Using Variational Autoencoders to Increase the Performance of Malware Classification

Thomas Taylor
School of Computing, Mathematics & Digital Technology
Manchester Metropolitan University
Manchester, United Kingdom
ttaylor3.142@gmail.com

Amna Eleyan
School of Computing, Mathematics & Digital Technology
Manchester Metropolitan University
Manchester, United Kingdom
A.Eleyan@mmmu.ac.uk

***Abstract*—Often complex datasets will have a large number of features for each of its samples. Sometimes, this can have a negative effect on the performance of models trained on the raw data. By reducing the number of features this problem can be avoided. However, this may cause a loss of information. One method to mitigate this is by using a type of unsupervised neural network structure called autoencoders. Autoencoders can be used to generate a reduced feature space which the models can then be trained on. This paper uses Convolutional Variational Autoencoders in order to create these latent features and then determines their effectiveness of improving performance of machine learning classifier models.**

***Index Terms*—Machine Learning, Neural Networks, Autoencoders, Variational Autoencoders, Cybersecurity, Malware, Malware Classification**

I. Introduction

Cybersecurity is an extremely important field in modern science. As the use of computers continues to increase so will the amount of software, both malicious and benign. In order to protect new and existing systems precautions need to be developed. Developing general purpose solutions to identify malicious attackers is one method to solve this issue. This report looks at a malware datasets and constructs models which are able to differentiate the various classes of malware.

Currently there is a large interest in the field of machine learning. It is used in all sorts of applications from game theory [3] to healthcare [1]. Specifically there has been large amounts of research into the field of Neural Networks (NNs). NNs provide a novel way of solving non-trivial classification and regression problems. By varying the format of the data and the shape of the network, NNs are able to solve a massive range of problems. This report will look into how a specific form of NNs, Variational Autoencoders (VAEs), are able to increase the performance of other machine learning algorithms.

A. Motivation and Impact

There is a large drive in cybersecurity to develop new methods to detect malicious executables. This paper looks into malware detection and investigates a novel method to attempt to increase performance of existing machine learning classifiers. If this paper determines that VAEs have a positive influence on the performance of machine learning classifiers, this may provide a method to increase the performance of malware detection software.

B. Related Work and Contribution

Plenty of research has been done in the field on ML but as the field and the applications of autoencoders are so broad, relatively little work has gone into looking in how they can improve classification algorithms. The paper that most inspired this work was [24]. This paper uses a standard autoencoder to generate latent features and then compares the accuracy of a variety of ML algorithms with and without the latent features. It uses Microsoft malware datasets for experiments. Naive Bayes, K nearest neighbours, SVMs, and gradient boosting were trained on these latent features. With the malware dataset, Gaussian Naive Bayes, K-NN (neural network), and SVMs increased their accuracy between 0.7 and 14.2 percent.

[5] focuses on the extraction method of features from the deconstructed assembly files. It uses opcode n-gram and opcode n-gram with control statement shingling in order to extract features and then trains random forests on the data in order to try and find the most effective method. These methods uses a combination of hashing and grouping in order to create these representations. It was found that with random forests, opcode 2-gram was the most accurate method of features extraction for the Microsoft malware dataset.

[20] uses the Microsoft Malware Classification Dataset as well as benign executables sourced from the GNU `binutils` package to compare the effectiveness of various classifiers with different dimension reduction techniques. It uses variance thresholding and autoencoders for the dimensional reductions, and random forests, and deep neural networks for the classifiers. The aim of this paper was to find the most effective method of malware detection. Using 3-cross fold validation to calculate the accuracy, the maximum metric achieved was 99.78% using variance thresholding and random forest.

This project explores an unused aspect of convolutional variational autoencoders. Existing machine learning classifier algorithms are trained with latent features generated by VAEs to discover if this application of VAEs improves performance. This technique has never been used before on the dataset this paper uses. The performance of this VAE technique is also compared against the effect of using no dimensional reduction techniques, using PCA thresholding, and also using a combination of VAE and PCA.

II. Background

A. Malware

Malware is a type of malicious software that attempts to control and spread to other computers. In the Microsoft malware challenge dataset (see section III) there are 9 distinct types that are classified.

