Malware Detection Using Visualization Techniques

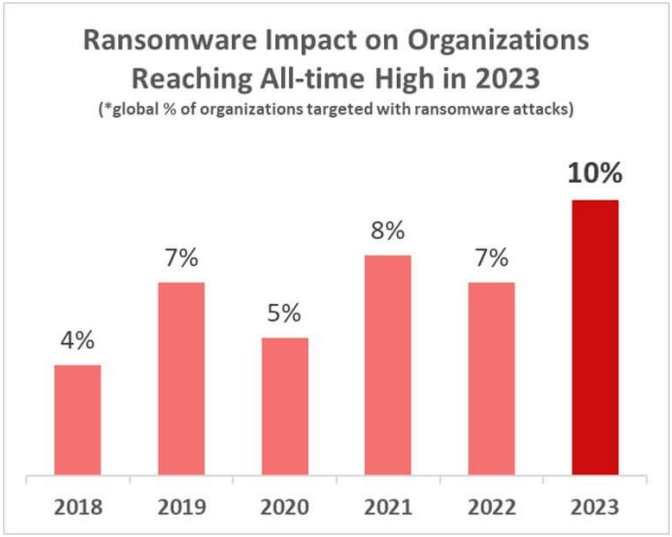
Bc. Ihor Salov

Faculty of Information Technology
Department of Information Security
Supervisor: Mgr. Olha Jurečková

Malware threat

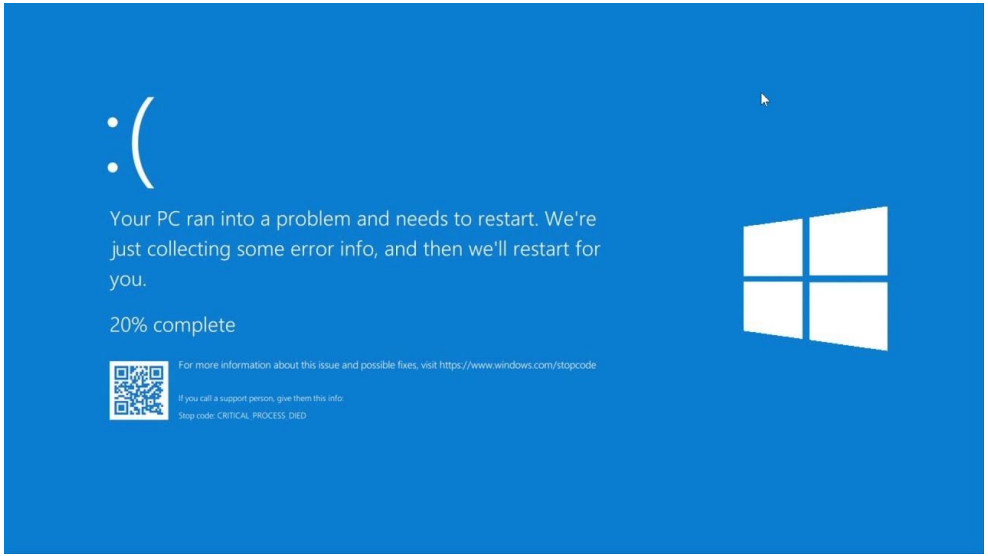
Malware threat

10 Billion Attacks Blocked in 2023, Qakbot's Resurrection, and Google API Abused



The visage of malware

The visage of malware

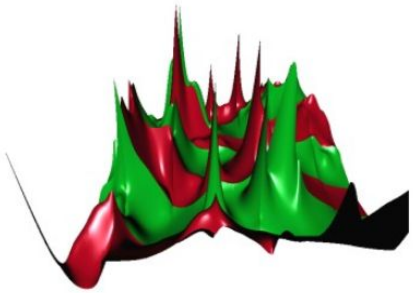


The visage of malware

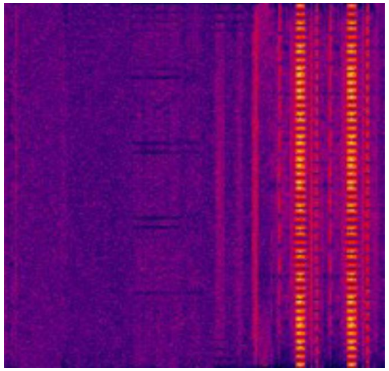
The visage of malware

```
isalov@ISALOV-P14S:~/saloviho-DP$ file ransomware.exe
ransomware.exe: PE32 executable, for MS Windows
isalov@ISALOV-P14S:~/saloviho-DP$ xxd ransomware.exe | head -n 20
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
00000010: b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 f800 0000  ................
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th
00000050: 6973 2070 726f 6772 616d 2063 616e 6e6f  is program canno
00000060: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS
00000070: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.......
00000080: f5cd 1312 b1ac 7d41 b1ac 7d41 b1ac 7d41  ......}A..}A..}A
00000090: 0530 8c41 bdac 7d41 0530 8e41 3eac 7d41  .0.A..}A.0.A>.}A
000000a0: 0530 8f41 acac 7d41 d2f1 7e40 a7ac 7d41  .0.A..}A..~@..}A
000000b0: d2f1 7840 8bac 7d41 d2f1 7940 93ac 7d41  ..x@..}A..y@..}A
000000c0: 6c53 b641 baac 7d41 b1ac 7c41 d8ac 7d41  lS.A..}A..|A..}A
000000d0: dff1 7440 b0ac 7d41 dff1 8241 b0ac 7d41  ..t@..}A...A..}A
000000e0: dff1 7f40 b0ac 7d41 5269 6368 b1ac 7d41  ...@..}ARich..}A
000000f0: 0000 0000 0000 0000 5045 0000 0000 0800  ........PE......
00000100: ce20 7158 0000 0000 0000 0000 e000 0201  . qX............
00000110: 0b01 0e00 00ec 0100 0032 0100 0000 0000  .........2......
00000120: 5256 0000 0010 0000 0000 0200 0000 4000  RV............@.
00000130: 0010 0000 0002 0000 0600 0000 0000 0000  ................
```

