Assignment Part 2 – Final Report

Sergio A. de Leon Rivera

**Abstract**— Machine learning problems usually require large amounts of data to be solved. Nevertheless, it is sometimes impossible to gather these amounts of information. Studying whether it is possible to train classifiers with small datasets is therefore important. In this report, an Autoencoder will be trained with small samples of three different datasets to learn features that will then be used to train a classifier network. The results will be compared to state of the art models for each dataset.

—————————— ◆ ——————————

# 1 Introduction

T

HIS is the project proposal for the assignment of the module CE888 – Data Science and Decision Making, Project 1. In data science and machine learning, one of the most important elements that influence the success of the models is the training data. It is always desirable work with as much data as possible, but sometimes the amount of available data is limited. The question of whether it is possible to extract features from limited data arises from this problem. One of the possible ways to extract features from training examples is to use an Autoencoder. This approach will therefore be tested. To do this, three different datasets will be selected and an Autoencoder will be used to learn features from limited samples of each one. A classifier will be appended to the Autoencoder to use its learned features to attempt to classify the data. This will be achieved through a two-step training procedure: first using a greedy layer-wise training and then fine-tuning with the whole network. The results will be compared to state of the art models to evaluate whether the learned features are enough to successfully train a classifier.

# 2 Background

A vast amount of work has been made on learning from small datasets or data without enough examples, and a wide variety of approaches have been taken. Data augmentation is an approach that has proven to yield good results [1], leading to the development of variants. One of them is the technique called SMOTE, in which the minority class is oversampled by creating synthetic examples along the lines joining real ones [2]. In addition to this, undersampling the majority class achieved good results [2]. Mixup is another data augmentation technique in which neural networks are trained on combinations of features and their labels, favouring a linear behaviour between training examples, also having a regularising effect [3]. Wang et al. [4] used Generative Adversarial Networks (GANs) to forge training examples for the recognition of palmprints.

Another possible approach is to learn new features instead of increasing the number of training examples. Lin et al. [5] proposed a sample extending attribute method for improving performance in small datasets by integrating membership function values of fuzzy rules to create new attributes. Neural networks have also been used for this task.  Choi et al. [6] describe an approach for transfer learning in music classification, in which a feature vector is constructed from the activations of various layers of a pre-trained convnet. These features are then used for various learning tasks. Other works have used a special kind of neural network called Autoencoder, which are neural networks that, given an input, attempt to reconstruct it as its output [7]. Ota et al. [8] used an Autoencoder to generate features from clothing images to estimate users’ clothing preferences. The generated features were fed to a second neural net to predict the preferences. Similarly, Shen et al. [9] used Stacked Contractive Autoencoders to extract features from data to classify rotating machinery faults. Jia et al. [10] combined the properties of Autoencoders with those of Long-Short Term Memory Networks (LSTMs) to learn features from electroencephalogram data, demonstrating the effectiveness of these networks when there are missing features in the dataset.

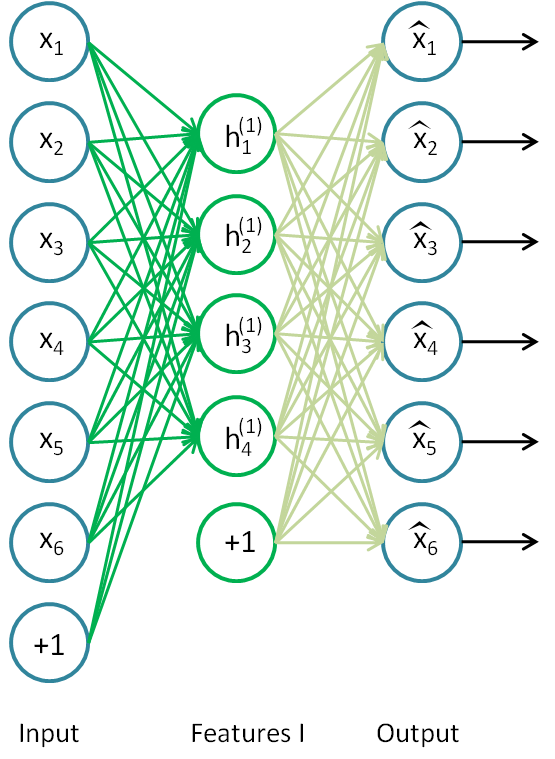


Fig. 1. Example of an undercomplete single-layer Autoencoder [15].

