# Classification of Malware by Using Structural Entropy on Convolutional Neural Networks

**Daniel Gibert**
Blueliv, Leap in Value
Barcelona, Spain

**Carles Mateu**
University of Lleida
Lleida, Spain

**Jordi Planes**
University of Lleida
Lleida, Spain

**Ramon Vicens**
Blueliv, Leap in Value
Barcelona, Spain

## Abstract

The number of malicious programs has grown both in number and in sophistication. Analyzing the malicious intent of vast amounts of data requires huge resources and thus, effective categorization of malware is required. In this paper, the content of a malicious program is represented as an entropy stream, where each value describes the amount of entropy of a small chunk of code in a specific location of the file. Wavelet transforms are then applied to this entropy signal to describe the variation in the entropic energy. Motivated by the visual similarity between streams of entropy of malicious software belonging to the same family, we propose a file agnostic deep learning approach for categorization of malware. Our method exploits the fact that most variants are generated by using common obfuscation techniques and that compression and encryption algorithms retain some properties present in the original code. This allows us to find discriminative patterns that almost all variants in a family share. Our method has been evaluated using the data provided by Microsoft for the BigData Innovators Gathering Anti-Malware Prediction Challenge, and achieved promising results in comparison with the State of the Art.

## Introduction

To evade detection, malware authors employ a variety of obfuscation techniques to hide malicious code inside executables. The most common are encryption and compression which are employed in most of the malware samples. In the information security industry, a common practice to detect the presence of encrypted or compressed segments hidden beneath portable executables is entropy analysis. In general, segments of code that have been compressed or encrypted tend to have higher entropy than native code (Lyda and Hamrock 2007).

In information theory, entropy (more specifically, Shannon's entropy) is the expected value of the information contained in each message. Generally speaking, the entropy of a bytes sequence refers to the amount of disorder(uncertainty) or its statistical variation. If occurrences of all values are the same, the entropy will be largest. On the contrary, if certain byte values occur with high probabilities, the entropy value will be smaller.

However, the use of simple entropy statistics may not be enough to detect sophisticated malware. Authors sometimes try to conceal encrypted or compressed code in a way that they pass through high entropy filters. For instance, they may add additional padding to reduce the mean file entropy. Anyway, native, encrypted or compressed segments and padding tend to differ markedly having distinct entropy levels. Thus, researchers (Sorokin 2011) started analyzing what is defined as the structural entropy of a file. In other words, each executable file is represented as a stream of entropy values, where each value describes the amount of entropy over a small chunk of code in a specific location of the file. Figure 1 displays the structural entropy of various malware executables belonging to two different families. It can be observed that the entropy streams extracted from malware samples belonging to the same family appear to be similar while distinct from those belonging to different families.

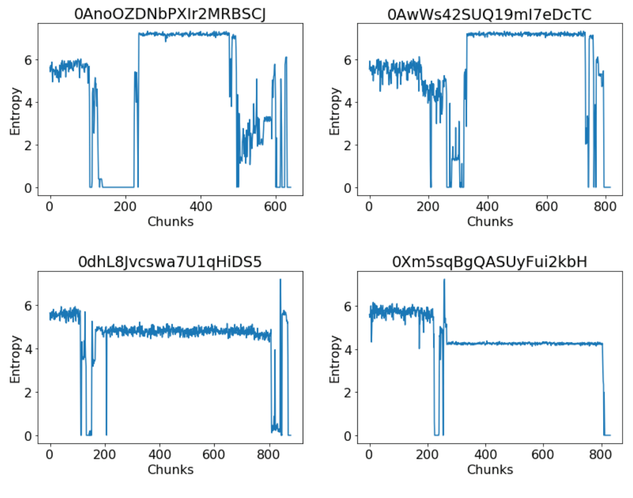


Figure 1: Entropy time series from malicious software. Samples from the first row belong to the Ramnit family, whereas samples from the second row belong to the Gatak family. Note the variation of the stream of entropy values between families.