- Kelihos_ver1 & Kelihos_ver3: Botnet bitcoin miner/stealer [19]
- Lollipop: Pop-up advertisements [9]
- Ramnit: Trojan credential stealer [12]
- Obfuscator.ACY: Malware obfuscater [11]
- Gatak: Trojan information scraper [10]
- Tracur: Trojan which redirects web searches to malicious URLs [14]
- Vundo: Pop-up advertisements [15]
- Simda: Trojan that downloads and executed more malware [13]

B. Machine Learning Algorithms

1) Bayes' Theorem and Bayesian networks: Bayesian networks attempt to classify data using a probabilistic model. They use Bayes' theorem in order to determine what the most likely class is of a data point given the data's values ([18]).

2) Support Vector Machines: Support Vector Machines (SVMs) attempt to find the best hyperplane in N-dimensional space that separates distinct classes [2].

When constructing a hyperplane the impurity of each division that the hyperplane produces is minimised. This is effective to do in the case that the classes are linearly separable, but in the opposite case the performance is poor.

3) Radial Basis Function: Often classes will not be linearly separable to such extent that attempting to fit a line to the samples will prove to be extremely ineffective. A method called the *kernel trick* can be applied to the vector space of the samples which adds extra dimensionality which hopes to create linear bounds between classes.

One method to create this extra dimensionality is using the Radial Basis Function (RBF) [22]:

$$K(x, x') = \exp(-\gamma||x - x'||^2) \tag{1}$$

For each sample point, the euclidean distance is calculated between it and every other sample point, and that new dimension is added.

C. Neural Networks

Neural networks are a type of machine learning inspired by a brain's neurons. A networks is made up of individual neurons. Each neuron can take some input, put that input through some sort of weighted function, and then output a value. These networks get far too big for a human to manually set the parameters, so they are trained via a technique called backpropagation.

Each neuron is defined as a function that takes in any number of inputs and produces a single output. The output z is defined in (2) where σ is the sigmoid function b is the activation, w is the weight of a input and a is the value of the input.

$$z = \sigma(b + \sum_{i}^{n} w_i a_i) \tag{2}$$

1) Training and Backpropagation: As the size of a network increases, the number of parameters increase exponentially. It is therefore impossible to manually tune all of the weights, and so a way of programmatically reducing the models error is used. Backpropagation is a method used to calculate a gradient that is needed for adaptation of a neural network [6] or the gradient in weight space for a neural network with respect to a loss function.

Note that this algorithm only finds the gradient of the loss function for each of the networks parameters. In order to train the network the parameters have to be altered by some function of that gradient. The most common way to do this is **Stochastic Gradient Descent** (SDG). The basic form of this first calculates the gradient of the loss function using backpropagation, then changes the variables of each node from the gradient of the loss function for each variable multiplied by a learning rate. This process is then repeated with the new values for the neural network. This results in the model gradually walks down the gradient until hopefully it finds a point of minimum loss, where the model performs as accurately as possible for the training data.

D. Autoencoders

Autoencoders are a type of neural network. They take a data samples as input and attempt to replicate that same data point. Inside the network they have a bottleneck layer of neurons. This layer can be used to generate latent features used for dimensional reduction [7].

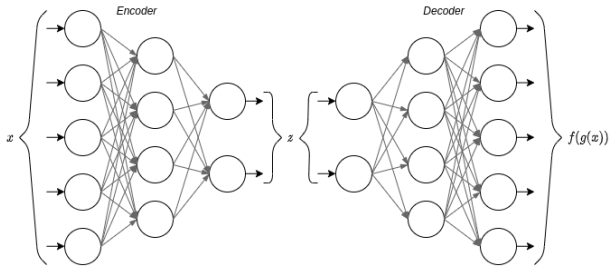

Fig. 1. Example of an autoencoder with 5 input features and 2 latent features.

An autoencoder is trained as one, i.e. the encoder and decoder are trained together placed in series. The loss is calculated by comparing some function of the difference between x and $f(g(x))$ where x is a data entity, f is the decoder function, and g is the encoder function (for example the mean squared error). The latent features are also denoted at z but note that $z = g(x)$.

1) Convolutional Variational Autoencoders: Variational autoencoders apply a probabilistic approach to the autoencoder model. Like a regular autoencoder they are trained to replicate a data point, but at the bottleneck layer instead of just passing

the computed latent feature values, a distribution is sampled around the computed point and then passed onto the decoder. The idea behind this is that similar feature values should produce similar results.

