries, reading the data into a *pandas* dataframe and randomly splitting the data into a train and a validation (or test) subset, corresponding to the 75% and 25% of the original datasets, respectively.

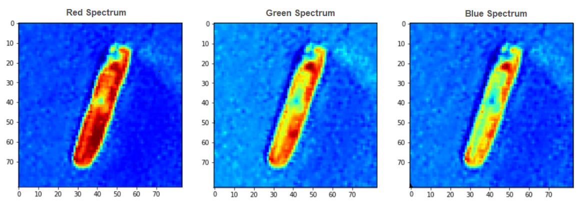

Fig. 9. Color spectrums gathered from a random sample.

For the second dataset (*Ships in Satellite Imagery*), it is also required importing and associating the target labels found in the adjacent *.json* file, and reshaping the data matrix to separate the 3 color channels of the imagery, which can be visualized in Fig. 9 for one of the samples.

B. *Cleaning the data*

Given that the first two datasets originate from the Kaggle website, the data are provided in an already convenient shape for training a model. However, the third dataset (*North Korea Missile Test*) presents several complications that require a special treatment for certain groups of features:

1) **Unusable features**
   a) Description: It involves all the features that come in a shape unusable by the model (e.g. plain text), features that are clearly going to have no correlation with the outcome (e.g. *Date Entered/Updated*), and those that are not known until the test is already performed (e.g. *Apogee*, *Distance Travelled*).

b) Features: *F1*, *Date Entered/Updated*, *Missile Name*, *Launch Agency/Authority*, *Confirmation Status*, *Additional Information*, *Apogee*, *Distance Travelled*, *Landing Location*, *Source(s)*.
c) Solution: Dropping (discarding) them from the dataset.

2) **Categorical features**
a) Description: Usable features that do not come in a continuous numerical form.
b) Features: *Missile Type*, *Facility Name*, *Facility Location*, *Other Name*.
c) Solution: Converting them to binary numerical dummies with One-Hot-Encoding.

3) **Timestamp features**
a) Description: Features that are presented in a timestamp format (YYYY-MM-DD HH:MM:SS).
b) Features: *Date*, *Launch Time*.
c) Solution: Converting them to a continuous numerical value representing the seconds elapsed.

4) **Numerical features with missing values:**
a) Description: Features that are of numerical nature and have empty or 'Unknown' values.
b) Features: *Launch Time (UTC)*, *Facility Longitude*, *Facility Latitude*.
c) Solution: Filling the missing values with random values taken from the normal distribution formed by the present values of the feature.

5) **Categorical features with missing values**
a) Description: Features of categorical nature with missing values in them.
b) Feature: *Other Name*.
c) Solution: Adding an extra 'Unknown' category to be encoded.

Once the suggested treatments are applied for all these features, the data are ready to be processed by a model.

C. Training and tuning the autoencoder

The intention of using an autoencoder in this project is utilizing its encoding property to reduce the dimensionality of the features, as represented in the following top-level diagram:

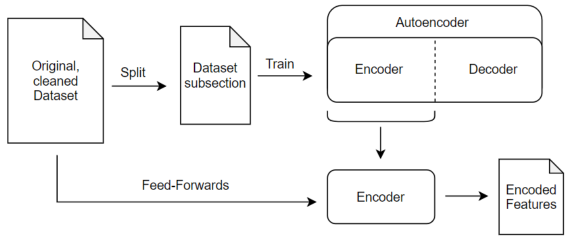

Fig. 10. Autoencoder functionality.

This process is therefore composed by 2 stages, the first one being tuning and training the autoencoder with a subsection of the dataset, and then feed-forwarding the entire dataset through the trained encoder to produce the encoded features. For the dataset containing the ship imagery, a convolutional autoencoder is used in an analogous fashion.

To train the autoencoder, we used simple trial and error along with hyperstochastical (*GridSearch*) methods to find the best encoding dimension. For the purposes of tuning the autoencoder, we used 100% of the data available in the datasets with a validation split of 20%. The optimal encoding dimension is therefore the one that presents the minimum validation error when training the autoencoder with the whole dataset. The following graph (Fig. 11) represents how the validation error changes depending on the autoencoder dimension for the dataset *Don't Overfit!(II)*, which reaches the global minimum at a lower dimension than the input's.

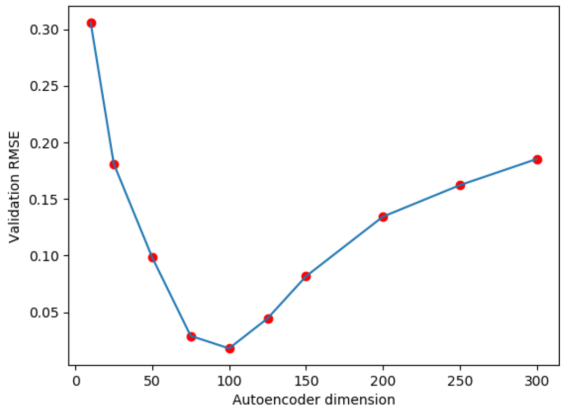

Fig. 11. Encoding dimension variation with the *Don't Overfit!(II)* dataset.