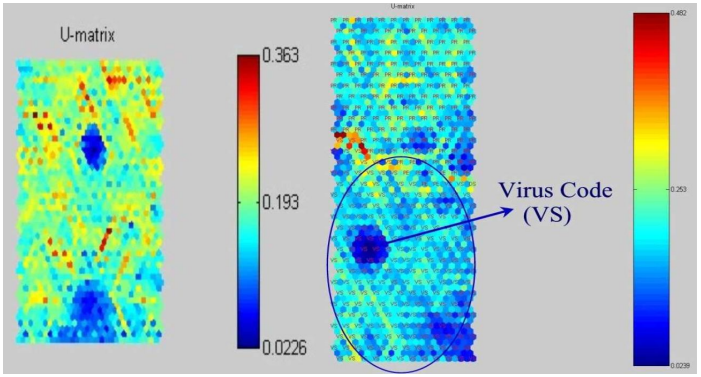
Malware visualization



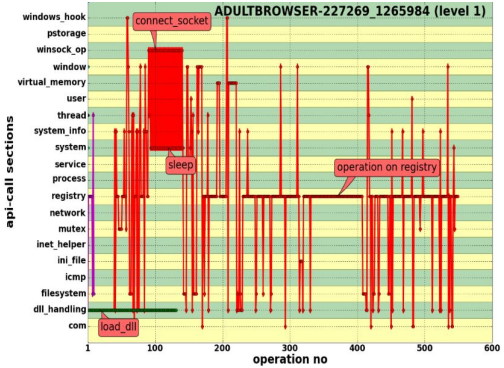
3D-visualization



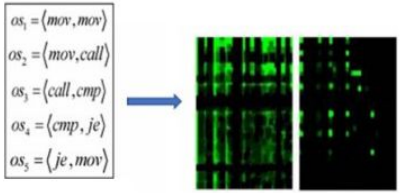
Spectrogram



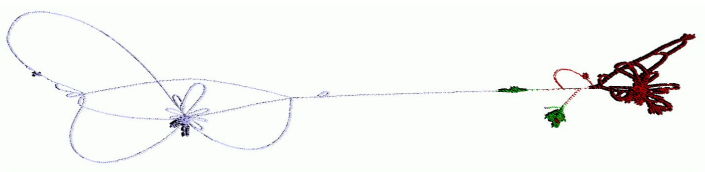
SOM visualization



Thread graph



OP-code bigram



Graph of basic blocks

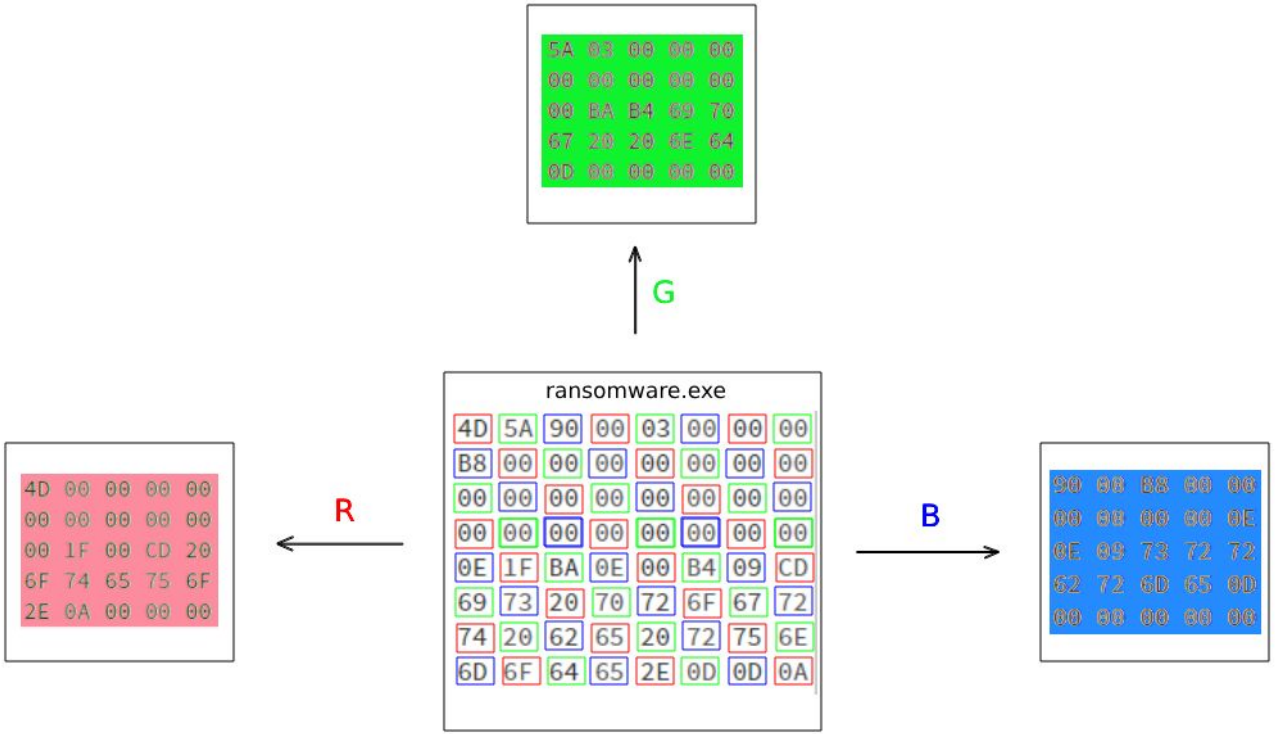
Bitmap image

R

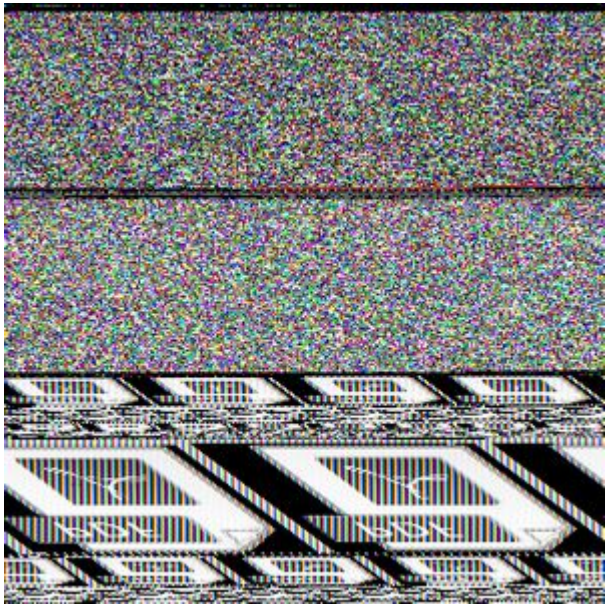
G

B

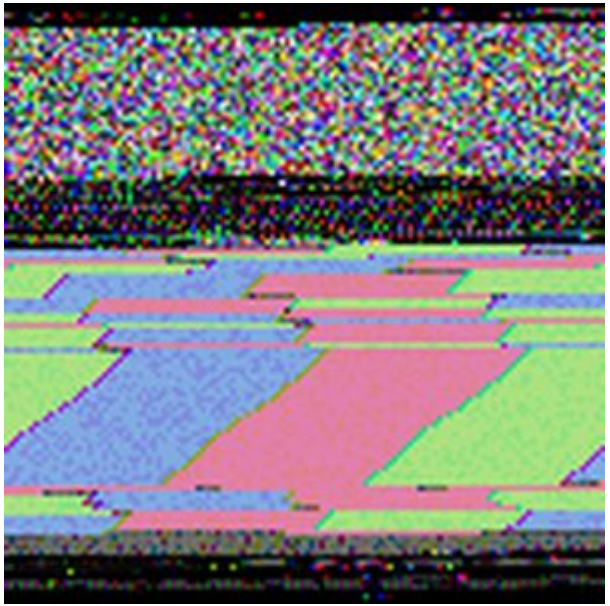
Byteplot visualization



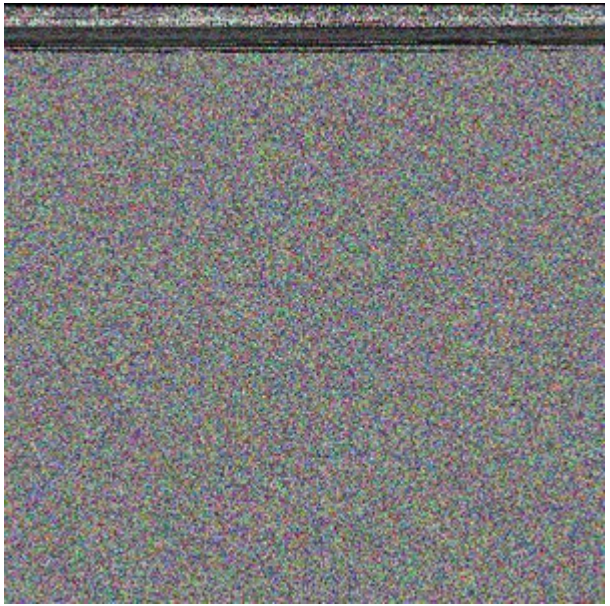
Byteplot visualization



Ransomware



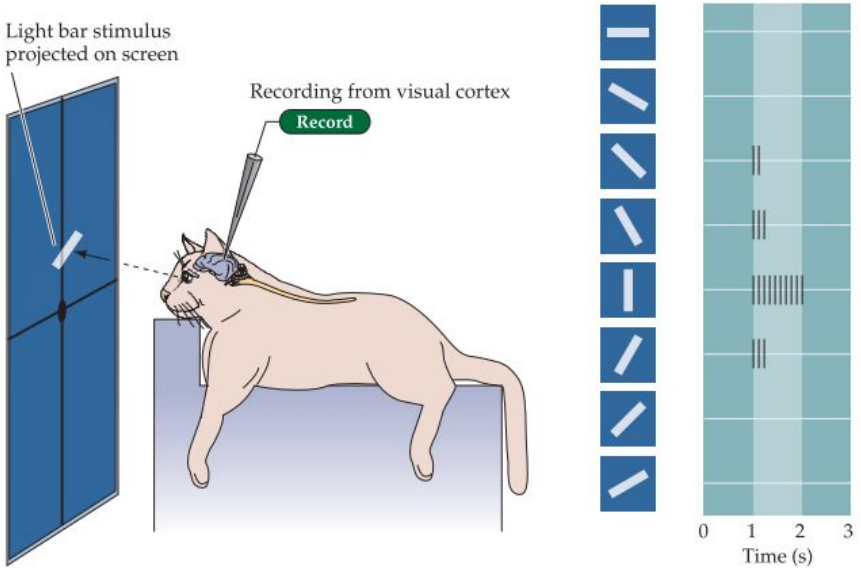
Spyware



Spyware

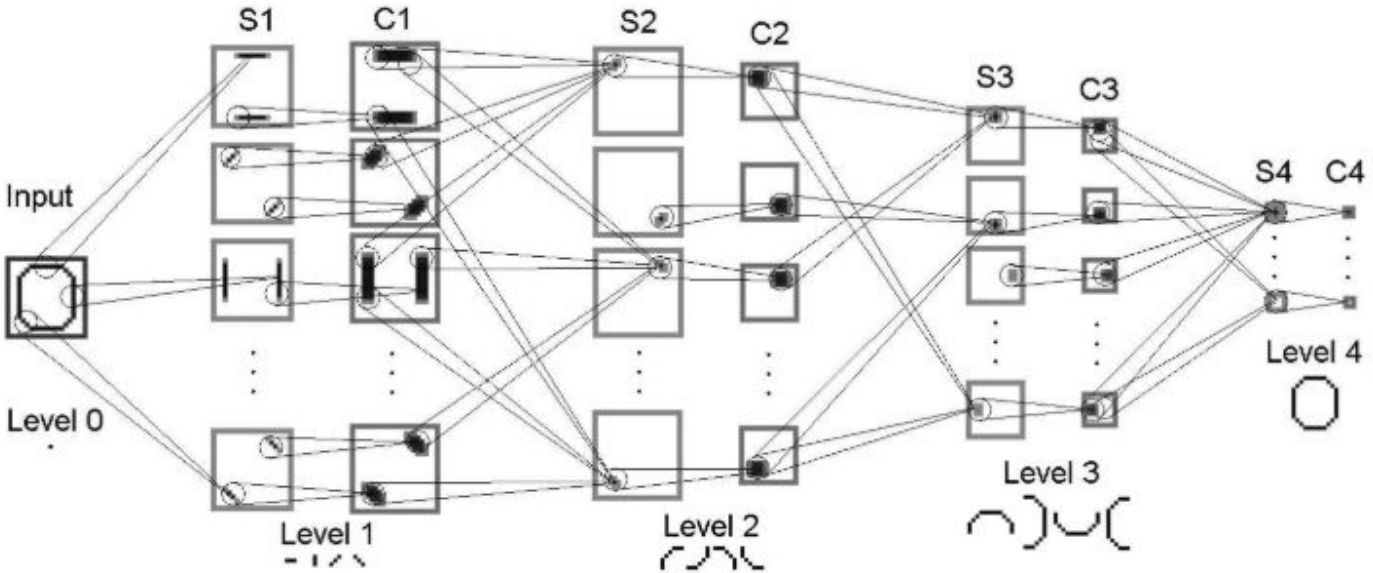
The history of convolutional neural networks