As a result of this probabilistic take the loss function is also different. Instead of looking at the difference between 2 specific values, the difference between 2 distributions is used. The evidence lower bound function is used for this [23]. See [21] for a practical example of this.

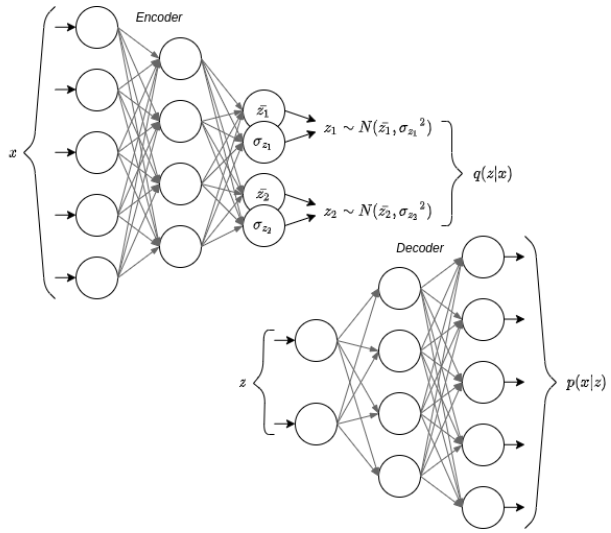

Fig. 2. Example of a variational autoencoder with 5 input features and 2 latent features.

The number of output notes in the encoder is double that of the decoder. The encoder nodes are paired and one is taken as the mean of a distribution, and the other as the standard deviation. This is referred to as 'reparameterization'. Each feature is constructed with the following form:

$$z = \mu + \sigma \odot \epsilon \tag{3}$$

Where ϵ is an error term and distributed as a standard normal distribution.

Also note that:

- The decoder is the same as a regular autoencoder in terms of structure, but the encoder now produces a distribution for each latent feature as an output, rather that a single value.
- When constructing latent features for training purposes only the mean output is used (or rather the error is set to 0).

E. Dimensionality reduction

Dimensional Reduction (DR) attempts to reduce the number of features of a dataset without reducing the performance of the classifiers trained on this set.

1) PCA: Principal component analysis is a technique that combines features in order to maximise the variance per feature. The data points are projected onto n-dimensional space. The vector direction of highest variance is then calculated. The direction of this vector then becomes the first new feature for the dataset. The n-dimensional space is then altered so all of the variance in the calculated vector is removed. This is then repeated for the new vector space, and the direction of highest variance is then calculated again and selected as the direction of the 2nd feature. This process is then repeated until there are no dimensions left. Note that the number of dimensions created is equal to the original number. This process does not remove or add any data, it only creates new features ordered by variance [4]. In order to reduce the number of features a threshold can be set, discarding all remaining features after a certain percentage of variance is obtained.

An alternative option to choosing an arbitrary threshold percentage is described in [17]. This paper describes a method using Bayesian model selection to determine the "true" dimensionality of the data. This is referred to as "MLE"

2) Latent Autoencoder Features: As mentioned previously the middle of the network of an autoencoder is purposely restricted to a smaller number of features than the input space. As a results of reducing this feature space the model is likely to produce latent features at this bottleneck which contain the highest amount of "information". These latent features can then be used for machine learning, reducing the feature space.

Unlike PCA which is only able to create features which are totally independent of each other, Autoencoders are able to produce features that have some amount of dependency on each other.

III. Microsoft Malware Classification Challenge

The Microsoft Malware Classification Challenge [16] is a collection of disassembled x86 malware files hosted on Kaggle[1]. This was originally constructed for WWW 2015 (international World Wide Web conference) [2].

For this report it is essential the models generated are evaluated. To do this, data used for training cannot also be used for testing. This ensures that over-fitting does not take place. To avoid this, 20% of the training data is set aside and only used for testing/generating metrics. From this, 8695 samples are contained in the training set and 2174 sample are in the testing set.

This report will be using only the assembly files. Each of these files contains a IDA [3] disassembled x86 binary malware file. Each line consists of the type of hex line (i.e. if it's data or code), the instruction offset, the function hex code, and then a list of arguments passed to that code. 2 hex digits are designated as this code, and hence there are 256 possible codes, although not all of these are used.