Autoencoders have also been used specifically for the problem of small datasets. Xue et al. [11] experimented using Autoencoders along with data augmentation through the addition of noise and with transfer learning with limited data. Liu et al. [12] described the training process for a Sparse Autoencoder and compared it with other feature extraction and dimensionality reduction methods, proving it to be more reliable. A detailed way on how to train a special kind of Autoencoder called Denoising Autoencoder is described in [13]. This type of Autoencoder will be used in addition to the basic one in the tests, and, although the training process is not entirely the same, most of it is and will be taken as a reference for the experiments.

# 3 Methodology

The goal of this study is to find, for each of the selected datasets, whether or not it is possible to learn useful features from a small sample using an Autoencoder and to successfully train a discriminative neural network with them. To achieve this, both, a basic and a Denoising Autoencoder, will be trained on samples of different sizes from the datasets. A discriminative neural network will be appended to the Autoencoder to test the learned features. It will classify the examples using the Softmax function. In the end, the performance of the various models will be evaluated using the Area Under the ROC Curve (AUC) [14]. This process, along with the explanation of the Autoencoder’s structure and activation functions will be explained next. In addition to this, an overview of the three datasets used for this study will be given.

## 3.1 The Stacked and Denoising Autoencoders

An Autoencoder is a neural network that attempts to reconstruct its inputs [7]. If the network is not constrained, then the task becomes trivial, as it learns just to copy its inputs. The idea is therefore to restrict them to only copy the input in an approximate way and, in this way, it learns the useful underlying properties of the data. Autoencoders usually consist of an encoder function and a decoder , where is the reconstruction of *x* [7]. To achieve this, the network is trained providing *x* as both, its input and expected output.

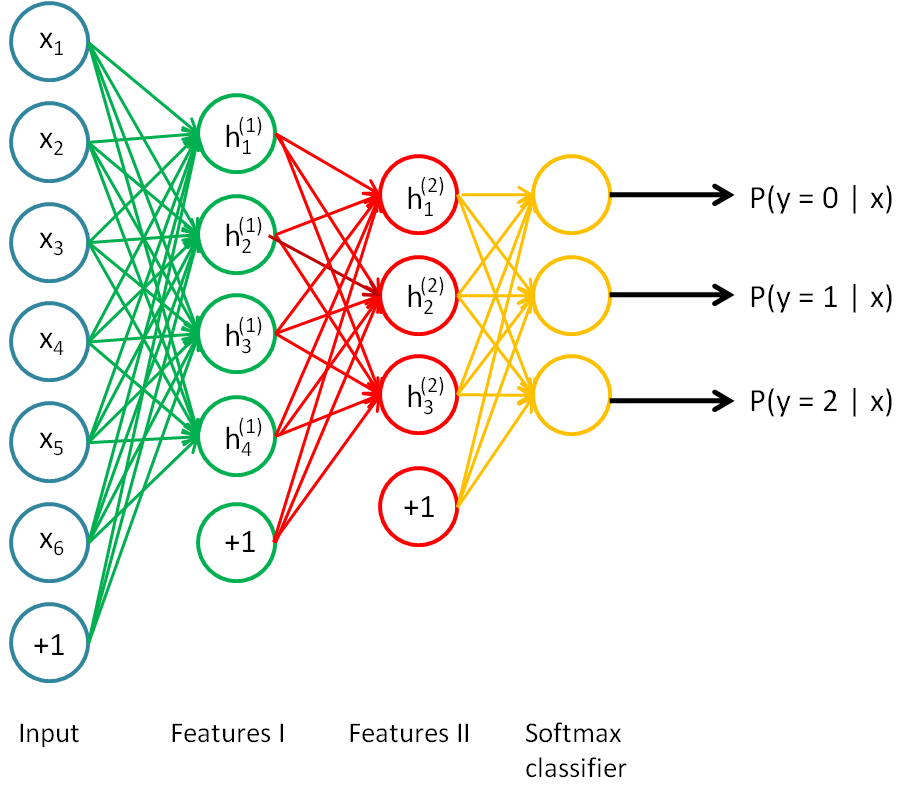


Fig. 2. Example of a Stacked Autoencoder followed by a Softmax layer [15].