David Hubel's and Torsten Wiesel's cat experiment



Neocognitron

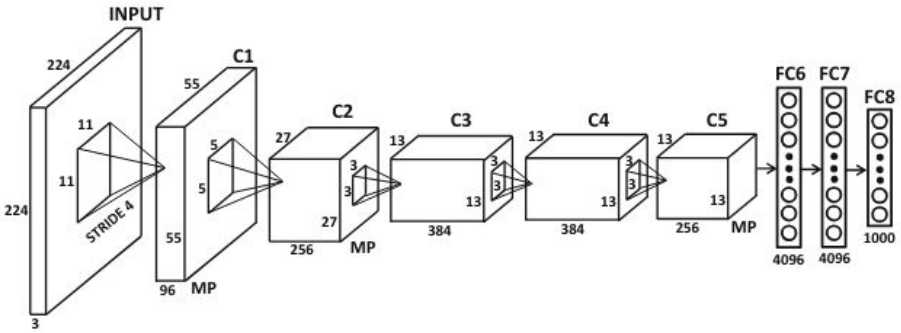
Proposed by Kunihiko Fukisima in 1979 and was used for Japanese handwritten character recognition



Convolutional Neural Networks

Layers:

- Convolution
- Pooling
- Activation
- Dense



Convolution

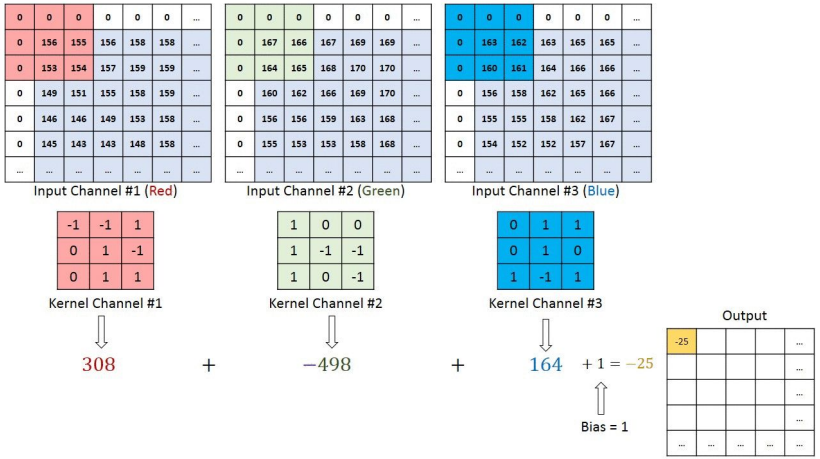
- Convolution is a binary operation, that combines two tensors

$$f_{i,j}^{(k)} = \sum_{d=1}^{D} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} x_{i+u,j+v,d} \cdot k_{u,v,d}^{(k)}$$

where:

  - D is the depth of the input.
  - m and n are the height and width of the kernel.
  - k indexes the kernel.