IV. Experimental Methodology

The goal of this experiment is to identify how altering the features of samples affects the performance of the machine

[1]https://www.kaggle.com/

[2]https://web.archive.org/web/20210117190951/http://www.www2015.it/welcome-letter/

[3]https://hex-rays.com/blog/ida-7-6-released/

learning classifiers. To do this the experiment can be broken down into several parts.

1. **Dataset Preprocessing** In this step the raw data is transformed into some form that can be used in the machine learning algorithms.
2. **Dimensionality Reduction** This step alters the preprocessed data and performs various types of dimension reduction techniques.
3. **Model Creation/Training** ML models are trained with the provided datasets.
4. **Model Evaluation** The various metrics are now calculated with the models and the testing data.

A. Dataset Preprocessing

The Microsoft Malware Dataset consists of a number of x86 assembly files (ASM) (see section III). Each of the lines in a ASM file has a number of hex digits. The first 2 begin the instruction to be executed on the set, and the rest are the arguments. This report will use a similar method as the winning entry of the Microsoft Malware Classification Challenge [8] to manipulate the data into a machine learning processable form: a unigram representation.

Equation (4) shows the unigram equation where f is instruction frequency.

$$\text{unigram}(f) = \begin{cases} 0, & \text{if} f = 0 \\ 1 + \log f, & \text{otherwise} \end{cases} \tag{4}$$

After this, min max scaling is applied.

B. Dimensionality Reduction

This stage applies the chosen dimensional reduction methods. There are specific parameters peculiar to each of these methods. By varying these parameters a better understanding of the effectiveness of each method can be achieved. Each of the following methods are applied to the raw datasets and then passed on.

- None
- PCA variance threshold (including MLE)
- VAE latent feature generation

C. Model Creation/Training

For each of the different versions of dimensionality reduction, several models are trained.

- Gaussian Naïve Bayesian Networks
- SVM with RBF kernel

D. Model Evaluation

At this stage each model is possessed with the appropriate test data into the metric generation. The following metrics are then generated:

- Accuracy
- F1
- Matthews Correlation Coefficient

V. Experimental Results

A. Dimensional Reduction Tuning

1) Variational Autoencoders: Fig. 3 shows how average performance across all models and latent feature dimensions increases as the model is trained. The results are inconclusive on if there is an ideal amount of training.

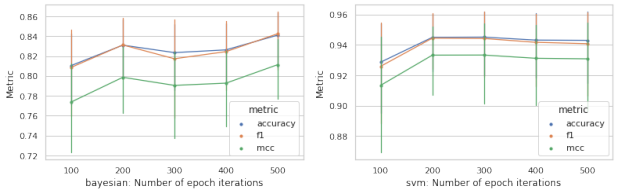

Fig. 3. Average performance across all combinations with an autoencoder against number of epochs trained

Fig. 4 shows how varying the number of latent feature dimensions effects the overall performance average across all models. Performance tends to increase as the number of latent dimensions increase. It is inconclusive if this experiment reached the ideal number of dimensions.

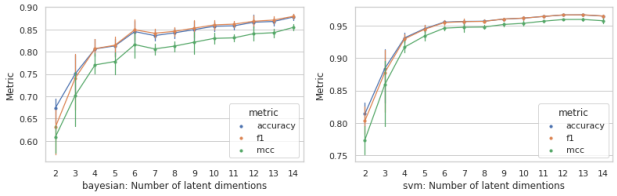

Fig. 4. Average performance across all combinations with an autoencoder against number of latent feature dimensions

2) PCA Variance Threshold: Fig. 5 displays the average effect on performance when including a PCA threshold before using ML classifiers. On average having a higher threshold seems to increase the performance.

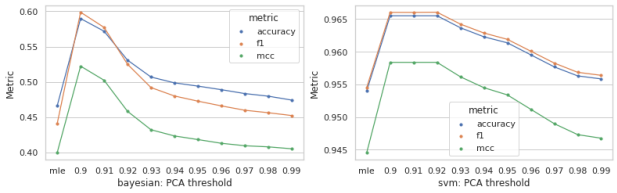

Fig. 5. Average performance across all combinations of ML classifier against type of PCA variance threshold

B. Classification Evaluations

For each model type, this subsection will cover the effect dimensionality reduction has on the performance of them.