There are various ways to constrain Autoencoders. One way is to set the hidden layer or “code” to be of a lower dimension than the input. This is called an undercomplete Autoencoder [7]. Fig. 1 shows this kind of Autoencoder.

Various Autoencoders can be stacked together to create a Stacked Autoencoder. When doing this, the network can be trained with a greedy layer-wise procedure by training pairs of layers, taking the first one as the input and expected output [14]. This method will be used in the experiments. Each layer will be trained to reconstruct the previous layer’s outputs with the help of a “dummy” decoder layer appended at the end, discarding them afterwards. New layers will be appended after each of the previous ones has been trained. Fine tuning can then be done with the complete network.

A Denoising Autoencoder is a variation in which the network is trained by initially corrupting the inputs *x* and training it with these corrupted inputs and the original *x* as output. In this way, the Autoencoder is expected to learn a mapping where is the corrupted input. This makes the Autoencoder more robust to noise.

Since the “code” of the Autoencoder can be thought of as a set of learned features from the input data, the classification will be made directly from it. Therefore, for this project, the decoder layers will be removed, exposing the hidden layer with the “code”.

## 3.3 The Softmax Function

The Softmax function is commonly used as the last layer of neural networks for classification problems [9]. In these problems, the dataset has the form where *X* is a vector of features, *y* is the class, and *N* is the number of training examples. The reason for using a Softmax function is that it produces an output vector (output layer) whose elements sum to 1, representing a probability distribution [7]. In this way, the element, or neuron, whose output value is larger, is selected as the network’s classification output. The Softmax function is defined by the following equation:

Where *z* is the dot product of the neuron’s weights times the activations of the previous layer, plus the bias term.

In order to use the features learned by the Stacked Autoencoder, the Softmax layer will be appended to it as shown in Fig. 2. In this way, the inputs of this layer will be the learned features. Since the problems in which the classifier will be tested are of binary classification, a single node in enough to distinguish between the two classes. Since just one node will be used, it is equivalent to use the Sigmoid function as activation instead of the Softmax, since Softmax is just a generalisation of Sigmoid for multi class problems. For computation performance purposes, Sigmoid will be used in the experiments instead of Softmax, but it should remain clear that they are equivalent in this context and that, to generalise for multi class experiments, it is sufficient to change the number of output nodes and from Sigmoid to Softmax in the output layer.

Training the whole network is then done in a two-step process, as described by [9]. The first step is pre training, which consists on training only the Autoencoder and was described previously. The second step is fine-tuning. In this step, the whole network is trained in a supervised way using the backpropagation algorithm, this time with the corresponding labels of the training examples. Training the network in this way not only considers the relationships between layers, but also makes it learn high level relationships between the features and labels [9].

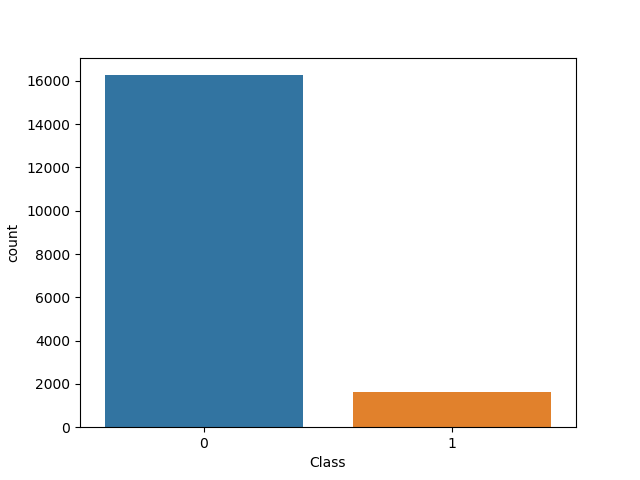
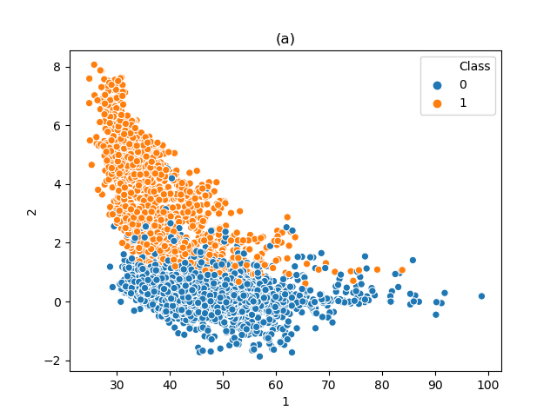


