International Journal of Computer Science, Communication & Information Technology (CSCIT)
Vol.8 pp. 17-22

A First Approach to Malware Detection using Residual Networks

Hoda El Merabet

Faculty of Science, Abdelmalek Essaadi University
B.P. 2117 Quartier M'hanech II, Av. Palestine, Tetouan, Morocco
helmerabet@uae.ac.ma

***Abstract*— The internet is omnipresent in our daily lives. As a result, potential attacks and compromises become possible to illegally access our data. With a focus on keeping our machines safe, research on malware detection field is strongly active, especially with the help of machine learning. Machine learning has been evolved into this field in the last two decades. Support Vector Machines SVMs, random forests, logistic regression and deep artificial neural networks ANNs were employed in different combinations in order to get high performance results, and thus to correctly identify malware from benign files. Since Windows machines are the most popular in the area of desktop and laptop computers, we are building our study upon the portable executable (PE) format; the file format for executables of Windows operating systems. Accordingly, features were extracted from the PE header fields to be used as inputs to our model. Several researches considered dealing with executable files as images a good way to benefit from the advantages of Convolutional Neural Networks CNNs to detect malware. In our study, we are using a relatively new class of CNNs, namely Residual Networks ResNets. These are CNNs based on shortcut connections for feature reuse. The extracted PE features were converted to byte values, afterwards fed to the ResNet model as greyscale images. The obtained results were reasonably satisfying.**

***Keywords*—Malware Classification; Malware Detection; Machine Learning; Artificial Neural Networks; Deep Learning; Convolutional Neural Networks; Residual Networks**

I. INTRODUCTION

Recognition and identification of malware is an active area because of the huge number of malicious software daily registered. Every day, the AV-TEST Institute registers over 350,000 new malicious programs (malware) and potentially unwanted applications (PUA) [1]. There are three approaches to identify malicious from legitimate files. The first one is the static approach. It relies on the extraction of signatures, which are values present in the files, or values that should be calculated using a hash function for instance, like the MD5 message-digest algorithm. Even though this approach excels at blocking known malware, it would never detect new ones as they use novel signatures. The second approach is the dynamic approach. This one is based on the execution of the files on a virtual environment or a sandbox to be able to detect the suspicious activities and visualize the behaviour of each file. This approach requires a long period of scanning, and a high consumption of resources. The third approach is the heuristic approach based on machine learning models, which has proven its success the last two decades. It can rely on static features, on dynamic features, or on both of them.

There are a lot of factors that influence the success of a model, particularly the types of extracted features, the feature selection techniques, and the used machine learning algorithms [2]. In this paper, we are using PE features as inputs to our model and a residual neural network model as classification technique. Residual networks are a new form of deep learning models introduced the last four years, initially for image recognition.

Deep learning models suffer from the well-known vanishing gradient problem. This problem gets bigger when using a large number of layers. The gradients become too small hence the training doesn't happen efficiently. The apparition of residual networks ResNets for image recognition [8, 13] was innovative for providing a new solution to this problem. This kind of networks was thereafter used in Natural Language Processing NLP [14], in morphological reinflection [15], in language identification [16], in semantic tagging and Part-of-Speech POS tagging [17], and in single image super resolution [18]. Inspired by the great results obtained by these studies, ResNets were used in our study for the first time for malware detection, to the best of our knowledge.

II. RELATED WORK ON MALWARE DETECTION

The ability of machine learning models to learn from training data, and subsequently to generalize to previously unseen data is a major reason to have an increasingly number of researches interested in this area. Each research relies on different features and different classification techniques.

In [3], each sample from the training dataset was executed within a virtual environment to capture dynamic behaviour of it. The model was then built upon 300 vectors of nine collected machine activity metrics. These metrics are: CPU user use, CPU system use, RAM use, SWAP use, received packets, received bytes, sent packets, sent bytes, and the number of running processes. These metrics were afterwards transformed to 300 vectors of x-y coordinates using Self Organizing Feature Maps SOFMs. The dataset was constituted of 1188 PE files; 594 malicious and 594 benign. Half the files were used for training and the other half for testing. After registering the machine activity metrics each second in a 5-minutes time window, the researchers ended by collecting a dataset of 345,000 observations. For comparison, two machine learning classifiers were used in two separate experiments. Random forest was used in a first experiment, and in a second

ISSN 1737-930X

one, logistic regression was used. The best accuracy, namely 86.70%, was obtained using logistic regression.