By representing executable files as a stream of entropy values, the task of malware classification can be described as a time series classification problem (Fu 2011; Xing, Pei, and Keogh 2010; Bagnall et al. 2017).

The approaches most studied have been (1) Time domain distance based classifiers and (2) Shapelet based classifiers. On the one hand, Dynamic Time Warping (DTW) has been widely used as the preferred method to measure the similarity between two temporal sequences that may vary in length. (Ding et al. 2008), evaluated 8 different distance metrics on 38 time series datasets and found that DTW jointly with 1-Nearest Neighbor (NN) outperformed most of them. However, the NN is limited by its space complexity and its classification time complexity. On the other hand, time series shapelets were first used for classification in (Ye and Keogh 2009). Shapelets are discriminant subsequences of time series. The idea is that different classes of time series can be distinguished by their local variations instead of their global structure. Nevertheless, the computational complexity for the brute force search algorithm is polynomial, for best cases, $O(n^2m^3)$, where n is the number of time series in the dataset and m is the average length of each time series. (Grabocka et al. 2014) introduced a more efficient alternative to learn shapelets. In their method, instead of searching among possible candidates from time series segments, they proposed a method to directly learn optimal shapelets without exploring all possible candidates. Their approach, starts by guessing a set of initial shapelets. Then, it iteratively learns the shapelets by minimizing an error function.

Inspired by the recent advances in the deep learning field and the work of (Grabocka et al. 2014), in this paper we propose a file agnostic end-to-end deep nonlinear feature learning and classification based method for categorization of malicious software based on its structural entropy. Our approach has been evaluated using the dataset provided by Microsoft for the Big Data Innovators Gathering (BIG 2015) Anti-Malware Prediction Challenge.

The rest of the paper is organized as follows. Firstly, we introduce our deep learning approach for malware classification. Then, the results of the performance evaluation for our method are presented. And lastly, the paper concludes with our remarks.

## Classification of Software's Structural Entropy via Deep Learning

As can be observed in Figure 1, there exists empirical evidence that entropy time series from a given family are visually similar and distinct from those belonging to a different family. This is perhaps the result of reusing the code to create new malware variants. Consequently, this visual similarity motivated us to apply convolutional neural networks for time series classification to automatically learn good feature representations from both time and frequency domains jointly. To do this, a given executable is transformed into a recognizable input by applying the following two-step process:

1. Structural entropy calculation. The entropy of an executable file is computed by splitting its hexadecimal representation (00h–FFh) into non-overlapping chunks of fixed size. In the literature, a common value is 256 bytes. For each chunk of code, the entropy is then computed using Shannon's formula defined as:

$$H(X) = -\sum_{i=1}^{n} p(i) \cdot log_b p(i)$$

   where $H(X)$ is the measured entropy value of a discrete random variable $X$ with values $x_1, \ldots, x_j$, $j$ is the number of values in X, $p(i)$ refers to the probability of appearances of the byte value $i$ in $X$ and $n$ is equal to $255$, i.e. byte code values are in the range of [0, 255].

2. Discrete Wavelet Transform. The single-level discrete wavelet transform is applied to the entropy time series in order to compress the signal and reduce the noise. The original vector of length $N$ is transformed into two vectors of length $N/2$, named the approximation coefficients and the detail coefficients. In this work, the Haar wavelet transform (Haar 1910) has been used, instead of any other transforms such as Daubechies or Morlet, due to its efficiency in computation.