Fig. 3. Counts of the classes in the HTRU2 dataset. There are 16259 examples of the negative class, while there are just 1639 of the positive one.

## 3.4 The Datasets



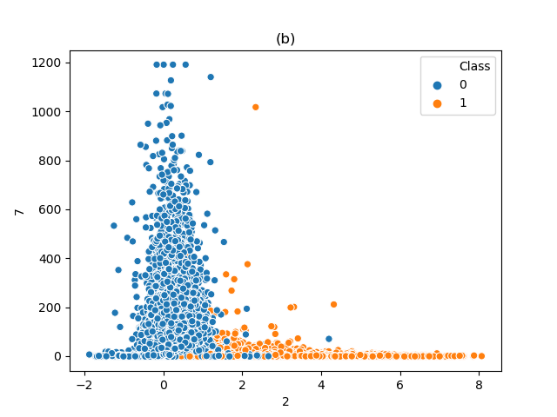


Fig. 4. Approximate linear separability when plotting features together. (a) Plot of Mean of the integrated profile (1) vs Standard deviation of the integrated profile (2). (b) Standard deviation of the integrated profile (2) vs Excess kurtosis of the DM-SNR curve (7).

Three datasets were selected, each of them with distinctive characteristics with which the proposed method should be tested. For each of them, the corresponding feature handling, such as one-hot encoding for nominal attributes, will be made, but feature reduction and other possible procedures such as PCA will be skipped because the aim is to carry them out automatically with the Autoencoders. Features were scaled to values between 0 and 1 as pre processing. An outline of each of the datasets will be given next.

### 3.4.1 HTRU2

The HTRU2 dataset from the UCI Machine Learning Repository [16] contains various statistics describing the pulse profile of pulsar candidates, which are a rare type of Neurtron star. It contains 17898 training examples, 1639 of which are real pulsar examples and the rest are RFI/noise. The goal is therefore to distinguish between these two classes. The features in the dataset are the following:

* Mean of the integrated profile
* Standard deviation of the integrated profile
* Excess kurtosis of the integrated profile
* Skewness of the integrated profile
* Mean of the DM-SNR curve
* Standard deviation of the DM-SNR curve
* Excess kurtosis of the DM-SNR curve
* Skewness of the DM-SNR curve
* Class

This dataset’s special characteristic is that it contains much more examples of the negative class than that of the positive one, as seen in Fig. 3. In addition to that, there are just eight features and the target class. This dataset specifically is where the model might benefit from the use of noise in the training process because of the limited examples of the positive class.

Another found characteristic is that the data seems to be somewhat separable when plotting some of the attributes together, as seen in Figure 4.

### 3.4.2 Arrhythmia

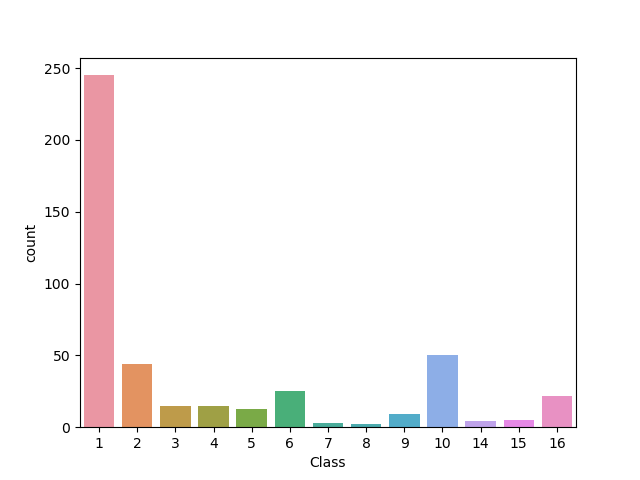


Fig. 5. Counts of the classes in the Arrhythmia dataset. Class 1 is by far the most common one.

