MALWARE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

MALWARE ANALYSIS

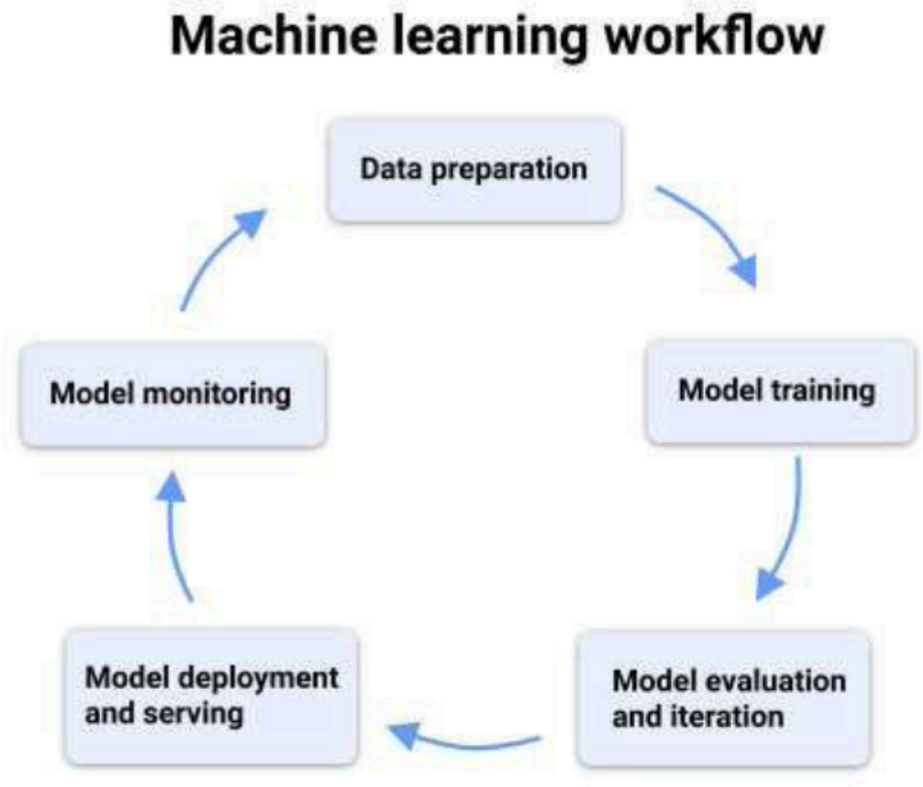
- The process of dissecting a malware to understand how it works and how to identify it.
- Classical approaches for extracting features in malware analysis:
  - Static analysis, code or structure examination without execution of the program;
  - Dynamic analysis, execution of the program and behaviour monitoring;
- A malware signature (fingerprint) is a set of features that uniquely distinguishes an executable.
- Standard antivirus solutions rely on signature and/or heuristic/behavioural databases to detect malware programs.

CLIK HERE

PROBLEMS WITH STANDARD ANTIVIRUS PROGRAMS

- With the growth of malware volumes, malware analysts need scalable and automated tools to handle large-scale malware samples.
- Malware authors continuosly adapt their techniques to evade detection, for example:
  - Unknown malware variants: an attacker can easily create multiple variants of the same malware.
  - Packed or obfuscated malware: compression and encryption algorithms make the analysis more complicated.
  - Polymorphic malware: the malware uses a polymorphic engine to mutate its features while keeping the same functionality.

ML/DL TECHNIQUES FOR MALWARE ANALYSIS



- Machine learning is well suited for processing large volumes of data
- It can facilitate the pattern identification and the analysis process.
- The ML/DL workflow has the objective of training a model to solve a task, in this case malware detection/classification.
- There is a preprocessing phase to extract the features from the executables.

MALWARE CLASSIFICATION AND DETECTION

- **Malware classification is the process of assigning a malware sample to a specific malware family.**

  In this case, the model outputs the probability of belonging to each malware class for a given executable.

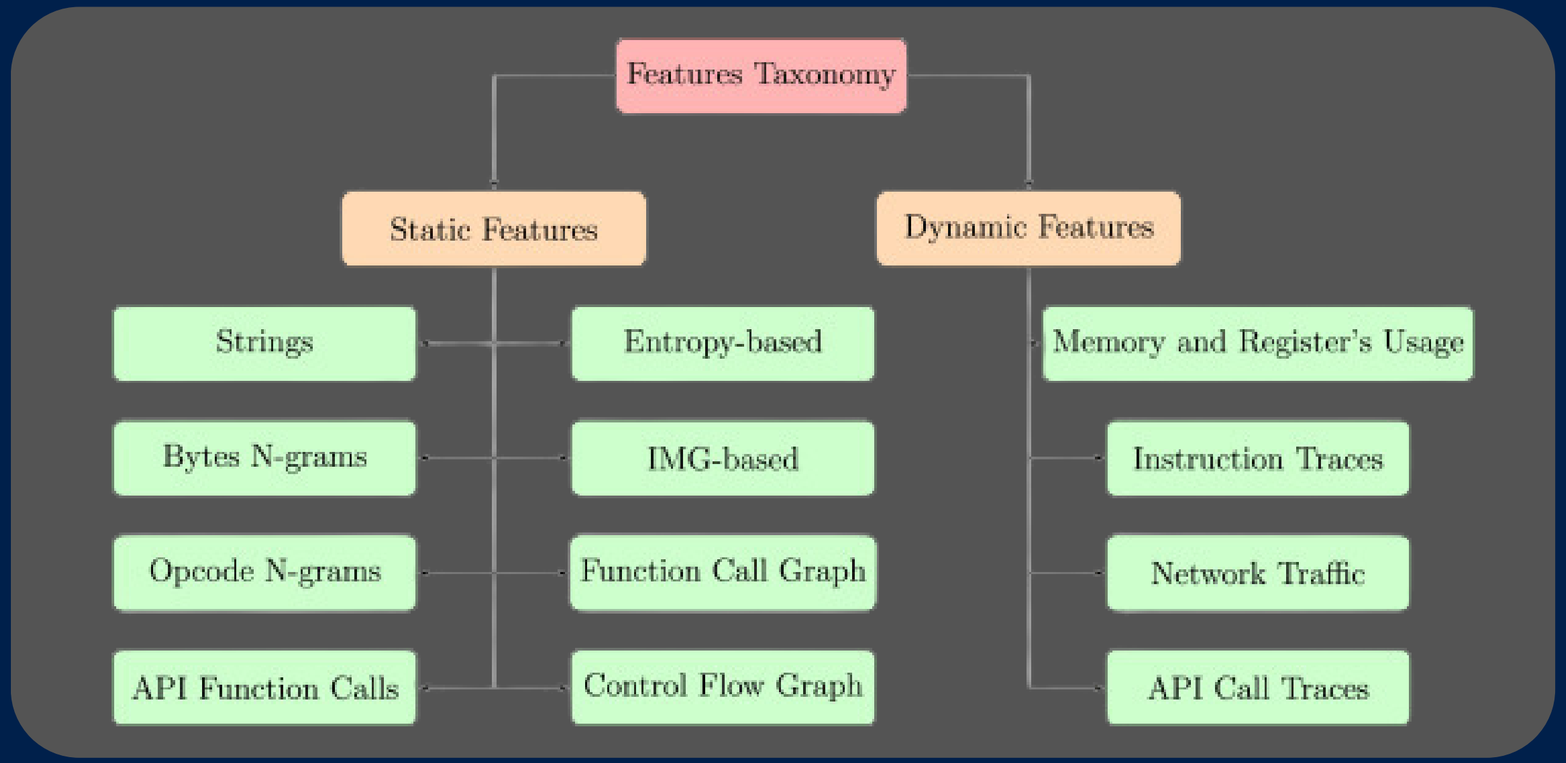
- **Malware detection is the process of estabilishing the maliciousness of an executable.**

  The detection model outputs a single probability (binary model): malicious or benign

EXECUTABLE FEATURES USED IN ML/DL APPROACHES

The feature types can be divided in 2 groups as the malware analysis approaches: static and dynamic features.