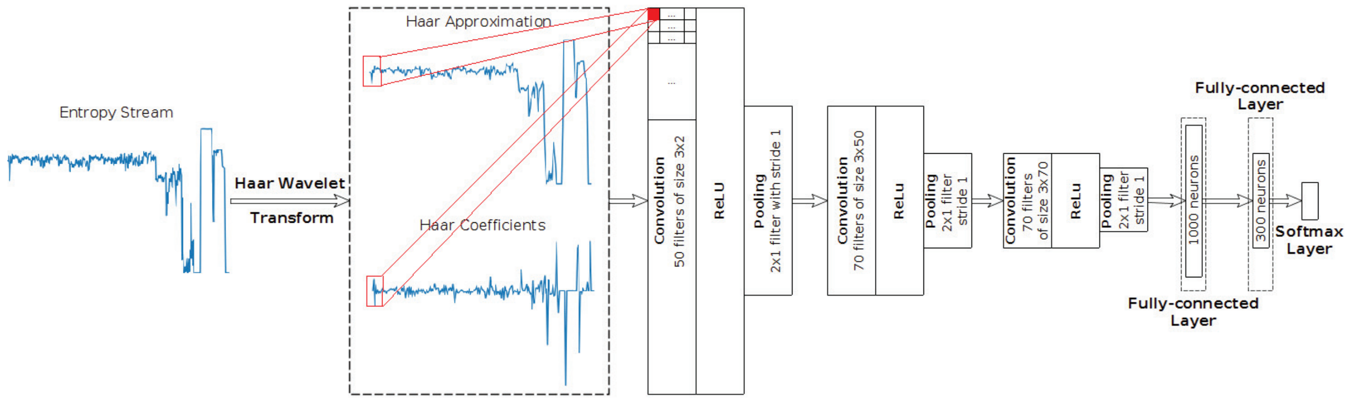
### Network Architecture

The overall architecture of the network is illustrated in Figure 2. The input is a multivariate time series $M$, defined as $M = \{m_1, m_2\}$, where each element $m_i$ is a univariate time series. A univariate time series is a sequence of data points measured at successive points in time. It is denoted as $T = \{t_1, t_2, \ldots, t_n\}$, where n is the length of $T$. At any time stamp $t$, $m_t = \{m_{1,t}, m_{2,t}\}$ where $m_{1,t}$ and $m_{2,t}$ are the values of the Haar approximation and Haar coefficient values at time stamp $t$.

The core of the convolutional neural network consists of three convolutional layers plus two fully-connected layers. The convolutional layers perform feature learning on both univariate series jointly. Then a normal feed-forward network is concatenated at the end of feature learning to perform classification. Specifically, the input is fed into a 3-stage feature extractor which learns hierarchical features through convolution, activation and pooling layers.

**Convolution** is an operation that takes an input signal and a feature map and produces one output signal. A convolution operation involves a filter $w_k \in \mathbb{R}^{ij}$ where $i \leq w$ and $j \leq h$ and $i$ and $j$ are the width and the height of the an input 2D signal. The output of convolving the k-th kernel of the convolutional layer $l$, $w_{l,k}$ over a 2D signal is defined as $c = w_{l,k} \times x + b_{l,k}$, where $b_{l,k}$ is the bias of the k-th kernel in layer $l$. The kernel slides over each value of the input signal, multiplies the corresponding entries of the input signal and the kernel and adds them up. The convolutional layers are composed of 50, 70 and 70 feature maps with 3 by 2, 3 by 50 and 3 by 70 receptive fields for the first, second and third convolutional layers, respectively.

**Activation function** introduces non-linearity into the network. It takes a single value $x$ and performs a mathematical operation on it. In particular, we adopt the ReLU function $\max(0, x)$ in all activation layers.



Entropy Stream
Haar Wavelet Transform
Haar Approximation
Haar Coefficients
Convolution 50 filters of size 3x2
ReLU
Pooling 2x1 filter with stride 1
Convolution 70 filters of size 3x50
ReLu
Pooling 2x1 filter stride 1
Convolution 70 filters of size 3x70
ReLu
Pooling 2x1 filter stride 1
1000 neurons
300 neurons
Fully-connected Layer
Fully-connected Layer
Softmax Layer

Figure 2: Convolutional network architecture for malware classification. It is composed by 3 convolutional layers followed by 2 fully-connected layers. The input of the network are two univariate time series, the average and the details vector generated by transforming an entropy stream with Haar wavelets. The output of the network is the predicted class of the malware sample.