- Kernel slides over the image, performing element-wise multiplication and summation to produce a result
- The resulting tensors are referred to as feature maps



Convolution

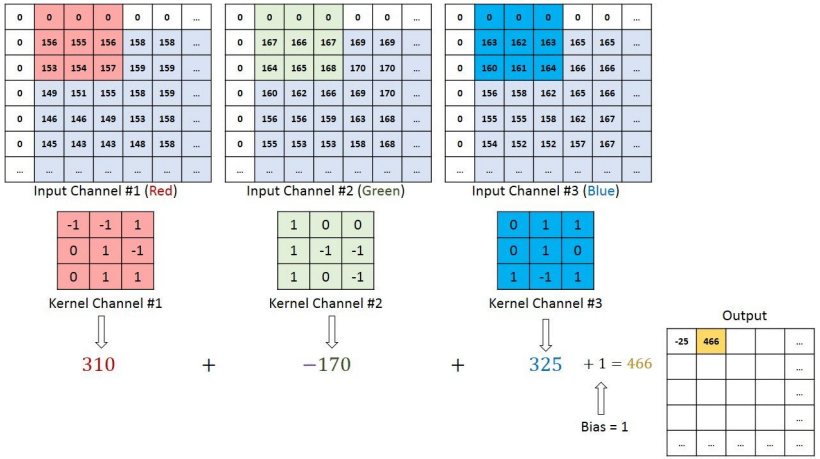
- Convolution is a binary operation, that combines two tensors

$$f_{i,j}^{(k)} = \sum_{d=1}^{D} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} x_{i+u,j+v,d} \cdot k_{u,v,d}^{(k)}$$

where:

  - D is the depth of the input.
  - m and n are the height and width of the kernel.
  - k indexes the kernel.

- Kernel slides over the image, performing element-wise multiplication and summation to produce a result
- The resulting tensors are referred to as feature maps



Convolution

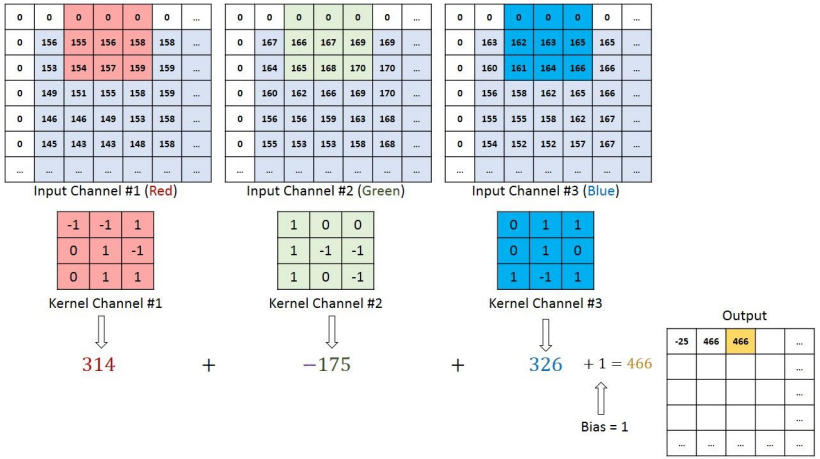
- Convolution is a binary operation, that combines two tensors

$$f_{i,j}^{(k)} = \sum_{d=1}^{D} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} x_{i+u,j+v,d} \cdot k_{u,v,d}^{(k)}$$

where:

  - D is the depth of the input.
  - m and n are the height and width of the kernel.
  - k indexes the kernel.

- Kernel slides over the image, performing element-wise multiplication and summation to produce a result
- The resulting tensors are referred to as feature maps



Convolution

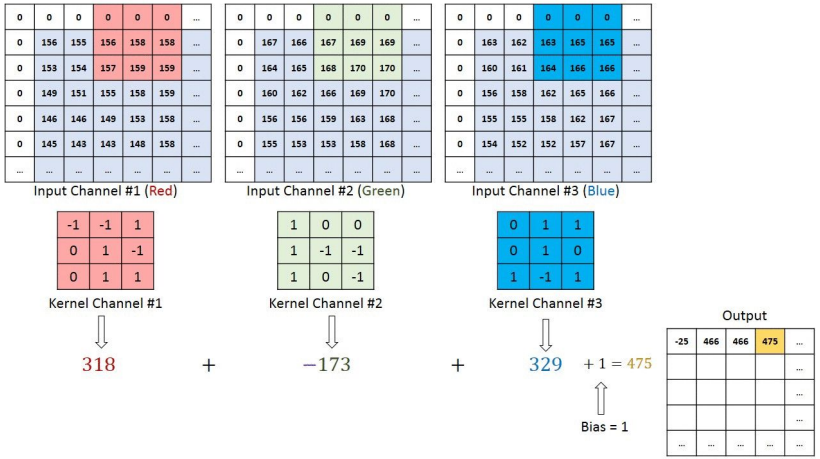
- Convolution is a binary operation, that combines two tensors

$$f_{i,j}^{(k)} = \sum_{d=1}^{D} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} x_{i+u,j+v,d} \cdot k_{u,v,d}^{(k)}$$

where:

  - D is the depth of the input.
  - m and n are the height and width of the kernel.
  - k indexes the kernel.

- Kernel slides over the image, performing element-wise multiplication and summation to produce a result
- The resulting tensors are referred to as feature maps



Convolution

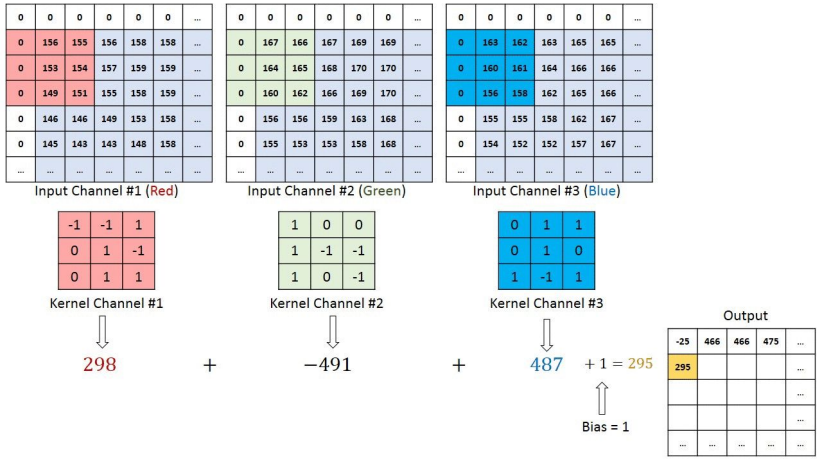
- Convolution is a binary operation, that combines two tensors

$$f_{i,j}^{(k)} = \sum_{d=1}^{D} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} x_{i+u,j+v,d} \cdot k_{u,v,d}^{(k)}$$

where:

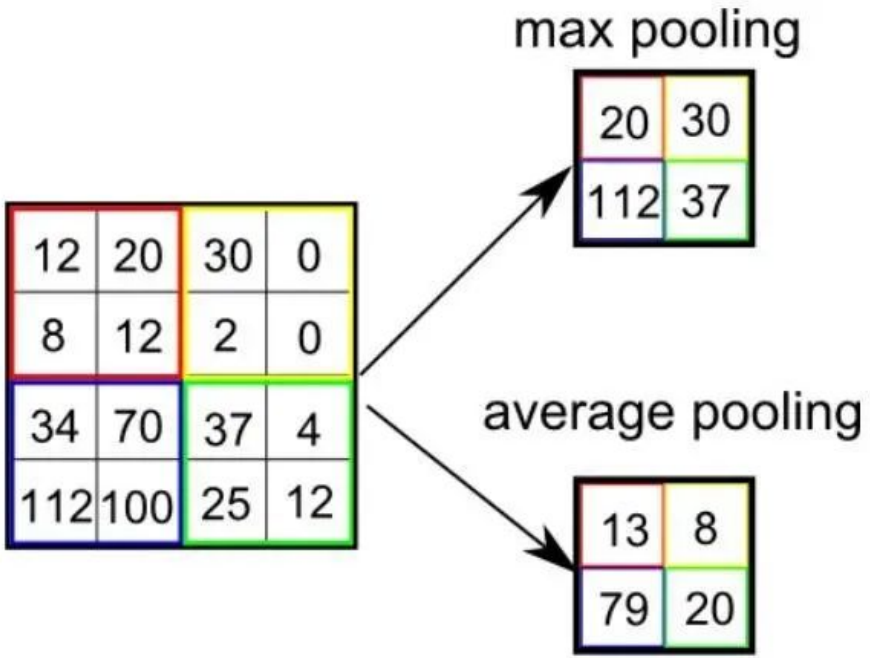
  - D is the depth of the input.
  - m and n are the height and width of the kernel.
  - k indexes the kernel.

- Kernel slides over the image, performing element-wise multiplication and summation to produce a result
- The resulting tensors are referred to as feature maps



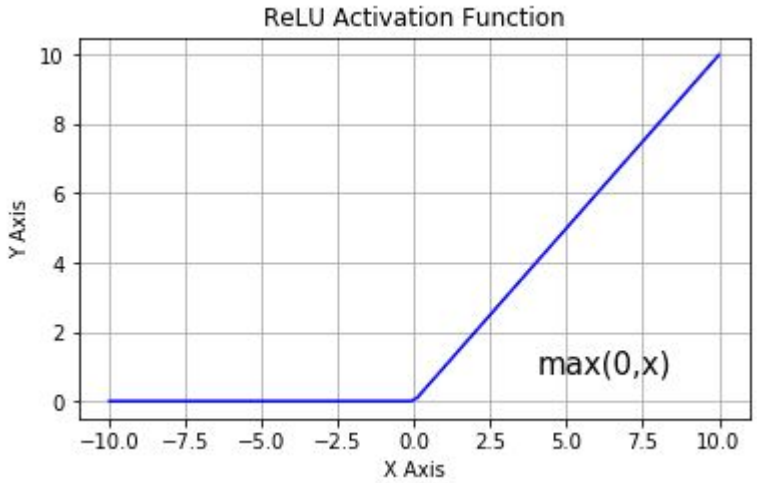
Pooling

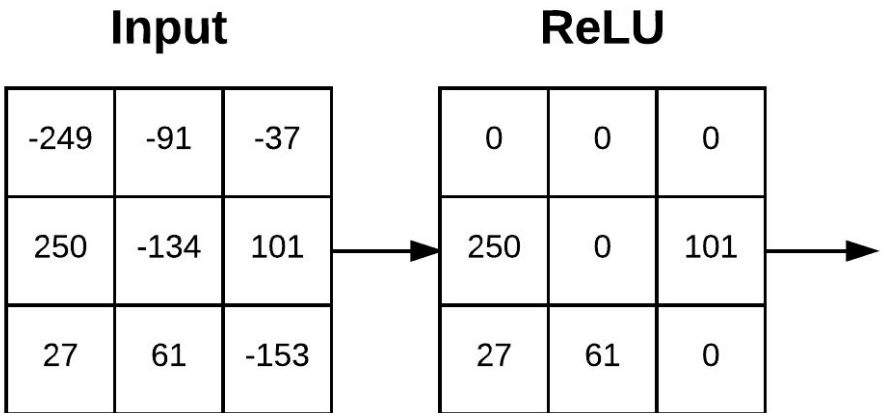
- Yields translation invariance
- Reduces feature-map dimensions
- Average vs Max pooling



Activation

- Defines, which neurons contribute to the result the most
- Allows to capture non-linearity of the data
- ReLU is the most popular choice

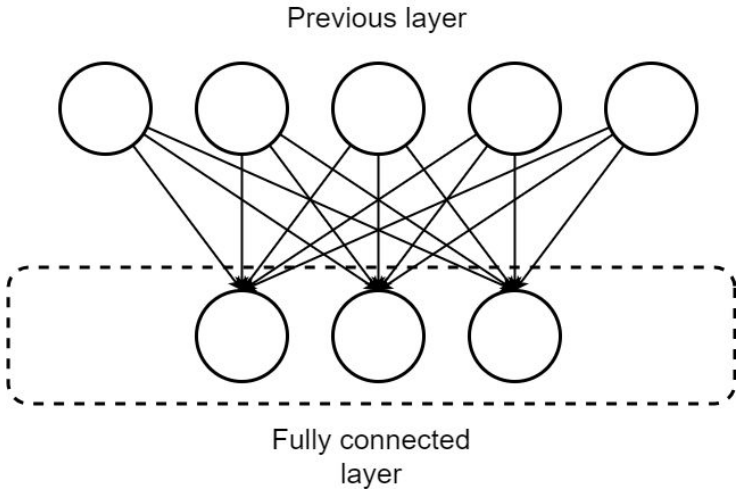




Fully Connected (Dense) Layer

- Layer with the highest number of connections
- Defines the number of result classes
- Softmax activation

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for j} = 1,\ldots,\text{K}$$