Features can be combined together to provide a better representation of an executable.



PE EXECUTABLE FILES

- Portable executable (PE) is the standard binary file format for executables (.exe) and DLLs (.dll) in Windows.
- It encapsulate all the information necessary for the Windows OS to manage the executable code.
- The header gives info about the external functions used by the program.
- The .text section contains the executable code.
- The .data section contains the global variables.

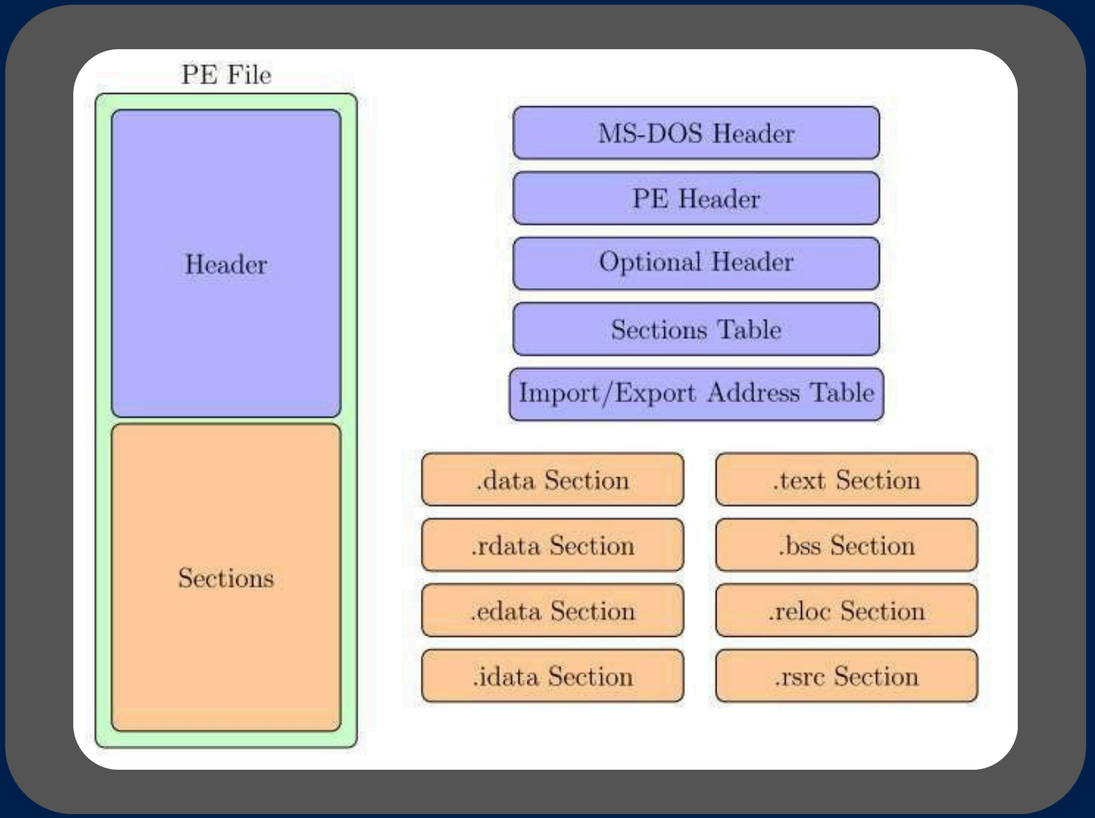
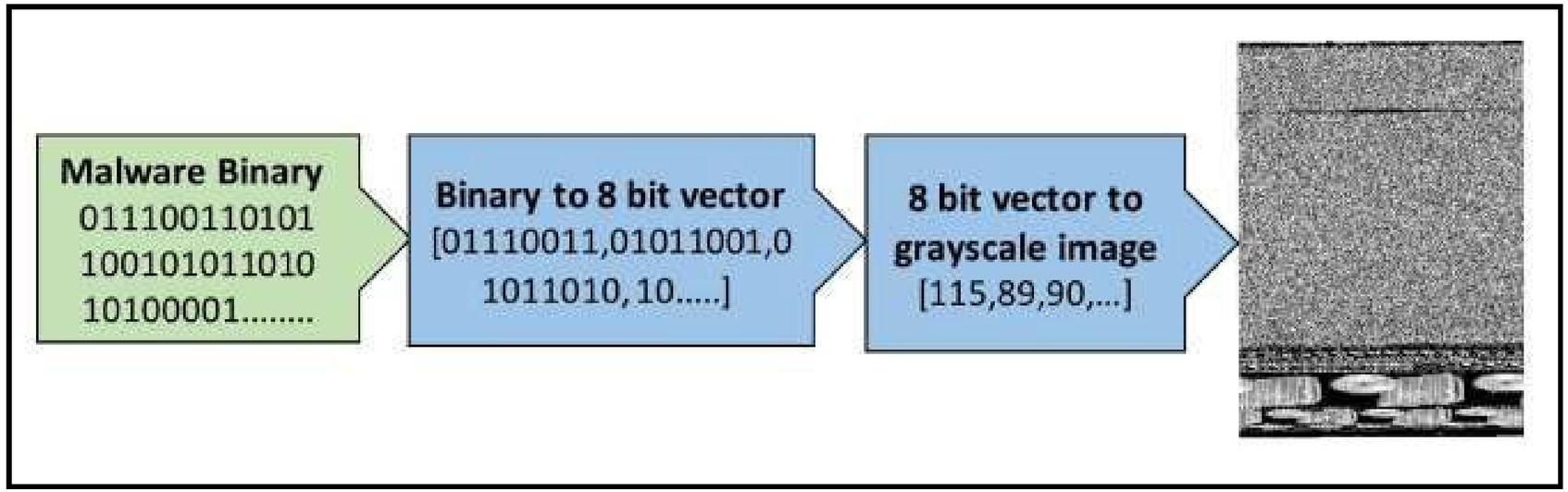