As for [4], API calls and operating system resource instances were used to identify API call graphs. These graphs were used as input features for the designed deep learning model. Multiple layers of Sparse Auto Encoders SAEs were employed to reduce the size of the original extracted features, followed by a Decision Tree classifier. The training dataset was constituted of 1760 samples, where 880 files were malicious and 880 were benign files. 10-fold cross-validation technique was used for training and testing. The achieved detection precision was 98.6%. In this paper, we are comparing the results of the different researches according to the accuracy metric. However, only the precision rate was published for [4].

The features used in [9] were n-gram system call sequences. A combination of two models was used, giving two pairs of probability values for each sample. Thereafter, these probabilities were compared to a threshold to get the final prediction result. The first model was the Long-Short-Term Memory LSTM. Here, the information gain technique was employed as feature selection technique, in order to remove redundant sub-sequences. Afterward, the LSTM model was fed, followed by a Max-Pooling layer then by a logistic regression classifier. The second model was the Random Forest model. At this point, the input features were API statistical features, obtained from the association of two API calls, based on the comparison of the two API call sequences hashes. Initially, the experiments were conducted using each model separately. The results got by combining both models were better. The obtained accuracy using the combination of both models was 95.7%.

Since CNNs demonstrated high performances in image recognition, numerous malware detection researches took advantage of this type of algorithms. The work of [5] relied on a dynamic approach. The samples were executed in virtual machines, then process performance metrics were extracted. Each sample was represented as an image, the rows contained the executed processes, and the columns contained the 28 extracted features per process like the process status, the CPU usage percent, the number of read requests, the number of write requests, the number of read bytes, the number of written bytes, etc. CNNs were employed to classify the samples. An accuracy of 90% was obtained on the test dataset.

In [6], the raw bytes of each executable file were converted using an embedding layer, in order to be used as input features. Through this embedding layer, each byte was mapped to a fixed length feature vector. They avoided to use raw byte values, as they can lead to the interpretation that certain byte values are intrinsically closer to each other than other byte values, which is absolutely false as byte value meaning depends on the context. The model was constructed from CNNs and Recurrent Neural Networks RNNs followed by a fully connected layer. A dataset of 400,000 files split evenly between benign and malicious files was used for training. And a distinct testing set of 77,349 files was used for the final test phase. An accuracy of 90.90% was obtained.

III. Deep Residual Learning

A. *Strengthening Vision with CNNs*

A convolution operation is an element wise matrix multiplication operation. CNNs are based on convolutions between the input images and a number of filters. These filters are not predefined, they are learned during the training process. After this operation, new images called feature maps are obtained. Each convolution, or the application of each filter in other words, will change the image in such a way that particular features get emphasized. Next to convolutions, pooling can also be used within CNNs. It is a way of compressing the convolved images, in order to reduce the number of parameters and computations in the network. Because of the ability of a CNN to learn the existence of a feature regardless of its position, and since the different parts of a PE executable file can be placed anywhere within the file, as observed for instance by Barker et al. in [6], CNNs have been introduced to malware detection field in plentiful researches.

B. *Problem with Deep Neural Networks*

The intuition behind deep learning is that adding more layers makes the model learning more complex features. Accordingly, it becomes able to make more accurate decisions. However, [8] showed that continuing to go deeper affects negatively the performance of a traditional CNN model. They conducted experiments on the CIFAR-10 dataset [10] to classify images. Fig. 1, represented by the experiments of [8] depicts the performance decreasing going from a model with 20 layers to a model with 56 layers.

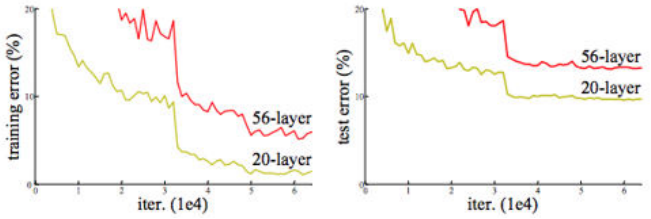

Fig. 1 Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer plain networks. The deeper network has higher training error and thus higher test error [8]

This problem of performance degradation can be returned to a very known problem, namely the vanishing gradient problem. The gradients become negligible when they pass from the end to the beginning of the model through the high number of layers. Furthermore, the problem can be caused by the loss of input information when it travels from the beginning of the network to its end. In order to lighten this problem while training very deep neural networks, the authors have introduced residual blocks. These blocks are based on feature reuse.