The Arrhythmia dataset was also taken from the UCI Machine Learning Repository [16]. Its aim is to identify the presence of arrhythmia and to classify it as one of its 16 kinds. Each example contains 279 attributes, 73 of which are nominal and the rest are linear. While there is a large number of attributes, the number of examples is small, with a total of just 452. Furthermore, nominal attributes will be transformed into one-hot encoding for a better network performance, so the number of attributes will increase. This is why the feature-reduction capabilities of Autoencoders might become useful in this dataset. The class distribution is shown in Fig. 5. It is evident that class 1 is the most common one by far, followed by classes 10 and 2.

### 3.4.3 Phishing

Finally, the Phishing dataset, also extracted from the UCI Machine Learning Repository [16], has data from 2456 websites cathegorised as being a “phishing” website or not. There are 30 attributes, all of which are nominal. Each attribute has either two or three possible classes: *Phishing* and *Legitimate* for attributes with two classes, and *Suspicious* in addition to the other two for the ones with three. The class distribution for this dataset can be seen in Fig. 6. Classes are approximately well balanced.

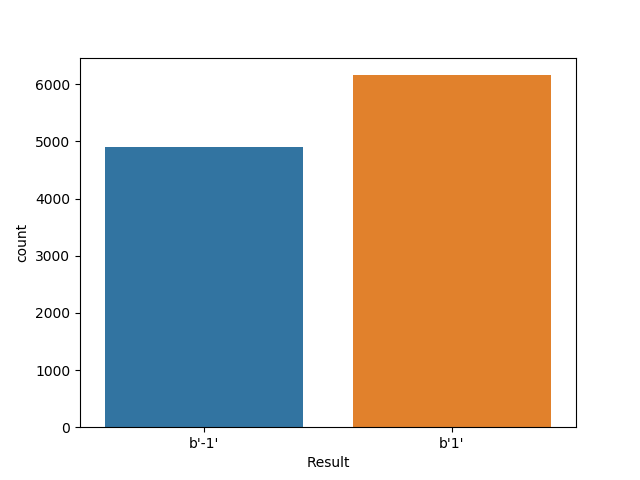


Fig. 6. Counts of the classes in the Phishing dataset. Class “-1” represents phishing websites and “1” represents legitimate ones.

## 4 Experiments

To test whether Autoencoders can learn features to effectively discriminate between classes with a neural network, a Stacked Autoencoder was trained in a greedy unsupervised way on different proportions of each dataset. This was done to test, for each dataset, how the classification accuracy varied with the training set’s size. The proportions used were 30%, 50%, 70%, and 100% of the dataset. After the greedy pre training, a dense layer connected to a single output node with a Sigmoid activation function was added to the Autoencoder. The whole network was then fine-tuned, now in a supervised way, using the same training partition as in the pre training and the rest of the dataset for testing, except for the 100% proportion, in which 70% of the data was used for training and the rest for testing.

In addition to this, all of the pre trained Autoencoders-Classifiers were also fine-tuned using 70% of the dataset as training data and the rest as testing data. This was done to compare the accuracies achieved when using either the whole dataset or just a portion of it for the fine-tuning. A small change in the accuracy between the two fine-tuning procedures might indicate that the pre training has a large weight in the final result and that, therefore, the Autoencoder is successful at learning useful features from the data partition. On the other hand, a large difference in the accuracies might indicate that much of the final result depends on the amount of training data used during the fine-tuning procedure, thus implying that the pre training might not have been as significant enough and that the Autoencoder was not capable of learning useful features on its own.