IMAGE-BASED REPRESENTATION OF EXECUTABLES

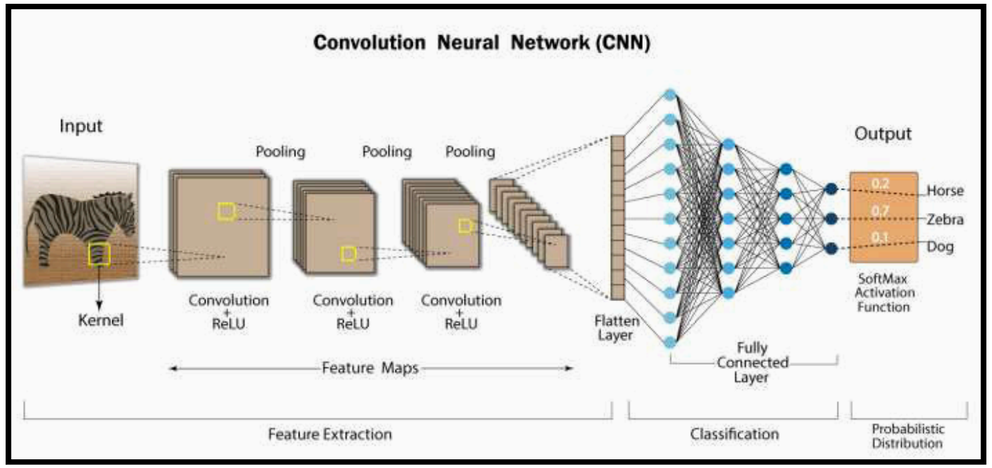
- Each PE executable can be represented as a one-dimensional array of bytes, so with decimal value in the range [0,255].

- The resulting array can be arranged as a 2D array with a reshape to a target image size, obtaining a gray-scale representation of the sample.



CONVOLUTIONAL NEURAL NETWORKS (CNNS)

- **A particular type of Neural Network specifically designed for processing and analyzing images and videos.**
- **The core component is the convolutional layer which uses a moving filter (kernel) to detect patterns in the image. The convolutional layer output is a feature map.**
    - The activation function is used to indicate the existence of likely features in the input signal.
    - Pooling layers reduce the spatial size of the feature map in input and provide robustness against noise.
    - Fully connected layers combine the learned features and determine a specific target output.



CONVOLUTIONAL NEURAL NETWORKS (CNNS)

- The CNN approach can be applied for malware classification by using the image representation of executables as input.
- The main advantage of this approach is that **different sections of a binary can be easily separated.**
- To produce new variants, attackers usually change only a small part of the code. So, **re-using old malware to create new binaries has the effect of generating very similar images to the old executable.**
- Additionally, by representing an executable as a gray scale image it is possible to **detect small variations between the samples of the same family.**
- Zero-padding can be easily detected, often used by attackers to reduce the overall entropy.

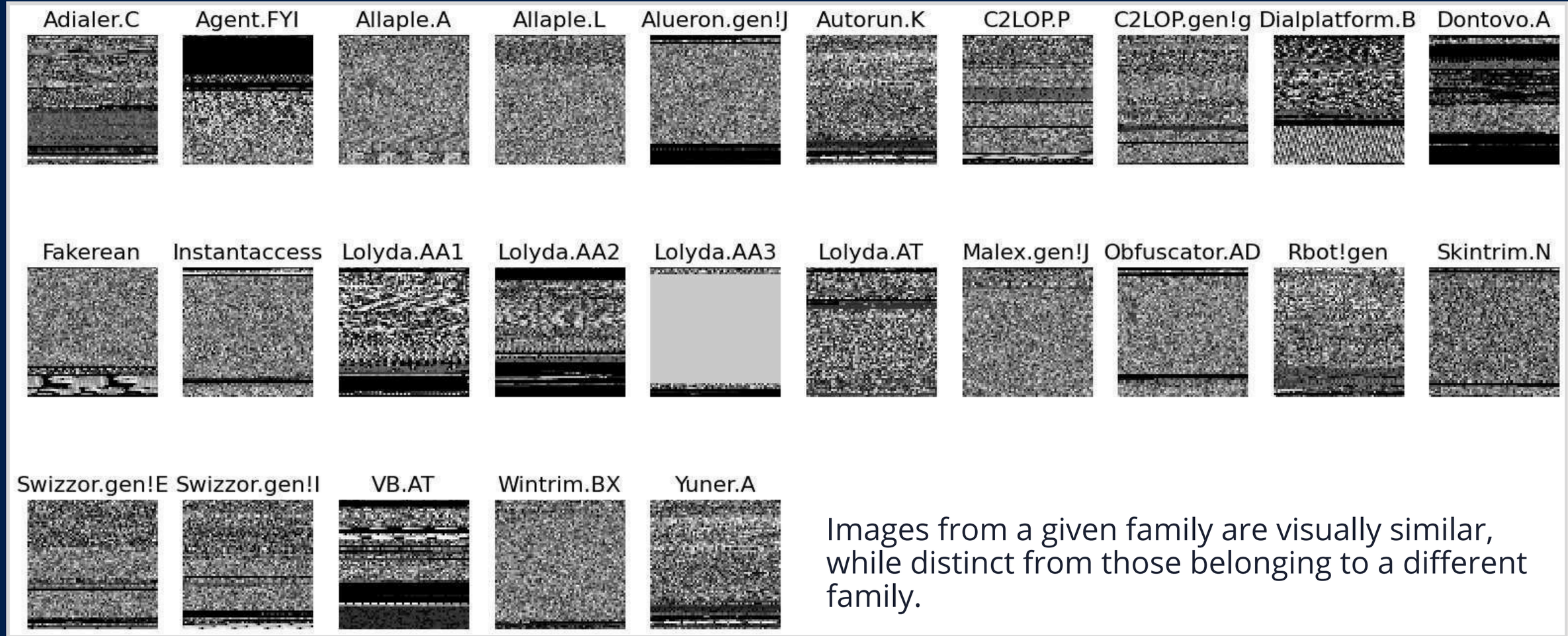
MALIMG DATASET

- Provided by [Nataraj et al.]
- Consists of 9339 gray scale images of 25 malware classes.
- It contains samples of malicious software packed with UPX: Autorun.K, Malex.gen!J, Rbot!gen, VB.AT, and Yuner.A.
- There are several family variants of the same malware such as Lolyda and Allaple.

No.	Type	Malware family	# di Img
1	Worm	Allaple.L	1591
2	Worm	Allaple.A	2949
3	Worm	Yuner.A	800
4	PWS	lolyda.AA 1	213
5	PWS	lolyda.AA 2	184
6	PWS	lolyda.AA 3	123
7	Trojan	C2Lop.P	146
8	Trojan	C2Lop.gen!G	200
9	Dialer	Instantaccess	431
10	Trojan Downloader	Swizzor.gen!I	132
11	Trojan Downloader	Swizzor.gen!E	128
12	Worm	VB.AT	408
13	Rogue	Fakerean	381
14	Trojan	Alueron.gen!J	198
15	Trojan	Malex.gen!J	136
16	PWS	Lolyda.AT	159
17	Dialer	Adialer.C	125
18	Trojan Downloader	Wintrim.BX	97
19	Dialer	Dialplatform.B	177
20	Trojan Downloader	Dontovo.A	162
21	Trojan Downloader	Obfuscator.AD	142
22	Backdoor	Agent.FYI	116
23	Worm:AutoIT	Autorun.K	106
24	Backdoor	Rbot!gen	158
25	Trojan	Skintrim.N	80

MALIMG DATASET

Dataset samples for each classe :



Images from a given family are visually similar, while distinct from those belonging to a different family.