C. *Residual Blocks*

A residual block is a block of layers that adds the input to the output, before continuing to feed a new block of layers. It can be constituted of numerous convolutional layers, generally two or three but can be more.

ISSN 1737-930X

Let us consider x the input to a few stacked layers of a deep machine learning model, and F(x) the output of it. If we assume that x and F(x) have the same dimensions, then a residual block can be defined by equation (1):

$$y = F(x) + x \quad (1)$$

Fig. 2 shows the building block of residual learning as given by [8].

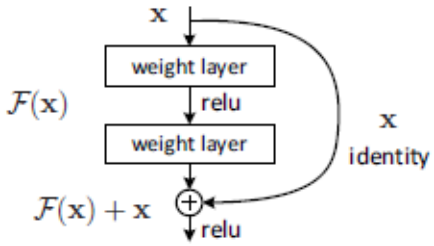

Fig. 2 A residual block [8]

The operation F + x is performed by a shortcut connection and element-wise addition [8]. It introduces neither extra parameters nor calculation complication.

If x and F(x) don't have the same dimensions, the identity mapping x can be multiplied by a linear projection just to match dimensions, as represented by equation (2):

$$y = F(x, \{W_i\}) + W_s x \quad (2)$$

Where: W_i are the weights to be learned in the stacked layers of the residual block

W_s is the matrix used to match dimensions

Residual networks are constituted by stacking consecutive residual blocks together. As we just saw above, they are based on shortcut connections to propagate and reuse information over the layers of the model. This helps in mitigating the effect of the vanishing gradient problem, allowing us to build deeper networks with higher performances.

D. Residual Networks in Image Recognition

In [8], ImageNet 2012 classification dataset [11] was used to test the efficiency of a residual network architecture. It consists of 1000 classes of images. The results are depicted in Fig. 3. In the left, training and test errors are drawn for two plain networks, without residual blocks. The first one is constituted of 18 layers and the second one of 34 layers. In the right, training and test errors are drawn for two residual networks, one model with 18 layers and the other one with 34 layers.

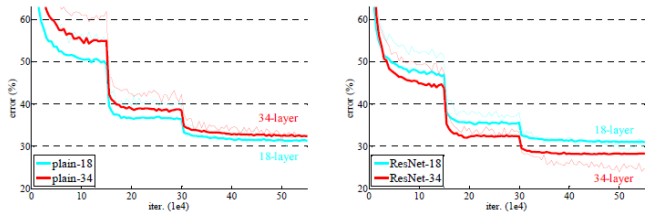

Fig. 3 Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts [8]

In Fig. 3, we can clearly see that the validation error of the 34 layers-residual network is the lowest one among the validation errors of all other networks. Both residual networks in this figure, have no extra parameters compared to their plain counterparts.

The study of [8] demonstrated that by increasing the depth of the network, and using residual networks, better validation accuracies can be obtained. We can conclude from this, that adding more residual blocks to the network helps with generalization.

IV. Implementation and Experiment Results

A. Dataset Used

We empirically demonstrate the effectiveness of the residual network built in our study on the EMBER dataset [7]. It consists of 900K training samples (300K malicious, 300K benign, and 300K unlabelled) where only the benign and the malicious ones were utilized in our experiments. In addition to the training samples, it contains 200K test samples (100K malicious and 100K benign). The training of our model was done on 80% of the 600K samples training dataset, the remaining 20% of the data were used for the validation at the end of each epoch. The 200K test samples were totally kept to the end of the training in order to evaluate the model on previously unseen data. The employed features are values extracted from the PE header fields. These features consist of information describing the executable file, like the virtual size of the file, the virtual address where the program will start the execution, API import and export functions, the number and size of different sections in the file, etc. As shown in [2], various researches adopt the PE features for building their models obtaining high accuracy rates. Fig. 4 shows an example of the extracted features from an executable in our study as given by [7].

B. Methodology

The vectorizing of raw features is provided by the EMBER authors [7], so each sample is represented as a feature vector of dimension 2351. A residual network needs images as inputs. To implement our malware detection model, we chose to represent each sample as a greyscale image of 50 by 50, so we padded each vector by 149 zeros, then we reshaped it to an array of 50 by 50 to feed the model.