Different network sizes were tested for each dataset, which are listed in Tables 3, 4, and 5. This was done to compare results from various depths and amounts of latent features.

Finally, two different activation functions and pre training approaches were used. The first one was training the network layer by layer to reconstruct the outputs from the previous layer. In this approach, the Sigmoid activation function was used in all of the hidden layers. This was due to the layers reconstructing the previous layer’s output. Since the inputs were scaled to values between 0 and 1, the first layer’s outputs cover this range. In order to use the same loss function throughout the whole pre training, in this case binary crossentropy, all of the layers should output values in the same range.

The second approach was using the ReLu activation in all of the hidden layers, but training the layers to reconstruct the network’s input instead of the previous layer’s output. This could be done because of the “dummy” decoder layer with Sigmoid activation appended to the Autoencoder. Thanks to the decoder having the same range of activations as the input, it was possible to use binary crossentropy throughout the whole pre training even if the hidden layers’ activations covered a completely different range of values than the input. The results from both training procedures can be seen in Tables 3, 4, and 5.

Two metrics were used: accuracy and the ROC AUC. Accuracy is defined as the ratio of correctly labeled examples over the total amount of training examples. The default Keras implementation of this metric was used. The ROC AUC was explained in the previous sections. The Scikit Learn implementation was used for the experiments. The network was trained using binary crossentropy loss with the Adam optimiser.

Besides evaluating the accuracy of the predictions by the entire network, it might also be useful to evaluate the performance of the Autoencoder on its own. Nevertheless, given that its task is not to reconstruct the inputs, but rather to convert them into latent features in an unsupervised manner, there is no metric that can be used directly on its outputs. An option is to use the resulting binary crossentropy loss from the validation set during the last epoch of the Autoencoder’s last layer’s training as an indicator of its performance. The rationale behind this alternative is that, as it is necessary to propagate the inputs through the rest of the layers to train the last layer, its loss will be representative of the whole network. This metric, nevertheless, can be deceptive because it takes into account the “dummy” decoder layer appended to the end of the encoder, which is needed for its training. This decoder layer’s weights also have an impact on the loss function’s evolution, as they are also trained to reconstruct the inputs. Therefore it is hard to tell how much the results shown by the loss function will degrade when taking the decoder layer away. Also, this alternative only makes sense under the second pre training procedure, in which the inputs are reconstructed by each layer rather than the outputs of the layer before. If it were used in the second case, the metric would only measure how well the last layer reconstructs the previous layer’s output. Despite these disadvantages, the binary crossentropy loss from the Autoencoder’s training will also be used as a metric, but only for the second pre training procedure, for the purpose of analysing its performance in some way.

## 5 Discussion

**Table 2: State of the Art Results**

|  |  |
| --- | --- |
| **Dataset** | **Accuracy** |
| HTRU2 | 98% [3] |
| Arrhythmia | 68% [17] |
| Phishing | 92.48% [18] |

**Table 3: Best Results for HTRU2 Dataset per Training Procedure**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Proportion of Dataset** | **Pre training Procedure** | **Fine-tuning Procedure** | **Encoder Layer Sizes** | **Accuracy** | **ROC AUC** |
| 30% | 1 | 1 | 5 | 97.55 | 0.9708 |
|  | 1 | 2 | 5 | 97.65 | 0.9786 |
|  | 2 | 1 | 4 | 97.57 | 0.9721 |
|  | 2 | 2 | 4 | 97.67 | 0.9777 |
| 50% | 1 | 1 | 5 | 97.67 | 0.9757 |
|  | 1 | 2 | 5 | 97.64 | 0.9785 |
|  | 2 | 1 | 4 | 97.77 | 0.9760 |
|  | 2 | 2 | 5 | 97.75 | 0.9774 |
| 70% | 1 | 1 | 5 | 97.71 | 0.9782 |
|  | 1 | 2 | 4 | 97.69 | 0.9776 |
|  | 2 | 1 | 5 | 97.69 | 0.9776 |
|  | 2 | 2 | 5 | 97.75 | 0.9768 |
| 100% | 1 | 1 | 5 | 97.67 | 0.9786 |
|  | 1 | 2 | 5 | 97.65 | 0.9782 |
|  | 2 | 1 | 5 | 97.78 | 0.9789 |
|  | 2 | 2 | 5 | 97.65 | 0.9778 |