Malimg class examples

- **Dontovo.A class**: is a trojan that downloads and executes arbitrary files.
- Installation:
  - When executed Win32/Dontovo.A runs a copy of %Windows%\svchost.exe and injects code into it.
  - It then deletes its executable.
  - Process injection MITRE ATT&CK T1055.
- Payload:
  - Through svchost.exe, the process contacts the following domain (or others) for configuration data: iframr.com.
  - Downloaded files are saved to the %temp% directory and executed.

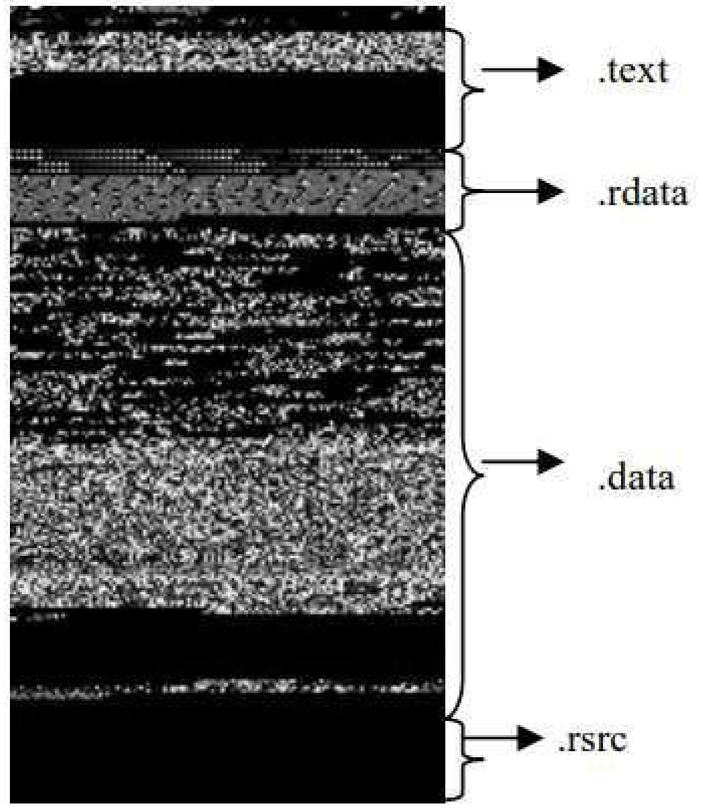
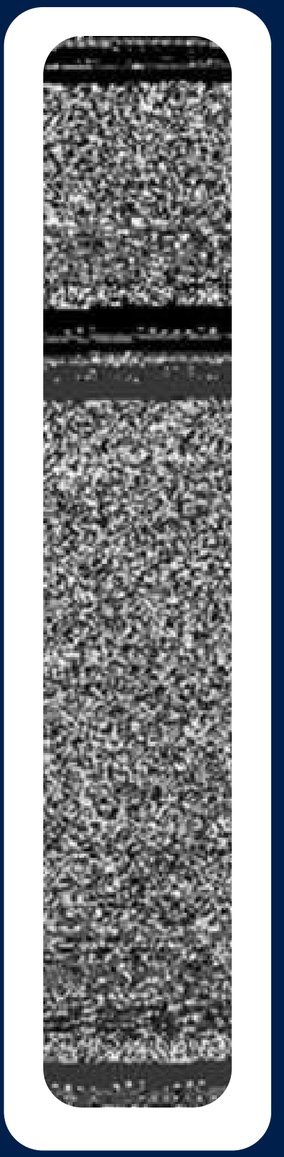


Fig. 2 Various Sections of Trojan: Dontovo.A

Malimg class examples

- **Lolyda.AT class**: is from a family of trojans that steals account information from popular online games and sends it to a remote server.
- It can also take screenshots, terminate processes, and hook certain APIs.
- Installation:
  - when executed, PWS:Win32/Lolyda.AT drops a DLL with a randomly-generated file name into the Windows system folder.
  - It then modifies the registry to ensure that it is loaded by the 'explorer.exe' process.
  - Modify Registry MITRE ATT&CK T1112, DLL injection MITRE ATT&CK T1055.001.
- Payload:
  - searches the running process memory of several online games to find usernames, passwords, server addresses and characters information.
  - Periodically checks if the foreground window title has the following strings: ACDSee, Internet Explorer. If found, it takes a screenshot and saves it in Windows temporary folder.
  - Hooks APIs, preventing the normal communication between the game client and the game server.

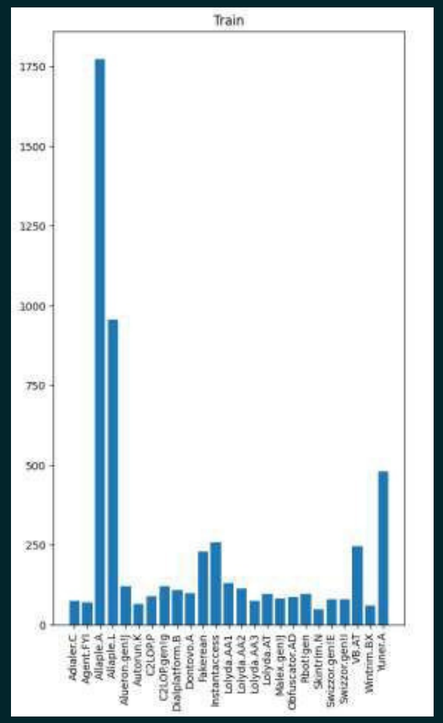


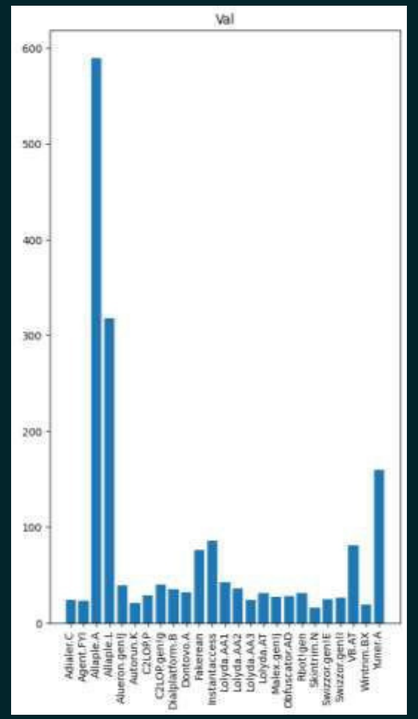
LOLYDA.AT SAMPLE

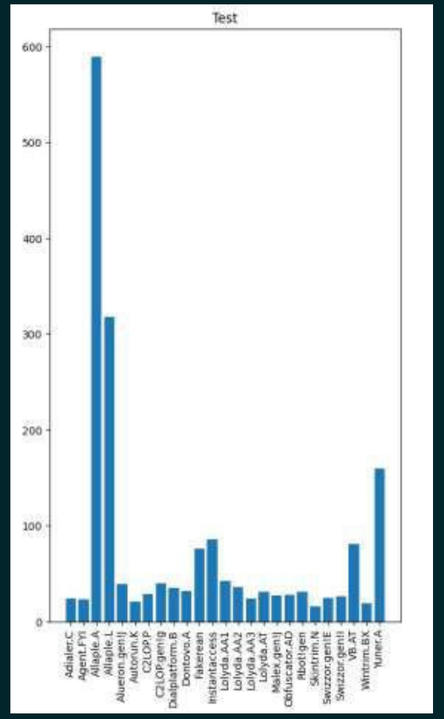
Dataset splitting

- Data partition applied for each class is the following:

60% for training – 20% for validation – 20% for test



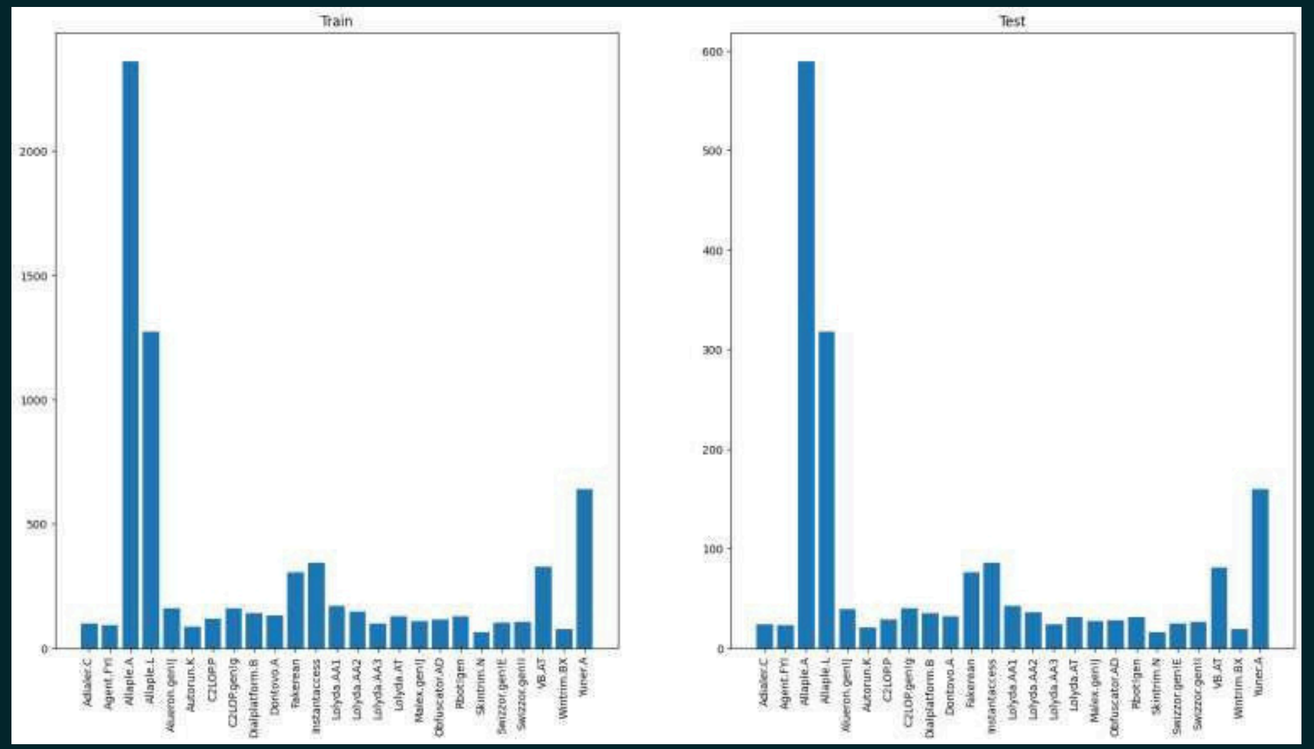




Dataset splitting

- After the best model has been found, validation and training sets can be merged as :

80% for training 20% for test



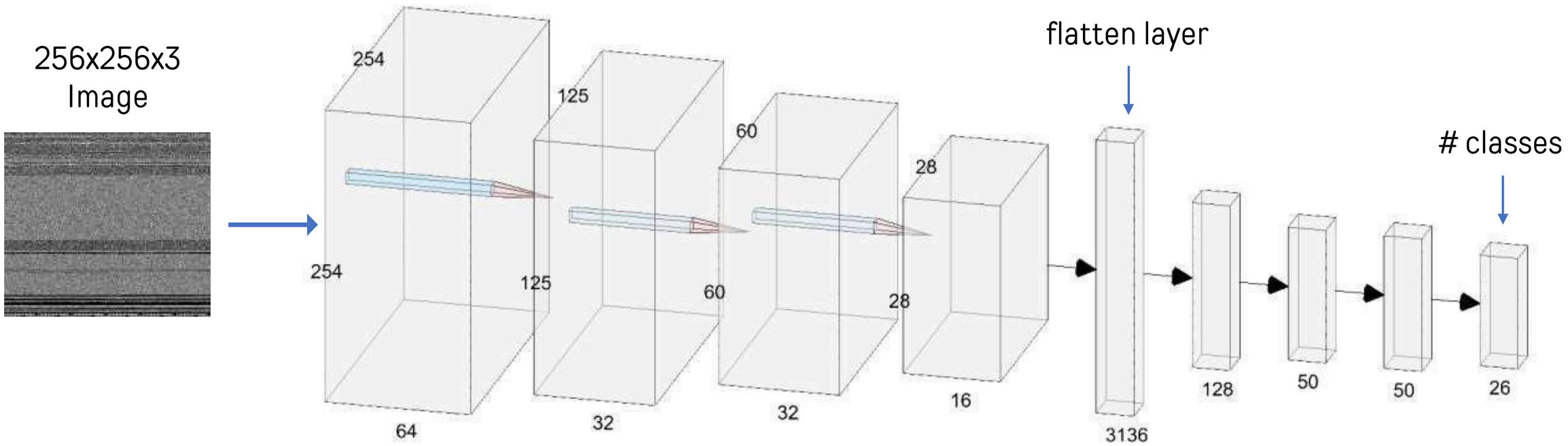
Building the CNN

- Tensorflow and Keras were the frameworks used for building and training the CNN.
- All the experiments were executed locally using jupyter notebook

```
def malware_model():
    Malware_model = Sequential()
    Malware_model.add(Conv2D(64, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=(target_size_custom[0],target_size_custom[1],3)))

    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=(target_size_custom[0]//2,target_size_custom[1]//2,3)))

    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=(target_size_custom[0]//4,target_size_custom[1]//4,3)))

    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Conv2D(16, (3, 3), activation='relu'))
    Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
    Malware_model.add(Dropout(0.25))
    Malware_model.add(Flatten())
    Malware_model.add(Dense(128, activation='relu'))
    Malware_model.add(Dropout(0.25))
    Malware_model.add(Dense(50, activation='relu'))
    Malware_model.add(Dropout(0.5))
    Malware_model.add(Dense(num_classes, activation='softmax'))
    Malware_model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=["accuracy"], weighted_metrics=['accuracy'])
    return Malware_model
```