From this figure we can infer that the optimal number of encoding neurons for the *Don't Overfit!(II)* dataset is exactly 100. This loss was also optimized in terms of activation function, contrasting the best-performing *sigmoid* function to the *tangent sigmoid* and *ReLu* functions.

Similarly to the way in which these results were gathered from the first dataset, a similar process was conducted in the remaining datasets, which led to the following results:

TABLE I
AUTOENCODER TUNINGS

	Dataset 1	**Dataset 2**	**Dataset 3**
Type of autoencoder	Regular	Convolutional	Regular
Encoding layers	1	4	1
Encoding dimension	100	28x28,14x14,7x7,7x7	5
Initial dimension	300	80x80	9
Activation function	Sigmoid	ReLu	ReLu
Epochs	100	18	100

Once the autoencoder is tuned, it can be trained using a portion of the dataset and then used to feed-forwards the whole dataset as depicted in Fig. 10 to produce the desired encoded features.

D. Training and tuning the predictor

After the encoded features have been obtained from the autoencoder, they are ready to be fed to a neural network to effectuate the predicting task. It is well worth noting that although we are training the network to produce predictions on the target label, we are not that interested in the actual prediction as much as we are in the evaluation of the prediction, as it is the metric that will help us evaluate how beneficial has been the autoencoder.

The model used for the predictor is a Multi-Layer Perceptron that further reduces the dimensionality of the encoded features until the information converted into the final binary target. For this reason, the number of hidden neurons will always be comprised between the dimension of the encoded features and the dimension of the target ($dim_{OUT} = 1$), distributed across one or multiple hidden layers.

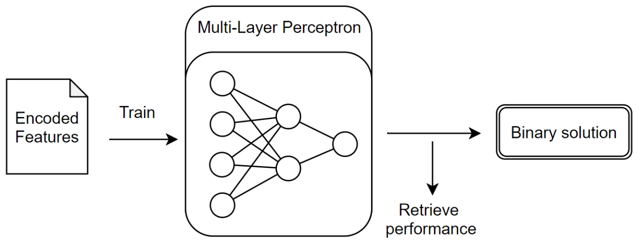

Fig. 12. Neural network functionality.

Similarly to the autoencoder, the neural network predictor requires a tuning process that was performed with the same conditions as the former, in terms of train/validation split and the portion of the dataset used for the tuning. The resulting hyperparameters for each dataset can be visualized in the following table:

TABLE II
PREDICTOR TUNINGS

	Dataset 1	**Dataset 2**	**Dataset 3**
Type	Regular	Convolutional	Regular
Hidden layers	2	4	1
Hidden dimension	50, 10	40x40,20x20,10x10,5x5	3
Activation function	Sigmoid	ReLu	Sigmoid
Epochs	100	18	100

From this now tuned MLP predictor, we can measure the obtained validation accuracy to evaluate the performance of the autoencoder, which will be covered in the following section of the report.

E. Resizing the data fed to the autoencoder

In this last section of the experiments, we aim to study how the performance of the predictor changes when we limit the amount of data that were previously used to train the autoencoder. This is measured by observing the validation accuracy of the final prediction while underfitting the autoencoder by limiting the data that it has access to the training. Note that despite modifying the amount of data used for training it, the hyperparameters are still the same ones described before, when we used the 100% of the training data for the tuning of the autoencoders.

The data obtained represented in the following graph (Fig. 13) captivates the experimentation results that were obtained through manually iterating the size of the autoencoder training set, which was performed in all three datasets.

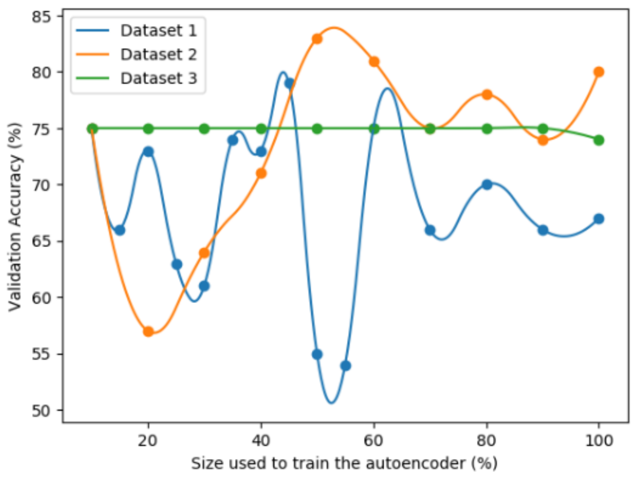

Fig. 13. Validation accuracy when resizing the datasets used to train the autoencoders.

V. Discussion

From Fig. 13 we can infer that there is no simple correlation between the size used to train the autoencoder and the overall performance of the predictor. The experiments show different behaviours for each different dataset. The behaviours observed in the datasets 1 and 3 seem very inconsistent, reaching many local maximums and minimums along the way, unlike the linear performance of the prediction on the second dataset.