**Table 4: Best Results for Arrhythmia Dataset per Training Procedure**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Proportion of Dataset** | **Pre training Procedure** | **Fine-tuning Procedure** | **Encoder Layer Sizes** | **Accuracy** | **ROC AUC** |
| 30% | 1 | 1 | 128 | 72.11 | 0.7771 |
|  | 1 | 2 | 256 | 72.22 | 0.8180 |
|  | 2 | 1 | 128 | 71.77 | 0.7834 |
|  | 2 | 2 | 256 | 76.19 | 0.8289 |
| 50% | 1 | 1 | 128 | 70.95 | 0.8041 |
|  | 1 | 2 | 256 | 70.63 | 0.8124 |
|  | 2 | 1 | 256 | 74.76 | 0.8028 |
|  | 2 | 2 | 128 | 74.60 | 0.8235 |
| 70% | 1 | 1 | 256 | 72.44 | 0.8146 |
|  | 1 | 2 | 256 | 72.22 | 0.8159 |
|  | 2 | 1 | 256 | 75.59 | 0.8168 |
|  | 2 | 2 | 256 | 76.19 | 0.8235 |
| 100% | 1 | 1 | 256 | 72.22 | 0.8149 |
|  | 1 | 2 | 256 | 71.43 | 0.8167 |
|  | 2 | 1 | 128 | 73.02 | 0.8241 |
|  | 2 | 2 | 256 | 75.40 | 0.8238 |

As previously explained, the accuracy and AUC ROC metrics will be obtained for each of the sizes of each of the datasets. The ROC is useful for analyzing the proportion of true positives and false positives, especially in the skewed datasets. Confusion matrices might also be useful for these cases. The models and training sizes that achieve the best metrics will then be compared to the state of the art results.

**Table 5: Best Results for Phishing Dataset per Training Procedure**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Proportion of Dataset** | **Pre training Procedure** | **Fine-tuning Procedure** | **Encoder Layer Sizes** | **Accuracy** | **ROC AUC** |
| 30% | 1 | 1 | 32 | 94.68 | 0.9910 |
|  | 1 | 2 | 24 | 95.48 | 0.9934 |
|  | 2 | 1 | 32 | 93.99 | 0.9878 |
|  | 2 | 2 | 24 | 94.57 | 0.9893 |
| 50% | 1 | 1 | 32 | 95.39 | 0.9927 |
|  | 1 | 2 | 32 | 95.93 | 0.9939 |
|  | 2 | 1 | 32 | 93.95 | 0.9885 |
|  | 2 | 2 | 24 | 95.06 | 0.9915 |
| 70% | 1 | 1 | 32 | 95.78 | 0.9932 |
|  | 1 | 2 | 32 | 95.63 | 0.9938 |
|  | 2 | 1 | 32 | 94.27 | 0.9892 |
|  | 2 | 2 | 32 – 16 | 94.51 | 0.9880 |
| 100% | 1 | 1 | 24 | 95.72 | 0.9936 |
|  | 1 | 2 | 32 | 96.05 | 0.9938 |
|  | 2 | 1 | 24 | 94.21 | 0.9887 |
|  | 2 | 2 | 32 | 94.69 | 0.9899 |

## 6 Conclusion

Given the training process and architecture of the network, the experiments should be straightforward to perform. The results of the larger datasets might be promising, but, at least in the Arrhythmia dataset, the state of the art is 68%. Given its small size, it is expected that the smaller sample sizes will underperform. If it doesn’t, though, the Autoencoders might prove to be a powerful feature learning method for small datasets.