**Pooling** is a function that reduces the spatial size of an input signal. It helps to reduce the amount of parameters and computation in the network as well as to control overfitting. We applied max-pooling with filters of size $2 \times 1$ with stride 1, which reduces the input signal by half.

Convolutional layers can be seen as detection filters for the presence of specific features or patterns present in the data. The first layers detect low-level features whereas the last ones detect increasingly complex features. At the end of the extractor, the feature maps are flattened and combined as the input of the subsequent feed-forward layers plus a softmax layer for classification. Particularly, the number of units in the feed-forward layers is equal to 1000 and 300 for the first and the second layer, respectively. To prevent overfitting, dropout (Srivastava et al. 2014) was used and, to improve the stability of our model, an ensemble algorithm was used, named bootstrap aggregating (Breiman 1996).

## Resilience to Obfuscation Techniques

By nature, the features learned by the convolutional neural networks are invariant to translation. That is, CNNs are able to detect patterns which may be displaced in space through the convolution and max-pooling operations. The convolution operation provides equivariance to translation. In our domain this means that signal patterns may be recognized at any temporal space. Additionally, the max-pooling operation returns the largest value in its receptive field. Thus, the location of this value, if it is still within the receptive field, do not alters the output of the pooling operation. Thus, both operations together provide invariance to translation. This property is really helpful against detecting the changes produced by the following obfuscation techniques:

**Dead-code insertion.** This technique adds ineffective instructions, such as the NOP instruction, to the program to change its appearance while maintaining the same functionality. By adding NOP instructions, the average entropy of the executable will decrease, but the entropy of the adjacent chunks containing the actual code will differ greatly from the chunks containing NOP instructions as it can be observed in Figure 3.

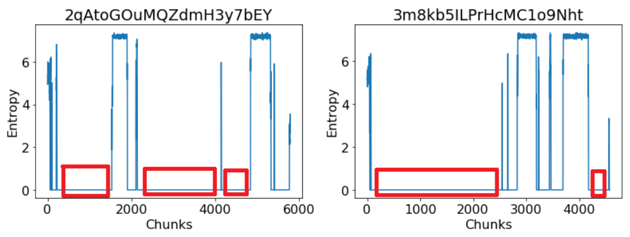


Figure 3: Two samples belonging to the Simda family whose code has been modified by the dead-code insertion technique (chunks highlighted in red).

**Code transposition.** This technique reorders the sequence of the instructions without changing the behavior of the program. For instance, the instructions

| | | |
|---|---|---|
| 1: ADD R1 R2 | | 1: ADD R3 R4 |
| 2: ADD R3 R4 | can be replaced by | 2: ADD R1 R2 |

If the sequence of instructions is located inside the same chunk of code as it was previously to the reordering, the entropy of the chunk will still be the same.

**Subroutine reordering.** By applying this technique, the order of the subroutines in the original code is changed randomly. Cf. Figure 4. Being invariant to translation means that the location of the subroutines will not affect the outcome of the classifier, because the network is able to find the patterns independent of their location.

**Encryption and packing** are the most common methods employed to hide malicious code into executables. These methods transform a series of original bytes into a series of random-looking data bytes. In the information security industry, to detect the presence of encrypted or com-

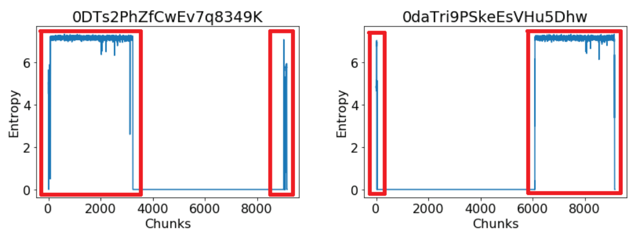


Figure 4: Two samples belonging to the Kelihos_ver3 family. It can be observed that the actual code of the program (highlighted in red) has been reallocated from the start to the end of the file.

pressed segments hidden beneath the executable, code entropy analysis is commonly performed. Typically, files with high entropy are relatively likely to have encrypted or compressed sections inside them (Lyda and Hamrock 2007). Thus, by representing an executable as a stream of entropy values, the presence of encrypted or compressed segments hidden within portable executable files can be detected. Figure 5 shows two samples belonging to the Obfuscator.ACY family. It can be observed that the entropy of different segments varies along the files. Therefore, the local patterns learned by the CNN should be able to detect these changes in the entropy values between encrypted or compressed chunks and the chunks containing the rest of the code.