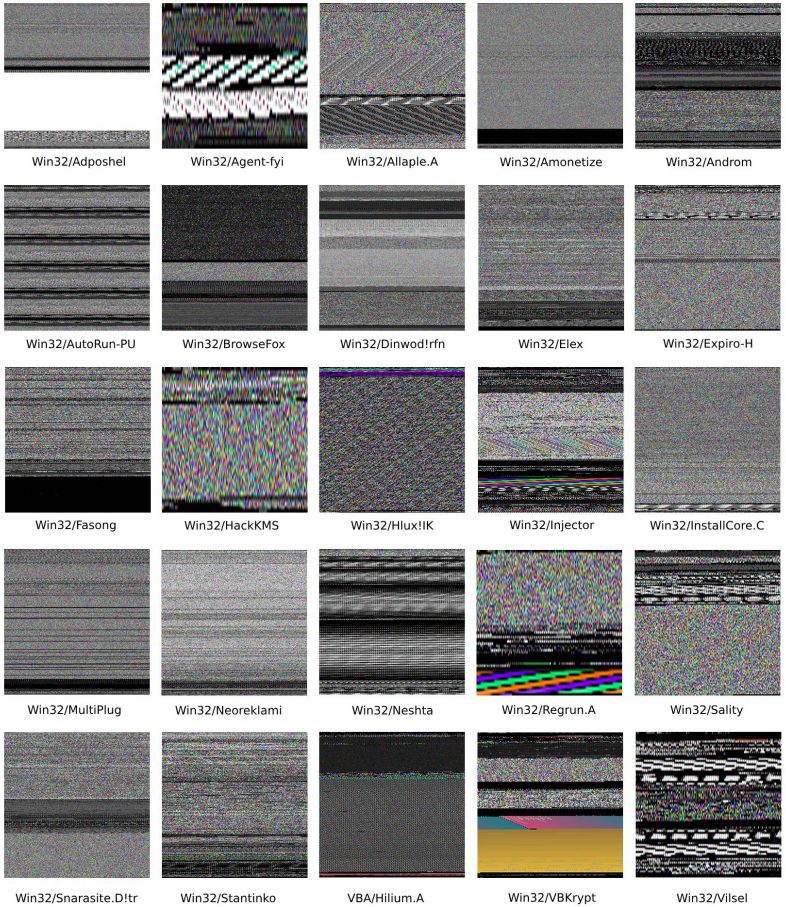


Combine two techniques

- CNN model: pre-trained MobileNetV2 + fine-tuning
- Two datasets: Malevis and SORRY
- # of training epochs: 20
- # of epochs for fine-tuning: 10

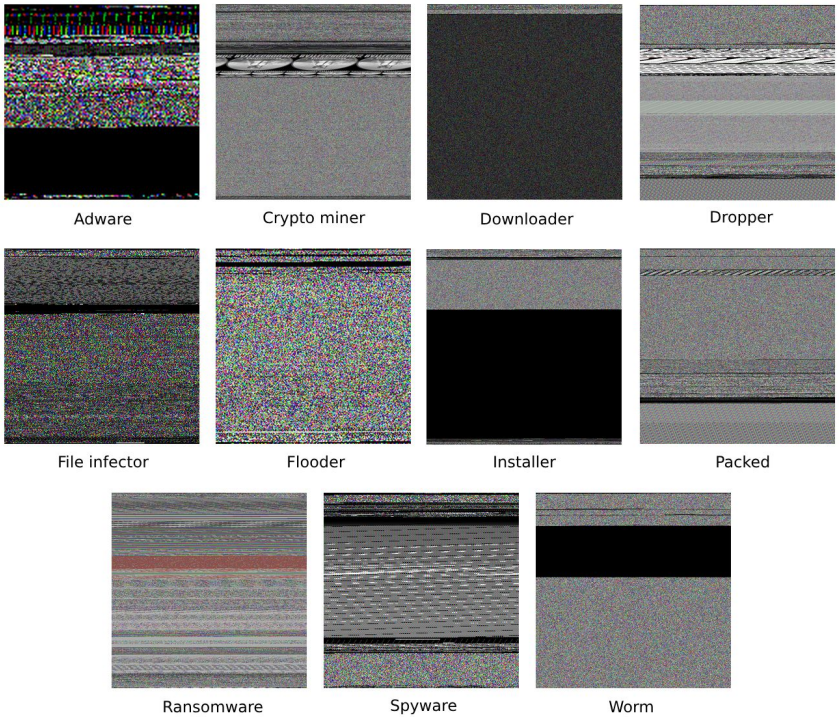
Malevis dataset

- Collected by the Multimedia Information Lab of Hacepette University Computer Engineering Department in cooperation with COMODO Inc.
- Contains RGB byte images for 25 malware classes and benign samples
- 9100 training and 5126 testing images



SORRY dataset

- Created as part of this master thesis
- The SOREL-20M dataset from Sophos was used as the basis for creating the SORRY dataset
- Contains RGB byte images for 11 malware classes and benign samples
- 8200 training and 5350 testing images



Experiment #1: MobileNetV2 + Malevis MCC (with Other)

Description: original Malevis dataset, including Other folder

Classes: **Adposhel**, **Agent, Allaple, Amonetize, Androm, Autorun, BrowseFox, Dinwod, Elex, Expiro, Fasong, HackKMS, Hlux, Injector, InstallCore, MultiPlug, Neoreklami, Neshta, Other (Benign), Regrun, Sality, Snarasite, Stantinko, VBA, VBKrypt, Vilsel**