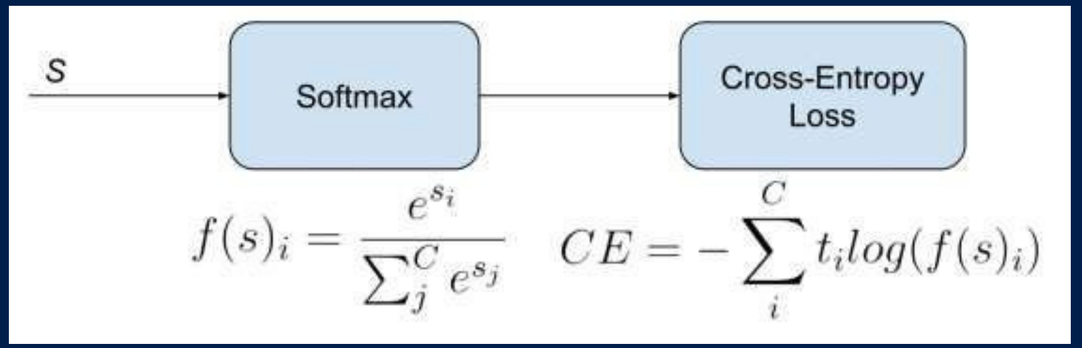
Building the CNN

- Only trainable convolutional layers are showed.
- Between each of the 2D layer, there is a max pooling layer and a dropout layer.
- Between each dense layer, there is a dropout layer with difference of 0.5



Building the CNN

- **Loss function:**
  - is a mathematical function that measures the discrepancy between the predicted output of a model
  - and the true or expected output.
  - The choice of the loss function depends on the specific problem and the nature of the data.
  - In this case, a multi-class classification problem, the Categorical Cross-Entropy (Softmax loss) is used:



i and j iterate through classes
C is the number of classes
s is the prediction vector
t is the ground truth vector

- **Optimizer Adam:**

  It stands for "Adaptive Moment Estimation“

  It is an adaptive optimization algorithm commonly used in training deep learning models.

Hyperparameters tuning

- **Batch size:**
  - indicates the number of training examples used in one iteration of the training process.
  - Trade-off between larger batch-size (faster training time) and small batch-size (better model generalization).
  - In this case a batch size = 32 is chosen.
- **Target image size:**
  - Refers to the image given in input to the CNN.
  - In this case 256x256 pixels is chosen.
- **Learning rate (LR):**
  - Determines the step size at which the model updates its parameters during the training.
  - If the LR is too high the model, the model may fail to converge, otherwise if it is too
  - low the model will slow down the convergence.
  - In this case a LR = 0.001 is chosen.

Evaluation Metrics

The following metrics are applied for each class:

- *Precision*: $$P = \frac{T_p}{T_p + F_p}$$
- *F1 Score*: $$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$
- *Recall*: $$R = \frac{T_p}{T_p + Fn}$$
- *Accuracy*: $$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

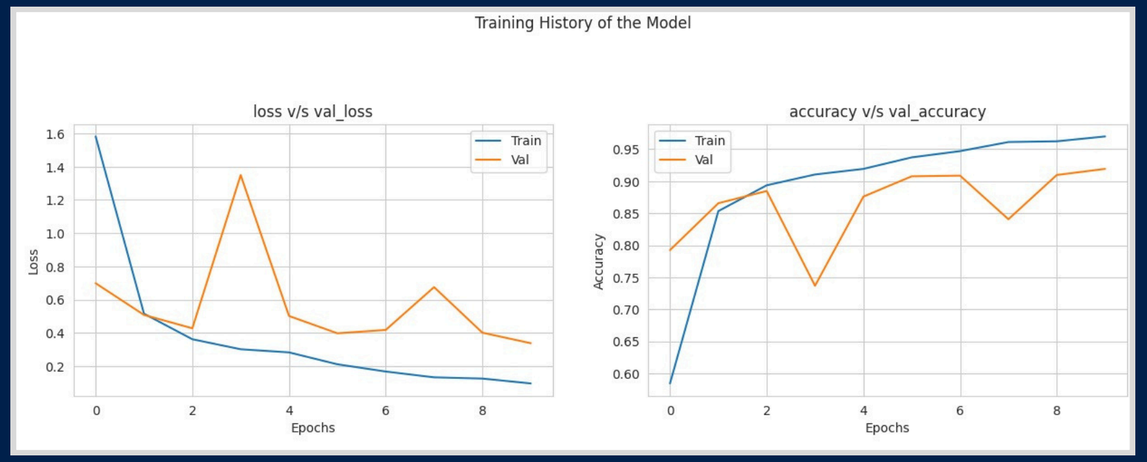
- Then, the average of the individual metrics is calculated, obtaining:
  - *macro_precision*, *macro_recall*, *macro_f1-score, avg_accuracy;*
- and the weighted average of the metrics:
  - *weighted_precision*, *weighted_recall*, *weighted_f1-score, weighted_accuracy.*