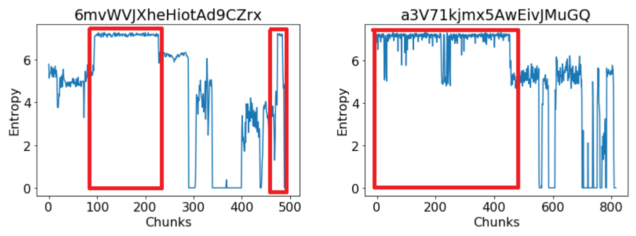


Figure 5: Two samples belonging to the Obfuscator.ACY family. The red box highlights the possible encrypted or compressed segments within the files.

## Evaluation

The data used to evaluate our deep learning approach were provided by Microsoft for the BigData Innovators Gathering (BIG 2015) Anti-Malware Prediction Challenge. The dataset is composed of 21741 samples, 10868 for training and 10873 for testing, grouped into 9 different malware families (Cf. Table 1).

### Experimental Setup

To estimate the generalization performance of our approach we used K-fold cross validation, where $K = 10$. Additionally, the best model was selected according to the F1 score. That is because classification accuracy alone can be misleading. Sometimes, it may be desirable to select a model with a lower accuracy but with greater predictive power (a.k.a. accuracy paradox). This is true in a problem like ours where there is a large class imbalance, where a model can predict the value of the majority class for all predictions and achieve high classification accuracy while misclassifying samples from the minority or critical classes. In particular, since the task we are trying to solve is a multi-class classification problem we used an adaptation of the score called macro-averaged F1 score, defined as the average of the individual F1 scores obtained for each class. Macroaveraging gives equal weight to each class. Thus, large classes will not dominate small classes.

Table 1: BIG 2015 dataset statistics.

| Family | Class ID | #samples | Average size (bytes) | Average length (256 bytes) |
|---|---|---|---|---|
| Ramnit | 1 | 1541 | 1482169.56 | 1597.17 |
| Lollipop | 2 | 2478 | 5829531.16 | 6281.75 |
| Kelihos_ver3 | 3 | 2942 | 8982629.66 | 9679.56 |
| Vundo | 4 | 475 | 1120945.27 | 1207.90 |
| Simda | 5 | 42 | 4552326.09 | 4905.52 |
| Tracur | 6 | 751 | 1801152.85 | 1940.90 |
| Kelihos_ver1 | 7 | 398 | 5051900.48 | 5443.85 |
| Obfuscator.ACY | 8 | 1228 | 827118.28 | 891.29 |
| Gatak | 9 | 1013 | 2555072.57 | 2753.31 |

The experimentation has been divided into three phases. In the first phase, it has been studied how the chunk size influences the output of the network. In the second phase, it has been analyzed how the Haar approximation of the initial entropic signal impacts our classifier. In the third phase we compared our best model with state of the art methods in the literature.

**Chunk Size Comparison** The size of the malicious programs varies greatly between families. Thus, their corresponding time series differ in length from one family to another, independently of the chunk size, cf. Table 1.

To study which chunk size provides a better trade-off between accuracy and performance, we evaluated three network models, by using as training data the time series obtained after splitting an executable file into chunks of size 256, 1024 and 4096 bytes.

The network architecture is the same as the one described in Figure 2, with the exception of the input layer. In this case, networks are fed with univariate time series containing the stream of entropy values representing an executable file. The percentage of correctly predicted labels over all predictions (accuracy) is 0.9626, 0.9720 and 0.9708 for 256, 1024 and 4096 bytes, respectively. On the contrary, the highest F1 score was achieved by the model trained on the time series obtained after splitting the malicious programs into chunks of 4096 bytes, which is 0.9314. In Table 2 and Table 3, it can be observed that both models failed to predict most of the samples belonging to the Simda family. Additionally, even though the overall accuracy is higher, the number of misclassified samples (54.76%) belonging to the Ramnit family has greatly punished the model trained on the time series obtained after splitting the files in chunks of size 1024.