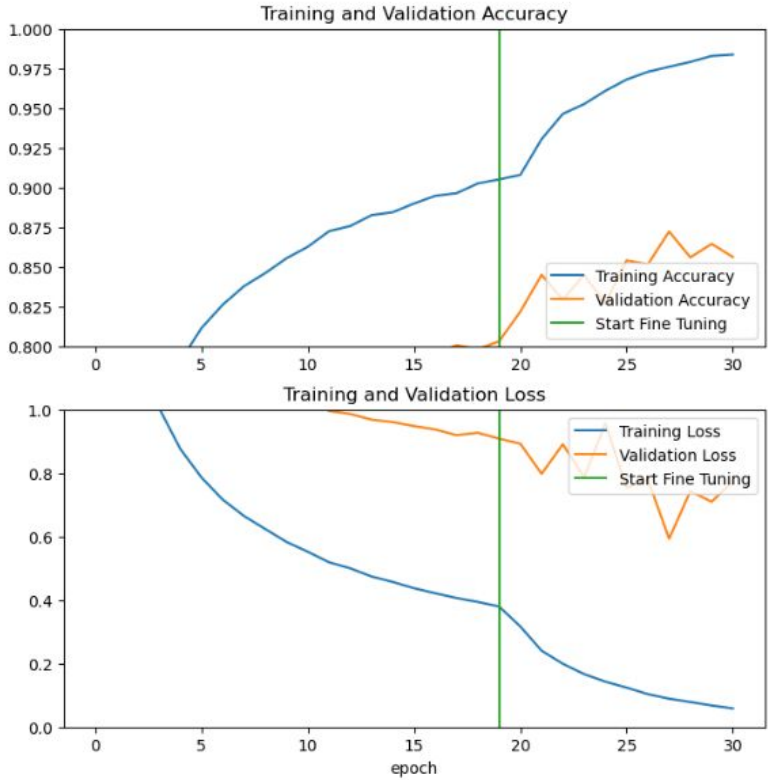
Training dataset: 9100 samples

Validation dataset: 4102 samples

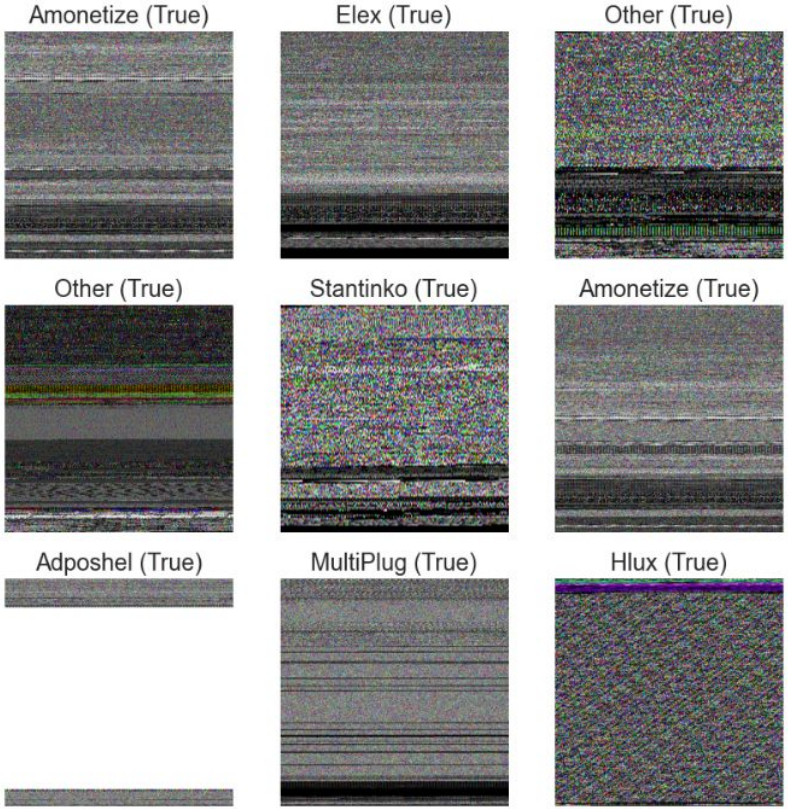
Testing dataset: 1024 samples

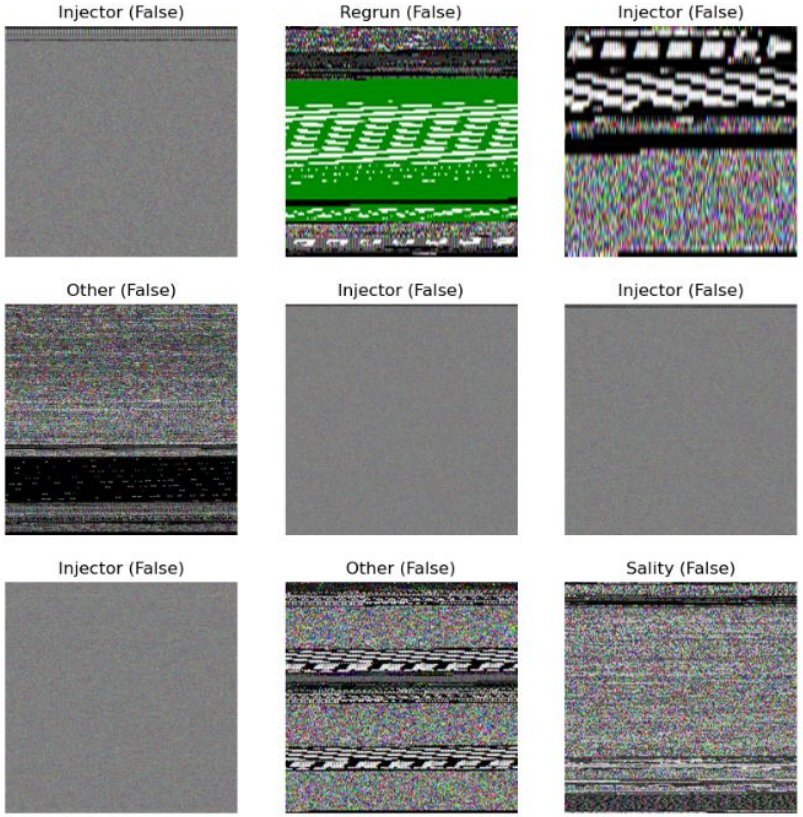
Initial accuracy: 0.02

	Accuracy	Precision	Recall	F1-score
Training	0.9195	0.9219	0.9195	0.9207
Validation	0.8040	0.8241	0.8936	0.8574
Testing	0.8486	0.8933	0.9389	0.9155



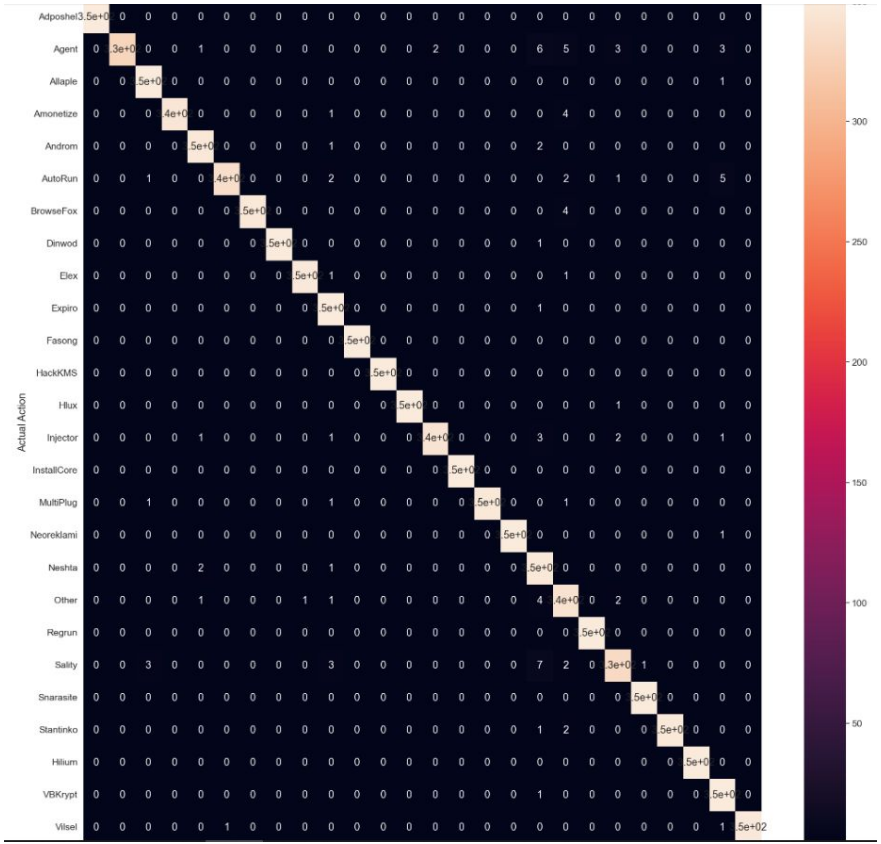
Experiment #1: MobileNetV2 + Malevis MCC (with Other)