1) Gaussian Naïve Bayesian: Fig. 6 shows the effect of the two different types of dimensional reduction on each of the performance metrics. The line in blue is the metric showing the performance of the model without any DR techniques applied.

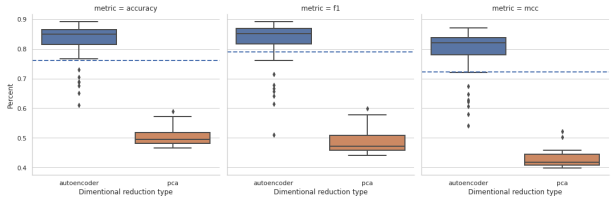

Fig. 6. Metric boxplot for all types of metric for Gaussian Naïve Bayes split by dimensional reduction method

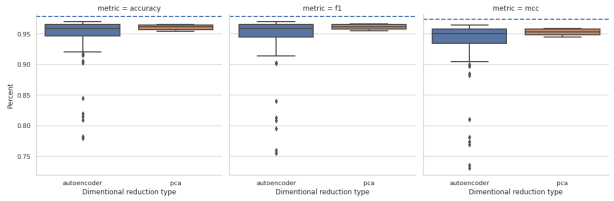

Fig. 7. Metric boxplot for all types of metric for SVM RBF kernel split by dimensional reduction method

This figure shows that on average the most effective method of increasing the performance of Bayesian networks is using a variational autoencoder. Table I displays the mean increase in performance for each metric and DR technique. Note that for each metric, VAEs increase the performance on average.

TABLE I
MEAN IMPROVEMENT FOR EACH DIMENSIONAL REDUCTION TECHNIQUE FROM BASE METRIC FOR GAUSSIAN NAÏVE BAYESIAN NETWORKS

DR type	VAE	PCA
Accuracy	0.066124	-0.252697
F1	0.034552	-0.297502
MCC	0.071725	-0.286058

The dimensional reduction technique that produced the highest metrics for Naïve Gaussian Bayes' classifier was a variational autoencoder with PCA trained for 300 epochs, with 12 latent feature dimensions. Table II shows the maximum performing dimensional reduction techniques for each type of DR.

TABLE II
MAXIMUM PERFORMANCE FOR EACH DIMENSIONAL REDUCTION TECHNIQUE FROM BASE METRIC FOR GAUSSIAN NAÏVE BAYESIAN NETWORKS

DR type	Params	Accuracy	F1	MCC
None	none	0.760350	0.790289	0.721719
VAE	300 epochs 12 features	0.891904	0.892821	0.870143
PCA	thresh 0.9	0.589696	0.598669	0.522298

2) Support Vector Machine with RBF Kernel: Fig. 7 displays each of the metrics for a SVM and the effect on them from altering the DR method. None of the DR methods increased the performance of the SVM. Table III show the maximum performing DR technique by MCC for each meta DR type. "none" has the highest performance with an accuracy of 0.978381 and a MCC of 0.973872. This is the highest for all of the techniques used in this dataset.

TABLE III
MAXIMUM IMPROVEMENT FOR EACH DIMENSIONAL REDUCTION TECHNIQUE FROM BASE METRIC FOR SVMS WITH RBF KERNEL

Meta DR type	DR	Accuracy	F1	MCC
None	none	0.978381	0.978586	0.973872
VAE	400 epochs 13 features	0.970101	0.970301	0.963812
PCA	threshold 0.92	0.965501	0.966044	0.958378

3) Performance: Table IV shows the top 5 performing models and the associated DR technique.

TABLE IV
TOP 5 PERFORMING MODELS

DR type	Model type	Accuracy	F1	MCC
VAE 400ep 12dim	svm rbf	0.968261	0.968434	0.961573
VAE 300ep 13dim	svm rbf	0.968721	0.968924	0.962157
VAE 300ep 14dim	svm rbf	0.968721	0.968856	0.962124
VAE 400ep 13dim	svm rbf	0.970101	0.970301	0.963812
none	svm rbf	0.978381	0.978586	0.973872

VI. CONCLUSION

When using suitable VAEs with Gaussian naïve Bayesian networks performance was increased above the original values. The maximum performing model of this type outperformed the base model by 13% to reach an accuracy of 0.896964. All DR methods were unable to improve the performance of SVM RBF models. However, a VAE generating 13 latent features from the original 256 feature space was able to get less than 1% less accuracy compared to the full 256 feature space when using a SVM RBF model.