**Haar Wavelet Transform.** In the second phase, Haar Wavelet Transform was used to decompose the initial sig-

Table 2: 10-fold cross validation confusion matrix obtained by training the CNN with the time series obtained by splitting a program into chunks of 1024 bytes.

| Family | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1499 | 8 | 1 | 3 | 0 | 8 | 2 | 15 | 5 |
| 2 | 14 | 2447 | 0 | 1 | 0 | 5 | 0 | 2 | 9 |
| 3 | 0 | 0 | 2940 | 0 | 0 | 2 | 0 | 0 | 0 |
| 4 | 8 | 3 | 0 | 455 | 0 | 4 | 0 | 4 | 1 |
| 5 | 19 | 2 | 0 | 3 | 14 | 0 | 1 | 2 | 1 |
| 6 | 16 | 5 | 0 | 4 | 0 | 715 | 4 | 6 | 1 |
| 7 | 5 | 2 | 0 | 0 | 0 | 3 | 388 | 0 | 0 |
| 8 | 59 | 9 | 4 | 9 | 1 | 23 | 2 | 1117 | 4 |
| 9 | 6 | 6 | 0 | 3 | 0 | 6 | 0 | 3 | 989 |
| Accuracy | 10564 / 10868 = 0.9720 | | | | | | | | |
| F1 score | 0.9127 | | | | | | | | |

Table 3: 10-fold cross validation confusion matrix obtained by training the CNN with the time series obtained by splitting a program into chunks of 4096 bytes.

| Family | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1490 | 9 | 1 | 9 | 1 | 15 | 1 | 10 | 5 |
| 2 | 12 | 2445 | 0 | 2 | 0 | 5 | 1 | 2 | 11 |
| 3 | 0 | 0 | 2940 | 0 | 0 | 2 | 0 | 0 | 0 |
| 4 | 8 | 3 | 0 | 455 | 0 | 4 | 0 | 5 | 0 |
| 5 | 12 | 0 | 1 | 4 | 23 | 1 | 0 | 1 | 0 |
| 6 | 15 | 9 | 0 | 13 | 0 | 702 | 4 | 4 | 4 |
| 7 | 2 | 0 | 0 | 0 | 0 | 1 | 395 | 0 | 0 |
| 8 | 57 | 3 | 3 | 8 | 1 | 40 | 1 | 1114 | 1 |
| 9 | 5 | 6 | 0 | 5 | 0 | 9 | 0 | 1 | 987 |
| Accuracy | 10551/ 10868 = 0,9708 | | | | | | | | |
| F1 score | 0.9314 | | | | | | | | |

nals into two sequences describing an approximation of the original signal, plus a set of details (coefficients) representing the localized changes. Then, two models were trained. On the one hand, we trained a model using as input the resulting signal approximations (hereinafter, Haar approximation model). On the other hand, the second model was fed with the approximation of the original signal plus the details sequence (hereinafter, multiresolution model). The F1 score increased slightly from 0.9314 to 0.9621 and 0.9636 for the Haar approximation and multiresolution models, respectively. Observe in Table 4 that the number of samples misclassified belonging to the Simda family has been reduced by more than half (from 19 to 8). In addition, even that the number of incorrectly classified samples belonging to the Obfuscator.ACY family has been reduced from 114 to 70, it is still a major source of error. That's because the Obfuscator.ACY family comprises malware that has been obfuscated by using compression and encryption techniques, among others. In consequence, the malware that lies underneath this obfuscation can have any purpose and in some cases, its structural entropy is very similar to those of samples from the rest of families.