CNNs were largely used for malware detection. As far as we know, residual networks were not yet employed in this field. Motivated by the success of these networks in image classification primarily, then in many other fields, we were convinced that they will give favourable results in the classification of legitimate and malicious files.

Various implementations were tested in our experiments. After comparison, we retained the implementation described below.

We chose to add noise to the input layer through a Gaussian noise layer. This layer has a regularizing impact and helps to lessen overfitting. Then, a 2D convolutional layer is added. 64 filters are used, which means trying to detect 64 different features.

ISSN 1737-930X

"**sha256**": "000185977be72c8b007ac347b73ceb1ba3e5e4dae4fe98d4f2ea92250f7f580e",
"**appeared**": "2017-01",
"**label**": -1,
"**general**": {
"**file_size**": 33334,
"**vsize**": 45056,
"**has_debug**": 0,
"**exports**": 0,
"**imports**": 41,
"**has_relocations**": 1,
"**has_resources**": 0,
"**has_signature**": 0,
"**has_tls**": 0,
"**symbols**": 0
},
"**header**": {
"**coff**": {
"**timestamp**": 1365446976,
"**machine**": "I386",
"**characteristics**": ["LARGE_ADDRESS_AWARE", ..., "EXECUTABLE_IMAGE"]
},
"**optional**": {
"**subsystem**": "WINDOWS_CUI",
"**dll_characteristics**": ["DYNAMIC_BASE", ..., "TERMINAL_SERVER_AWARE"],
"**magic**": "PE32",
"**major_image_version**": 1,
"**minor_image_version**": 2,
"**major_linker_version**": 11,
"**minor_linker_version**": 0,
"**major_operating_system_version**": 6,
"**minor_operating_system_version**": 0,
"**major_subsystem_version**": 6,
"**minor_subsystem_version**": 0,
"**sizeof_code**": 3584,
"**sizeof_headers**": 1024,
"**sizeof_heap_commit**": 4096
}
},
"**imports**": {
"KERNEL32.dll": ["GetTickCount"],
...
},
"**exports**": []
"**section**": {
"**entry**": ".text",
"**sections**": [
{
"**name**": ".text",
"**size**": 3584,
"**entropy**": 6.368472139761825,
"**vsize**": 3270,
"**props**": ["CNT_CODE", "MEM_EXECUTE", "MEM_READ"]
},
...
]
},
"**histogram**": [3818, 155, ..., 377],
"**byteentropy**": [0, 0, ... 2943],
"**strings**": {
"**numstrings**": 170,
"**avlength**": 8.170588235294117,
"**printabledist**": [15, ... 6],
"**printables**": 1389,
"**entropy**": 6.259255409240723,
"**paths**": 0,
"**urls**": 0,
"**registry**": 0,
"**MZ**": 1
},
}

Fig. 4 Raw features extracted from a PE file [7]

After that, a Batch Normalization layer is added. As explained by [12], a batch normalization consists of performing the normalization for each training mini-batch. This procedure accelerates the training of deep neural networks.

The chosen activation function is the Rectified Linear Unit ReLU. The role of each activation function is to do a non-linear transformation to the input data, making the model qualified to achieve more complex tasks. ReLu is the more popular activation function nowadays. As demonstrated by [19], the main benefit of ReLu is the non-saturation of its gradients, which enormously hasten the convergence of stochastic gradient descent in comparison with the sigmoid and the tanh functions.

Next, a max pooling layer is introduced. This layer has no parameters to learn, it takes a region of the convolved image of a fixed filter size, in our case, this region has a dimension of 3 * 3, then selects the maximum value of its different values; the 9 values in our case. The biggest role of this layer is to continuously lessen the spatial size of the feature maps, as said earlier, and consequently to reduce the amount of computations on the network, while preserving the most important information.

Subsequently, two residual blocks are added. Each residual block consists of two convolutional layers with batch normalization and a ReLU activation function, followed by adding a shortcut connection to the output of these two convolutional layers, then passing the result again through a ReLU activation. Finally, since we are involving binary decisions, either malicious or benign class, we are using binary cross-entropy as loss function.

C. Experiments and Evaluation Results

All the experiments in this paper were conducted on a laptop computer with Intel® Core (TM) i7-6500U @ 2.50 GHz, 2.59 GHz, and 16 GB of RAM.