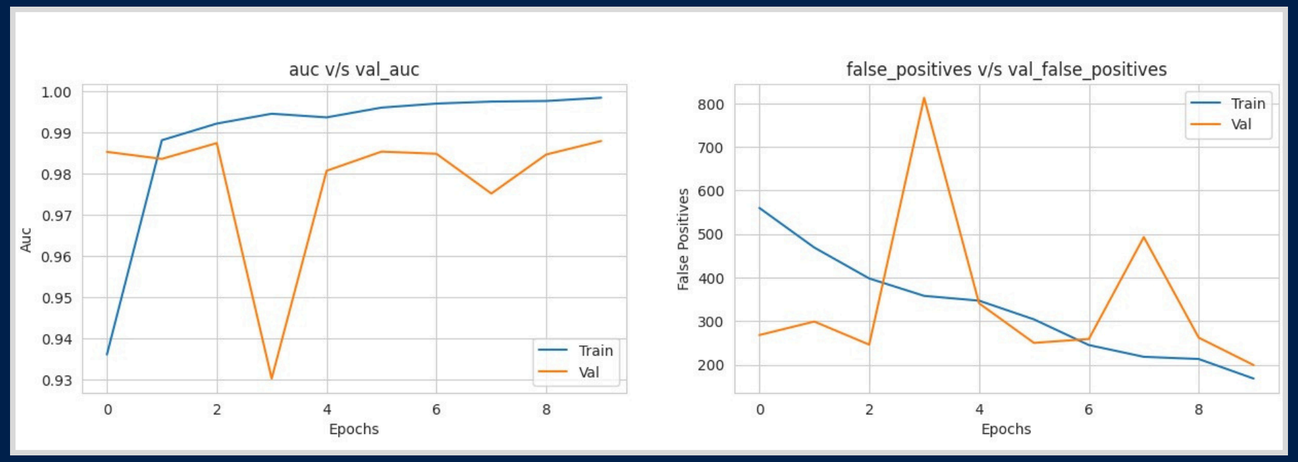
Training phase on Malimg dataset

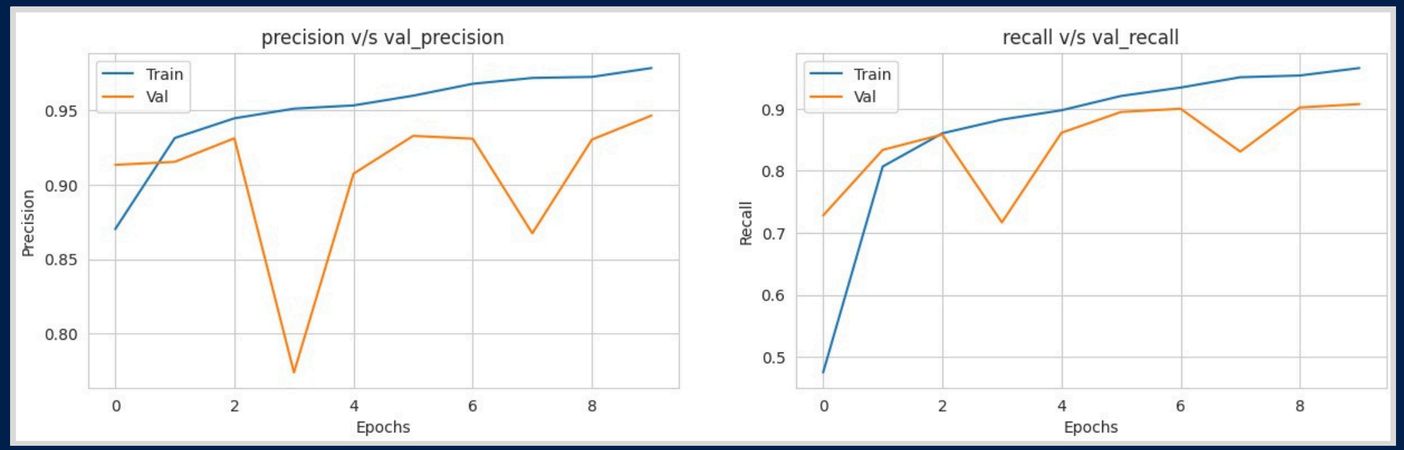
- Training on 10 epochs.
- Using the validation data.
- Using the class weights to balance the dataset.
- To evaluate the quality of training different metrics are plotted:
  - Training loss and validation loss;
  - Accuracy and validation accuracy;
  - Area under curve and validation area under curve;
  - False positives and validation false positives
  - Precision and validation precision
  - Recall and validation recall

- Average metrics obtained on the validation set (1858 samples):
  - loss: 0.0987
  - accuracy: 0.9709
  - auc: 0.9985
  - false_positives: 81.0686
  - precision: 0.9797
  - recall: 0.9646