**State-of-the-Art Comparison.** Many algorithms have been developed for the task of time series classification. From among them, the nearest neighbor (particularly 1-NN) in combination with the Dynamic Time Warping (DTW) similarity metric achieves the state of the art performance. If applied to the training data provided by Microsoft, the resulting 10-fold cross validation accuracy and F1 score

Table 4: 10-fold cross validation confusion matrix obtained by training the CNN with both the approximation and the coefficient signals of the entropy time series after applying the Haar Wavelet Transform.

| Family | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1519 | 5 | 1 | 3 | 0 | 4 | 2 | 7 | 0 |
| 2 | 6 | 2457 | 0 | 2 | 0 | 1 | 0 | 5 | 7 |
| 3 | 0 | 0 | 2941 | 0 | 0 | 1 | 0 | 0 | 0 |
| 4 | 3 | 1 | 0 | 463 | 0 | 6 | 0 | 2 | 0 |
| 5 | 1 | 0 | 0 | 3 | 34 | 0 | 1 | 3 | 0 |
| 6 | 5 | 5 | 0 | 15 | 0 | 720 | 0 | 4 | 2 |
| 7 | 0 | 0 | 0 | 0 | 0 | 2 | 395 | 1 | 0 |
| 8 | 29 | 4 | 2 | 8 | 3 | 21 | 1 | 1158 | 2 |
| 9 | 2 | 6 | 0 | 4 | 0 | 4 | 0 | 3 | 994 |
| Accuracy | 10681 / 10868 = 0.9828 | | | | | | | | |
| F1 score | 0.9636 | | | | | | | | |

achieved by the 1-NN algorithm are higher than ours. On the contrary, if both approaches are evaluated on the test set, our model substantially outperforms the 1-NN algorithm. Table 5 presents an overview of the results obtained by our deep learning approach and the nearest neighbor algorithm in both the training and the test data. The superior predictive power of convolutional networks (CNN) can be observed with respect to the nearest neighbor algorithm. In particular, the CNN reduced the logarithmic loss to 0.124431 while the logloss obtained by the nearest neighbor is equal to 0.367724. Furthermore, if bagging is employed, by averaging the predictions of the 10 models the test logarithmic loss is reduced to 0.075081. Moreover, a great advantage of the CNN over the nearest neighbor is that its prediction time always remains constant (approximately 0.02 seconds per sample). On the contrary, the prediction time of the nearest neighbor grows linearly as the data set grows larger. That is because to predict the label of an unknown sample, it has to be compared with all individuals in the dataset.

Table 5: Comparison of the CNN with the nearest neighbor algorithm.

| Method | 10-Fold accuracy | F1 score | Test logloss |
|---|---|---|---|
| DTW + K-NN | 0.9894 | 0.9813 | 0.367724 |
| Haar Transform + DTW + K-NN | 0.9870 | 0.9710 | 0.458191 |
| CNN Entropy | 0.9708 | 0.9314 | 0.134624 |
| CNN Multiresolution | 0.9828 | 0.9636 | 0.124431 |
| Bagging CNN Multiresolution | - | - | 0.075081 |

**Transfer Learning.** Once trained, the model could be used to generate domain-specific features based on the structural entropy of a malicious program. Instead of classifying executable files into families, we could extract the features learned by the network in the last feed-forward layer. Then, these features could be integrated into a bigger classifier based on different subsets of features. Next, we prove the suitability of this approach by transferring the features learned into an XGBoost classifier and comparing with the results obtained by (Ahmadi et al. 2015). In their work, they extracted a wide range of hand-crafted features from the malicious executables. The subset of features that achieved the best results individually were:

**Entropy (ENT).** Statistical measures from the structural entropy of malicious programs such as quantiles, percentiles, mean, etc.

**Opcodes (OPC).** Use of a subset of 93 operation codes based on their commonness or on their frequent use in malicious applications.

**Application Programming Interfaces (APIs).** Frequency of use of 794 APIs.

**Section (SEC).** Characteristics from sections such as the total number of lines in each section, the proportion of each section in comparison to the whole file, etc.