Based on the dataset described in Section IV.A, we evaluated the experiments in two perspectives; a residual network model without dropout regularization and a residual network model with dropout regularization.

After several attempts of parameter tuning, for each model type, the best results were obtained using a learning rate of 0.00007, a batch size of 256 and 64 filters for the different convolutional layers. We trained each model for 80 epochs.

The figures 5 and 6 represent the training and validation loss and accuracy obtained with both models, without and with dropout regularization. After a certain epoch, the model continues to learn peculiarities of the training data and is not any more able to generalize well to previously unseen data. Therefore, we had to opt for an early stopping approach in order to get the best test accuracy. Our code saved the trained model at the end of each epoch, in order to be able to adopt the most fitting one. The epoch to stop at is different from a model to another.

For the residual network model without dropout regularization, the best validation accuracy was obtained at epoch 72, namely 93%, however the test of the saved model at this epoch on the 200,000 previously unseen test samples,

ISSN 1737-930X

lessened enormously, resulting in 83.10% as test accuracy. The second-best validation accuracy was observed at epoch 27, namely 91.01%. Here we obtained a test accuracy of 88.30%. Therefore, we retained the model saved at epoch 27. The accuracy decreased by 2.71% from validation to testing data. This decrease is a normal behavior of machine learning models as seen on [2], as the models are exposed in this final phase to totally new data.

As for our residual network model with dropout regularization, the best validation accuracy was obtained at epoch 79, namely 92.29%. At this epoch, the saved model presented the best test accuracy, that is 90.38%. Here we have an accuracy decrease of 1.91%.

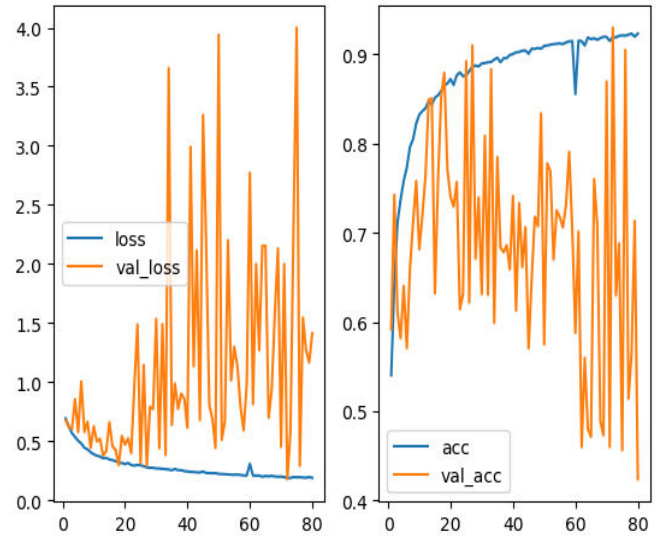

Fig. 5 Train and validation loss and accuracy obtained with our ResNet model without dropout regularization

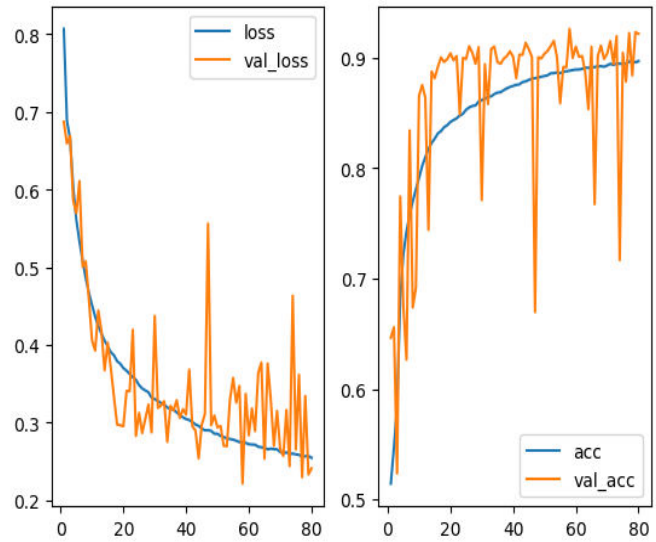

Fig. 6 Train and validation loss and accuracy obtained with our ResNet model with dropout regularization

The final test accuracies of both models are shown in Table1.