A. Future Work

One of the benefits of using a reduced number of features for training is the increased training time. However, this measure was not recorded in this paper. It would be worthwhile to formally determine how VAEs reduce this.

This report only looked at how VAEs compared to PCA. It would be interesting to see how regular AEs compare to VAEs with the same dataset.

The depth of the VAEs was mentioned, but not explored. It would be worthwhile to look further into how varying the depth and structure of the VAE effects its performance.

This report does not look into to accuracy of detecting malicious software. This is because of the extra amount of effort it would have been to collect and then decompile benign software to compare against the malware dataset. Future work could look at this problem.

REFERENCES

[1] Muhammad Aurangzeb Ahmad, Carly Eckert, and Ankur Teredesai. "Interpretable machine learning in healthcare". In: *Proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics*. 2018, pp. 559–560.

[2] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[3] DeepMind. *AlphaGo - The story so far*. 2016. URL: https://deepmind.com/research/case-studies/alphago-the-story-so-far (visited on 08/05/2021).

[4] Karl Pearson F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[5] Mehadi Hassen, Marco M Carvalho, and Philip K Chan. “Malware classification using static analysis based features”. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2017, pp. 1–7.

[6] Henry J Kelley. “Gradient theory of optimal flight paths”. In: *Ars Journal* 30.10 (1960), pp. 947–954.

[7] Cheng-Yuan Liou et al. “Autoencoder for words”. In: *Neurocomputing* 139 (2014), pp. 84–96.

[8] Jiwei Liu. *First place approach in Microsoft Malware Classification Challenge (BIG 2015)*. 2015. URL: https://www.youtube.com/watch?v=VLQTRlLGz5Y (visited on 09/05/2021).

[9] Microsoft. *Adware:Win32/Lollipop*. 2013. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Adware:Win32/Lollipop (visited on 08/24/2021).

[10] Microsoft. *Malware Protection Center: Win32/Gatak*. 2013. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Gatak%5C&threatId=-2147289564 (visited on 08/24/2021).

[11] Microsoft. *Malware Protection Center: Win32/Obfuscator.ACY*. 2014. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/Obfuscator.ACY (visited on 08/24/2021).

[12] Microsoft. *Malware Protection Center: Win32/Ramnit*. 2011. URL: https://web.archive.org/web/20130325055658/http://www.microsoft.com/security/portal/threat/encyclopedia/Entry.aspx?Name=Win32/Ramnit (visited on 08/24/2021).

[13] Microsoft. *Malware Protection Center: Win32/Simda*. 2010. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Simda (visited on 08/24/2021).

[14] Microsoft. *Malware Protection Center: Win32/Tracur*. 2011. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Tracur (visited on 08/24/2021).

[15] Microsoft. *Malware Protection Center: Win32/Vundo*. 2009. URL: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Vundo (visited on 08/24/2021).

[16] Microsoft. *Microsoft Malware Classification Challenge*. 2018. URL: http://arxiv.org/abs/1802.10135 (visited on 08/11/2021).

[17] Thomas Minka. “Automatic choice of dimensionality for PCA”. In: *Advances in neural information processing systems* 13 (2000), pp. 598–604.

[18] Tom M Mitchell et al. “Machine learning”. In: (1997).

[19] Stefan Ortloff. *FAQ: Disabling the new Hlux/Kelihos Botnet*. 2012. URL: https://securelist.com/faq-disabling-the-new-hluxkelihos-botnet/32634/ (visited on 08/24/2021).

[20] Hemant Rathore et al. “Malware detection using machine learning and deep learning”. In: *International Conference on Big Data Analytics*. Springer. 2018, pp. 402–411.

[21] TensorFlow. *CVAE tutorial*. 2021. URL: https://www.tensorflow.org/tutorials/generative/cvae (visited on 08/16/2021).

[22] Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. “A primer on kernel methods”. In: *Kernel methods in computational biology* 47 (2004), pp. 35–70.

[23] Xitong Yang. “Understanding the variational lower bound”. In: *variational lower bound, ELBO, hard attention* (2017), pp. 1–4.

[24] Mahmood Yousefi-Azar et al. “Autoencoder-based feature learning for cyber security applications”. In: (2017), pp. 3854–3861.