**Registers (REG).** Frequency of use of the registers.

**Miscellaneous (MISC).** Frequency of 95 manually chosen words from the disassembled code.

Table 6 shows the results obtained in the training data after performing 5-fold cross validation. It can be observed that the model trained on the entropy-based features extracted by the CNN achieved better accuracy and lower logloss than most hand-crafted features, and in particular, the opposite manually extracted entropy-based features.

Table 6: List of feature categories and their evaluation with XGBoost

| | | 5-Fold Cross Validation | |
|---|---|---|---|
| Feature Category | #Features | Accuracy | Logloss |
| ENT | 203 | 0.9862 | 0.0505 |
| OPC | 93 | 0.9907 | 0.0405 |
| API | 796 | 0.9843 | 0.0610 |
| SEC | 25 | 0.9899 | 0.0420 |
| REG | 26 | 0.9833 | 0.0695 |
| MISC | 95 | 0.9917 | 0.0306 |
| CNN Multiresolution | 300 | 0.9896 | 0.0369 |

## Conclusions and Future Work

In this work, we presented a file agnostic deep learning system for categorizing malware. As far as we know, it is the first approach that applies deep learning to find discriminant patterns from executable files represented as streams of entropy values. The proposed solution has a number of advantages that help to detect malicious programs efficiently in an enterprise environment. First, it is file agnostic and is based solely on the structural entropy of a file. Second, the nature of the features learned by convolutional neural networks demonstrated robustness against the most common obfuscation techniques. Third, neural networks scale well with the data. In general, the more data provided the better the quality of the model. Fourth, once the entropy values are computed, the prediction time is minimal. The approach has been compared with state-of-the-art methods in the literature for time series classification and demonstrated the superior predictive power of our deep learning approach.

A future direction of research is to study how different mother wavelets affect the model. Applying any other mother wavelet, such as the Daubechies or the Morlet, might lead to higher accuracy. Additionally, the hierarchical entropy-based features learned by the convolutional neural networks could be useful as a subset of features for machine learning models which attempt to identify malware based on distinct types of file features.

## Acknowledgements

We would like to thank the Blueliv Labs team, specially Daniel Solís, and Àngel Puigventós for their support and the feedback provided during the development of this work. This work has been partially funded by the Spanish MICINN Projects TIN2014-53234-C2-2-R, TIN2015-71799-C2-2-P and ENE2015-64117-C5-1-R, and by AGAUR DI-2016-091. This research article has received a grant for its linguistic revision from the Language Institute of the University of Lleida (2017 call).

## References

Ahmadi, M.; Giacinto, G.; Ulyanov, D.; Semenov, S.; and Trofimov, M. 2015. Novel feature extraction, selection and fusion for effective malware family classification. *CoRR* abs/1511.04317.

Bagnall, A.; Lines, J.; Bostrom, A.; Large, J.; and Keogh, E. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 31(3):606–660.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Ding, H.; Trajcevski, G.; Scheuermann, P.; Wang, X.; and Keogh, E. 2008. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.* 1(2):1542–1552.

Fu, T.-C. 2011. A review on time series data mining. *Engineering Applications of Artificial Intelligence* 24(1):164 – 181.

Grabocka, J.; Schilling, N.; Wistuba, M.; and Schmidt-Thieme, L. 2014. Learning time-series shapelets. In *KDD'14*, 392–401. ACM.

Haar, A. 1910. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen* 69(3):331–371.

Lyda, R., and Hamrock, J. 2007. Using entropy analysis to find encrypted and packed malware. *IEEE Security and Privacy* 5(2):40–45.

Sorokin, I. 2011. Comparing files using structural entropy. *Journal in Computer Virology* 7(4):259.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15:1929–1958.

Xing, Z.; Pei, J.; and Keogh, E. 2010. A brief survey on sequence classification. *SIGKDD Expl. New.* 12(1):40–48.

Ye, L., and Keogh, E. 2009. Time series shapelets: A new primitive for data mining. In *KDD'09*, 947–956. ACM.