TABLE I
TEST ACCURACIES ON EMBER TEST DATASET FOR BOTH MODELS

	Test Accuracy
Without Dropout	88.30%
With Dropout	90.38%

We see evidently that the residual network model with dropout regularization performs better giving us an accuracy of 90.38%. This accuracy is even better than some other previous researches on malware detection using machine learning as shown in Table 2.

TABLE II
COMPARISON OF OUR MODEL WITH SOME PREVIOUS RESEARCHES

	Accuracy
Random forest [3]	86.70%
CNNs [5]	90%
Our ResNet model	**90.38%**

V. CONCLUSIONS

The use of residual networks in malware detection is new as far as we know. The model built in this paper represents a first approach using these powerful networks in this field. The obtained results were satisfying as shown. However, they can be much more improved in the future with more hyper-parameters tuning, with the employment of feature selection techniques, and with adopting other types of features in addition to PE features. Implementing deeper residual networks may also help on getting better results. However, the training of these deeper networks can be painfully time-consuming and needs more powerful machines with GPUs and larger RAM sizes.

REFERENCES

[1] The AV-TEST website. *The Independent IT Security Institute*. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[2] H. El Merabet and A. Hajraoui, "A survey of malware detection techniques based on machine learning," *International Journal of Advanced Computer Science and Applications (IJACSA),* vol. 10, pp. 366–373, Jan. 2019.

[3] P. Burnap, R. French, F.Turner and K. Jones. "Malware classification using self-organizing feature maps and machine activity data," *Journal Computers and Security,* vol. 73, pp. 399–410, 2018.

[4] F. Xiao, Z. Lin, Y. Sun and Y. Ma. "Malware detection based on deep learning behavior graphs," *Mathematical Problems in Engineering,* vol. 2019, Feb 2019.

[5] M. Abdelsalam, R. Krishnan, Y. Huang and R. Sandhu. "Malware detection in cloud infrastructures using convolutional neural networks," in *Proc. 11th IEEE Conference on Cloud Computing (CLOUD)*, San Francisco, CA, Jul 2-7 2018.

[6] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro and C. Nicholas. "Malware detection by eating a whole exe," *AAAI Workshop on Artificial Intelligence for Cyber Security*, 2018.

[7] H. Anderson and P. Roth. "EMBER: An open dataset for training static PE malware machine learning models," in *ArXiv e-prints*. Apr 2018.

[8] K. He, X. Zhang, S. Ren and J. Sun. "Deep residual learning for image recognition," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] L. Xiaofeng, Z. Xiao, J. FangshuoY. Shengwei and S. Jing. "ASSCA: API based sequence and statistics features combined malware detection architecture" in *Procedia Computer Science 129,* Jan 2018, pp 248-256.

ISSN 1737-930X

[10] A. Krizhevsky. "Learning multiple layers of features from tiny images," *Tech Report*, 2009.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al. "Imagenet large scale visual recognition challenge," in *ArXiv:1409.0575*, 2014.

[12] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *CoRR*, vol. 1502.03167, 2015.

[13] K. He, X. Zhang, S. Ren and J. Sun. "Identity mappings in deep residual networks," in *ArXiv preprint* arXiv:1603.05027, 2016.

[14] A. Conneau, H. Schwenk, L. Barrault and Y. Lecun. "Very deep convolutional networks for natural language processing," in *ArXiv preprint*, arXiv:1606.01781, 2016.

[15] R. Ostling. "Morphological reinflection with convolutional neural networks," in Proc. Of the 2016 Meeting of SIGMORPHON, Berlin, Germany. *Association for Computational Linguistics*.

[16] J. Bjerva. "Byte based language identification with deep convolutional networks," in *ArXiv preprint* arXiv:1609.09004, 2016.

[17] J. Bjerva, B. Plank and J. Bos. "Semantic tagging with deep residual networks," in *Proc. Of COLING*, Osaka, Japan, Dec. 2016.

[18] B. Lim, S. Son, H. Kim, S. Nah and K. M. Lee. "Enhanced deep residual networks for single image super-resolution," in *the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* Workshops, 2017, pp. 136-144.

[19] A. Krizhevsky, I. Sutskever and G. E. Hinton. "ImageNet classification with deep convolutional neural networks," Proc. NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems, vol. 1, pp. 1097-1105, Dec. 2012.

ISSN 1737-930X