The erratic behaviour observed in the prediction with the datasets 1 and 3 is likely to be due to the extremely small size of the original datasets. The correlations that should appear on the observation of the experiments are probably not consistent due to the lack of data available to train a robust model. The second dataset, however, is considerably large (4000 sample images in total), which explains why its accuracy remains unflappable during the experimentation if the autoencoder always has more than enough data to train on.

Nevertheless, we can extract a couple of common factors in the experimentation of all three datasets:

- The accuracy of the prediction performed with 10% of the dataset to train the autoencoder strangely converges at exactly 75% accuracy for each separate dataset. This has no logic explanation other than random variance.
- The global maximum of validation accuracy is achieved when using somewhere around half the original dataset to train the autoencoder.

This last remark demonstrates that at the very least, autoencoders have proven to be beneficial for the performance of a predictor, at least in small datasets. From the experimentation performed on the second dataset, we can infer that the imperturbability of the observed behaviour is likely due to the sheer size of the dataset, but it could also be caused by its nature as imagery, or to the nature of the convolutional models used to make the prediction.

VI. Conclusions

The datasets were intentionally selected because of their inconvenient shape. Whether it is the huge dimensionality, the imagery nature, or the cleaning process required, each one of the datasets added a unique level of complexity to the challenge. All these difficulties have been successfully overcomed in order to conduct the main experiments.

After analysing the results it becomes clear that limiting the size of the data used to train an autoencoder is factually beneficial to the performance of the predictor when working with small datasets. The optimal split ratio used to train the autoencoder is dependant on the dataset but it is likely to be close to 50%. When limit the amount of data available for the autoencoder, we are regularizing the autoencoder's capability to learn, thus avoiding overfitting when encoding the features.

References

[1] Hsieh, P. F., & Landgrebe, D. (1998). "Classification of high dimensional data." ECE Technical Reports, 52.

[2] Kppen, M. (2000, September). "The curse of dimensionality." In 5th Online World Conference on Soft Computing in Industrial Applications (WSC5) (Vol. 1, pp. 4-8).

[3] Online: Shetty, Badreesh. "Curse of Dimensionality. Towards Data Science, Towards Data Science, 15 Jan. 2019, towardsdatascience.com/curse-of-dimensionality-2092410f3d27.

[4] Pasini, A. (2015). "Artificial neural networks for small dataset analysis." Journal of thoracic disease, 7(5), 953.

[5] Hawkins, D. M. (2004). "The problem of overfitting." Journal of chemical information and computer sciences, 44(1), 1-12.

[6] Blum, A. L., & Langley, P. (1997). "Selection of relevant features and examples in machine learning." Artificial intelligence, 97(1-2), 245-271.

[7] Dash, M., & Liu, H. (1997). "Feature selection for classification." Intelligent data analysis, 1(1-4), 131-156.

[8] Kira, K., & Rendell, L. A. (1992). "A practical approach to feature selection." In Machine Learning Proceedings 1992 (pp. 249-256). Morgan Kaufmann.

[9] Shannon, C. E. (1948). "A mathematical theory of communication." Bell system technical journal, 27(3), 379-423.

[10] Khalid, Samina, Tehmina Khalil, and Shamila Nasreen. "A survey of feature selection and feature extraction techniques in machine learning." 2014 Science and Information Conference. IEEE, 2014.

[11] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with gaussian hidden units as universal approximations," Neural computation, vol. 2, no. 2, pp. 210-215, 1990.

[12] M. Kubat, "Neural networks: a comprehensive foundation by Himon Haykin, Macmillan, 1994, isbn 0-02-352781-7," Knowledge Engineering Review - KER, vol. 13, pp. 409-412, 02 1999.

[13] W. S. Sarle, "Neural networks and statistical models," 1994.

[14] A. Van Ooyen, B. Nienhuis, et al., "Improving the convergence of the backpropagation algorithm.," Neural Networks, vol. 5, no. 3, pp. 465-471, 1992.

[15] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep learning." MIT press, 2016.

[16] Malhi, A., & Gao, R. X. (2004). "PCA-based feature selection scheme for machine defect classification." IEEE Transactions on Instrumentation and Measurement, 53(6), 1517-1525.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, pp. 1097-1105, 2012.

[18] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arXiv preprint arXiv:1511.08458 (2015).

[19] Wu, Jianxin. "Introduction to convolutional neural networks." National Key Lab for Novel Software Technology. Nanjing University. China (2017): 5-23.

[20] Holden, Daniel, et al. "Learning motion manifolds with convolutional autoencoders." SIGGRAPH Asia 2015 Technical Briefs. ACM, 2015.

[21] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders" in Proceedings of the 25th International Conference on Machine Learning ser. ICML '08., New York, NY, USA:ACM, pp. 1096-1103, 2008.

[22] Tomar, Y. Prasad, M. K. Thakur and K. K. Biswas, "Feature Selection Using Autoencoders," 2017 International Conference on Machine Learning and Data Science (MLDS), Noida, 2017, pp. 56-60.

[23] Wang, Shuyang, Zhengming Ding, and Yun Fu. "Feature selection guided auto-encoder." Thirty-First AAAI Conference on Artificial Intelligence. 2017.