**References**

|  |  |  |  |
| --- | --- | --- | --- |
| [1] | P. Y. Simard, Y. A. LeCun, J. S. Denker and B.  Victorri, "Transformation Invariance in Pattern  Recognition - Tangent Distance and Tangent  Propagation," in *Neural Networks: Tricks of the*  *Trade*, Berlin, Springer, Berlin, Heidelberg, 2012, pp.  235-269. | | |
| [2] | N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P.  Kegelmeyer, "SMOTE: Synthetic Minority  Over-sampling Technique," *Journal of Artificial*  *Intelligence Research,* vol. 16, pp. 321-357, 2012. | | |
| [3] | H. Zhang, M. Cisse, Y. N. Dauphin and D.  Lopez-Paz, "mixup: Beyond Empirical Risk  Minimization," in *ICLR 2018*, Vancouver, 2018. | | |
| [4] | G. Wang, W. Kang, Q. Wu, Z. Wang and J. Gao,  "Generative Adversarial Network (GAN) Based  Data Augmentation for Palmprint Recognition," in  *2018 Digital Image Computing: Techniques and*  *Applications (DICTA)*, Canberra, 2018. | | |
| [5] | L.-S. Lin, D.-C. Li, H.-Y. Chen and Y.-C. Chiang,  "An attribute extending method to improve learning  performance for small datasets," *Neurocomputing,* vol.  286, pp. 75-87, 2018. | | |
| [6] | K. Choi, G. Fazekas, M. Sandler and K. Cho,  "Transfer Learning for Music Classification and  Regression Tasks," in *18th International Society of*  *Music Information Retrieval (ISMIR)*, Suzhou, 2017. | | |
| [7] | I. Goodfellow, Y. Bengio and A. Courville, Deep  Learning, Cambridge: MIT Press, 2016. | | |
| [8] | | S. Ota, H. Takenouchi and M. Tokumaru, "Kansei  clothing retrieval system using features extracted by  autoencoder," in *2017 IEEE Symposium Series on*  *Computational Intelligence (SSCI)*, Honolulu, 2017. | |
| [9] | | | C. Shen, Y. Qi, J. Wang, G. Cai and Z. Zhu, "An  automatic and robust features learning method for  rotating machinery fault diagnosis based on  contractive autoencoder," *Engineering Applications of*  *Artificial Intelligence,* vol. 76, pp. 170-184, 2018. |
| [10] | | | Y. Jia, C. Zhou and M. Motani, "Spatio-temporal  autoencoder for feature learning in patient data with  missing observations," in *2017 IEEE International*  *Conference on Bioinformatics and Biomedicine (BIBM)*,  Kansas City, 2017. |
| [11] | | | J. Xue, P. P. K. Chan and X. Hu, "Experimental  study on stacked autoencoder on insufficient training  samples," in *2017 International Conference on Wavelet*  *Analysis and Pattern Recognition*, Ningbo, 2017. |
| [12] | | | J. Liu, C. Li and W. Yang, "Supervised Learning  via Unsupervised Sparse Autoencoder," *IEEE Access,* p. 1,  2018. |
| [13] | | | P. Vincent, H. Larochelle, Y. Bengio and P.-A. Manzagol,  "Extracting and composing robust features with  denoising autoencoders," in *Proceedings of the 25th*  *International Conference on Machine Learning*, Helsinki, 2008. |
| [14] | | | E. Alpaydin, Introduction to Machine Learning, Cambridge:  The MIT Press, 2014. |
| [15] | | | Kandeng, "Unsupervised Feature Learning and Deep  Learning," Stanford, 31 March 2013.  [Online]. Available: http://ufldl.stanford.edu/wiki/index.php?title=  Stacked\_Autoencoders&oldid=2296. [Accessed 17 February 2019]. |
| [16] | | | D. Dua and E. Taniskidou, "UCI Machine Learning Repository," University of California, School of Information and Computer Science, 2017. [Online]. Available: http://archive.ics.uci.edu/ml. |
| [17] | | | H. Guvenir, B. Acar, G. Demiroz and A. Cekin, "A supervised machine learning algorithm for arrhythmia analysis," in *Computers in Cardiology 1997*, Lund, 1997. |
| [18] | | | R. M. Mohammad, F. Thabtah and L. McCluskey, "Predicting Phishing Websites based on Self-Structuring Neural Network," *Neural Computing and Applications,* vol. 25, no. 2, pp. 443-458, 2014. |