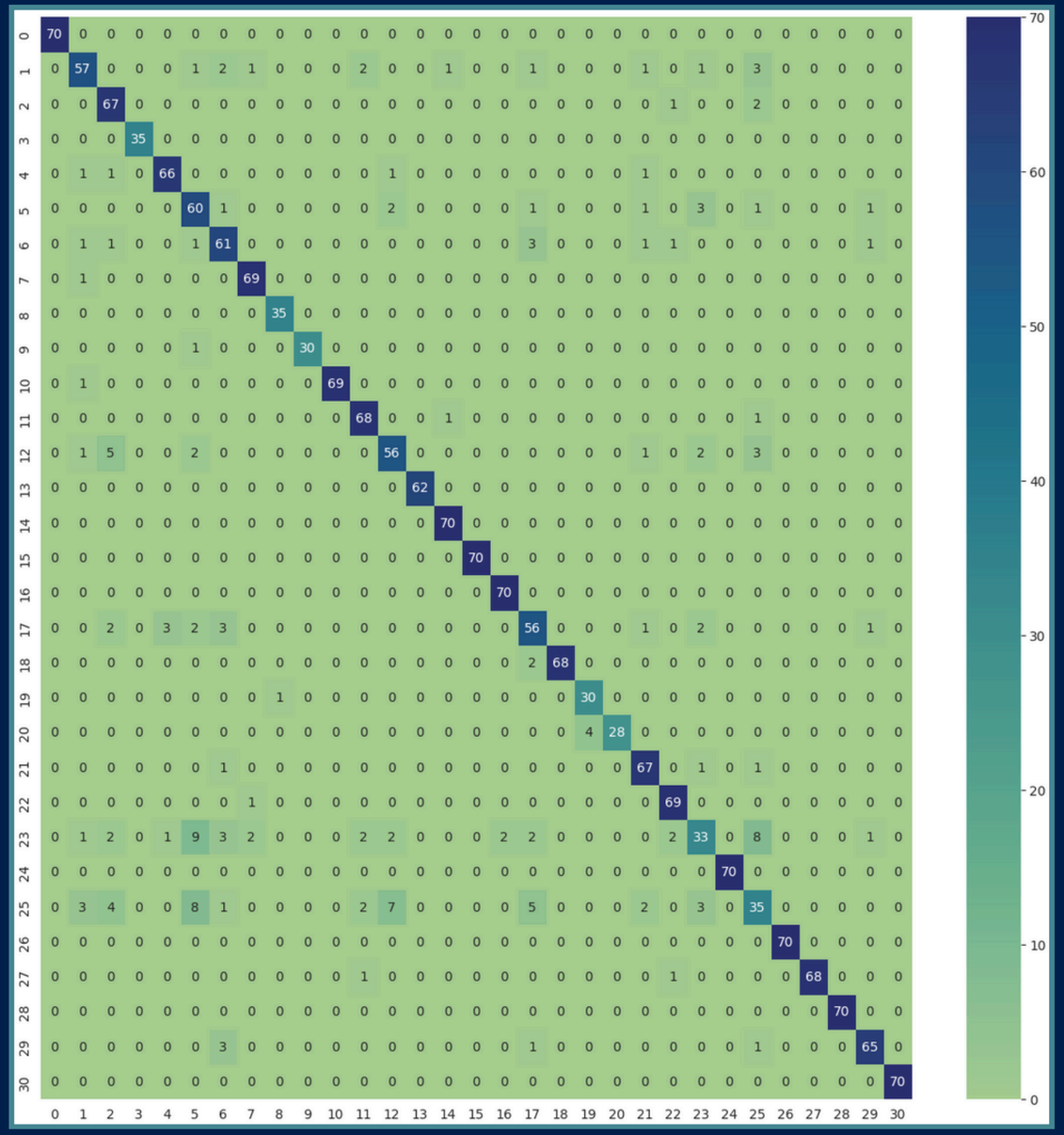
Training phase on Malimg dataset



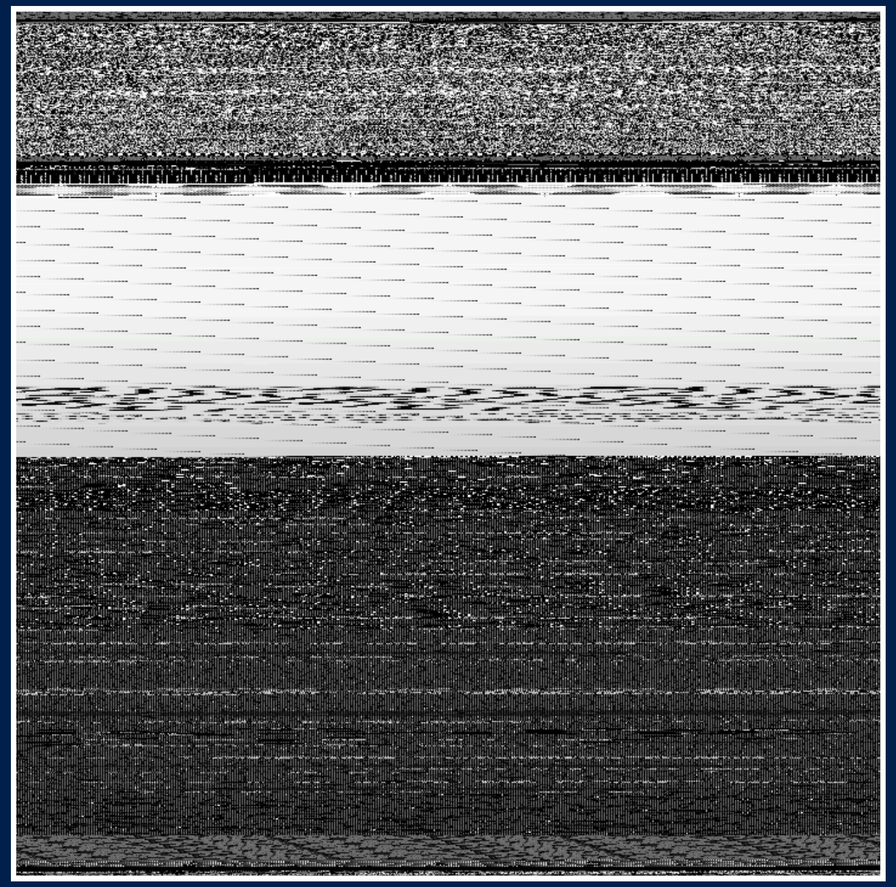




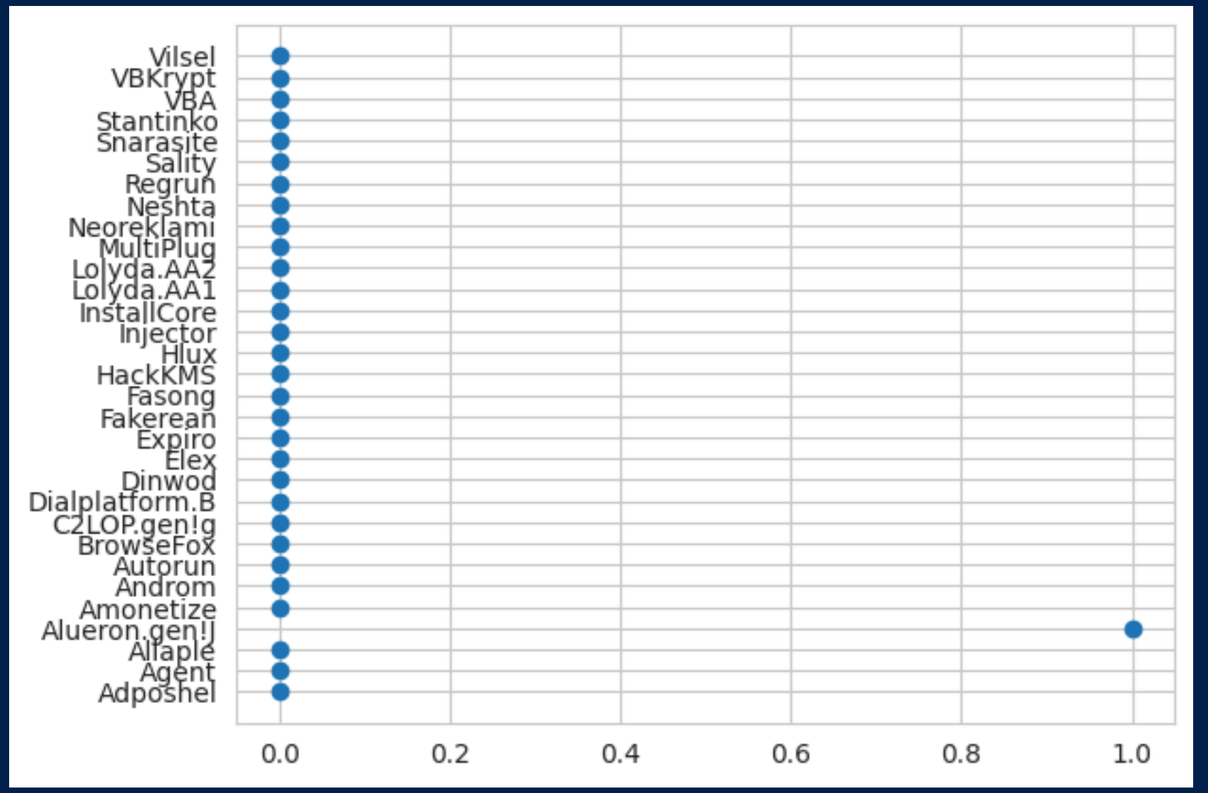
Evaluation phase on validation set



Prediction example result:



input image: DXSETUP.EXE.png generated from benign DirectX setup executable.



Prediction results:
Probability of input belonging to 30 classes of malware: 0.0 with 1 false positive

Conclusions

- The gray scale image representation of executables has some drawbacks related to how images are generated:
  - New hyperparameter to tune: image size.
  - Imposing spatial correlation between pixels in different rows, which is not always true.

- Although the drawbacks, the final model can differentiate between malicious and
- benign data.

- The malware detection task can be improved by:
  - Aggregating all the malware classes under one malicious class;
  - Collecting more benign samples in the wild;

Future Work

- A more generalizable approach is the multimodal learning where different feature vectors, belonging from different inputs of the PE executable (strings, api calls, control flow graphs ecc.), can be used.
  - For each feature vector there is a classifier;
  - A fusion layer gathers all the predictions to decide the final output.

- Because of the continuous evolution of malware and its variants, another important task to achieve is the class incremental learning:
  - a model, pretrained on a set of malware classes, gains new knowledge
  - by learning new malware classes without forgetting the old ones.

References

- Gibert, D., Mateu, C., Planes, J. et al. Using convolutional neural networks for classification of malware represented as images. Using convolutional neural networks for classification of ... – Springer.

- Daniel Gibert, Carles Mateu, Jordi Planes, Journal of Network and Computer Applications, The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. – ScienceDirect.

- Songqing Yue, Tianyang Wang, Imbalanced Malware Images Classification: a CNN based Approach.

- Nataraj, Lakshmanan & Karthikeyan, Shanmugavadivel & Jacob, Grégoire & Manjunath, B.. (2011). Malware Images: Visualization and Automatic Classification. 10.1145/2016904.2016908. Malware Images: Visualization and Automatic Classification – ResearchGate.

- M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang and F. Iqbal, "Malware Classification with Deep Convolutional Neural Networks," 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328749.

- Tuan, Anh Pham; Phuong, An Tran Hung; Thanh, Nguyen Vu; Van, Toan Nguyen (2018). Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset.

Thank you!