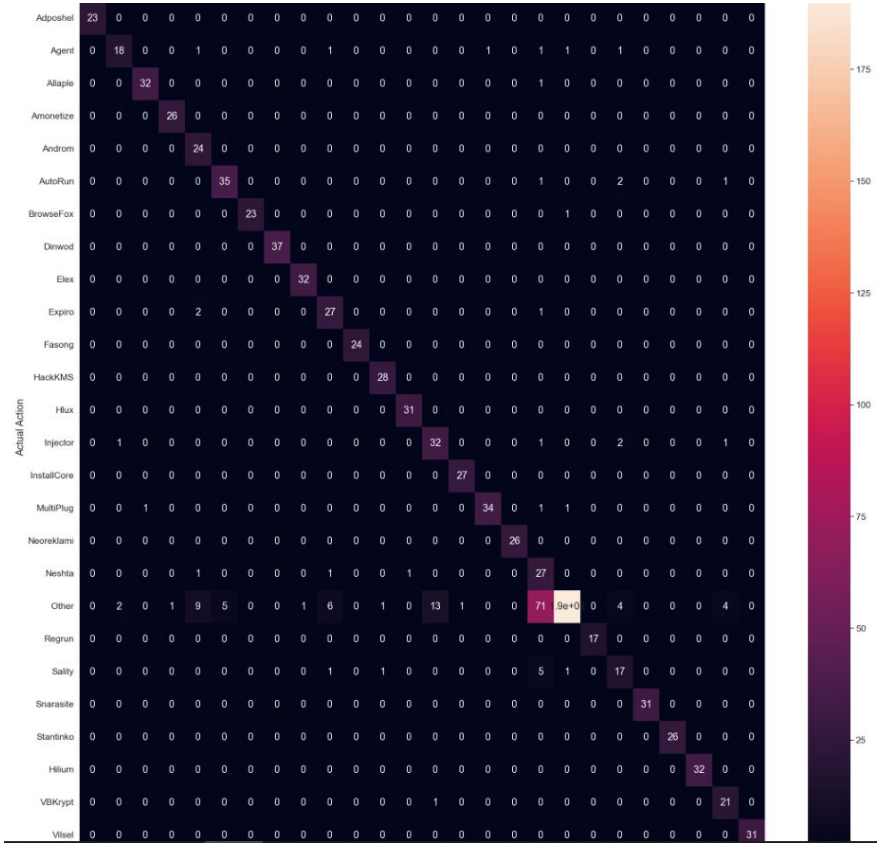


Experiment #1: MobileNetV2 + Malevis MCC (with Other)

Confusion matrix - training

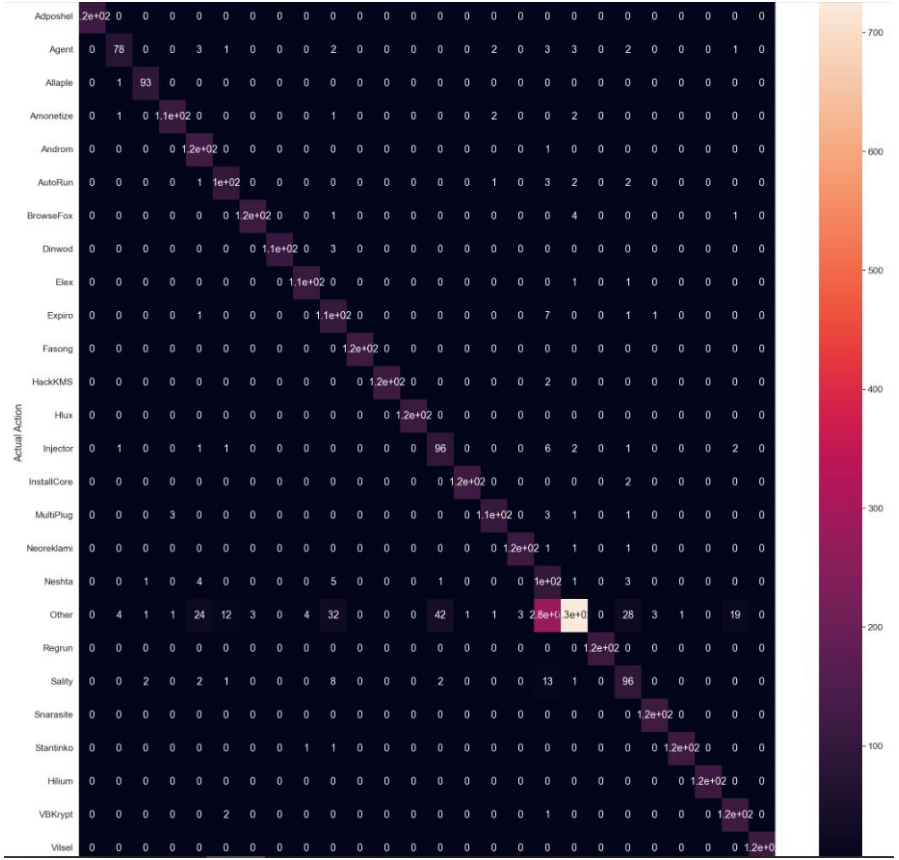


Confusion matrix - testing



Experiment #1: MobileNetV2 + Malevis MCC (with Other)

Confusion matrix - validation



Experiment #2: MobileNetV2 + Malevis BC + more benign

Description: original Malevis dataset + cca 3000 benign samples from SORRY dataset

Classes: **Benign**, **Malware**

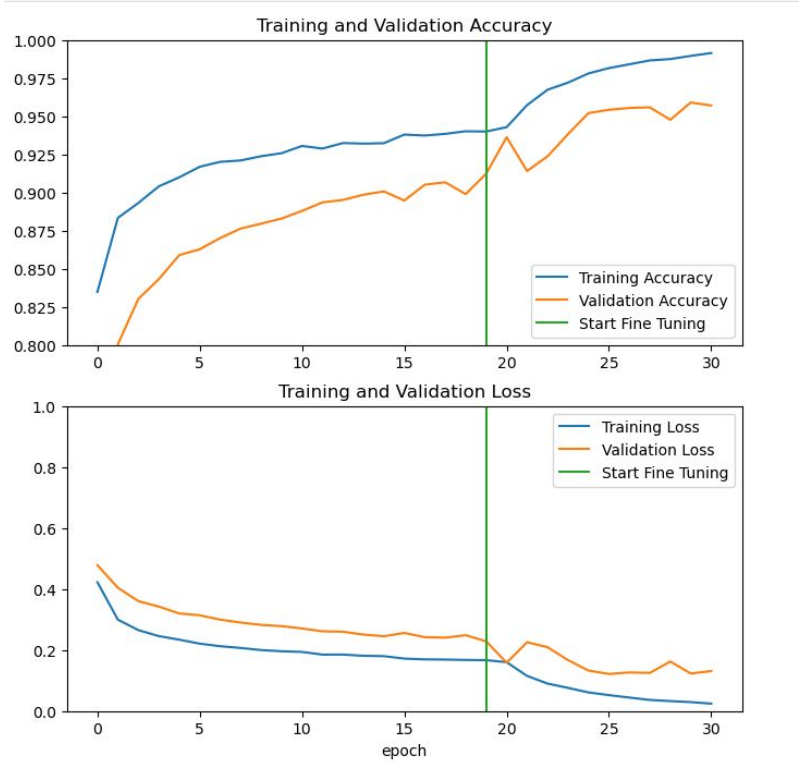
Training dataset: 2378 vs 8750

Validation dataset: 2078 vs 2928

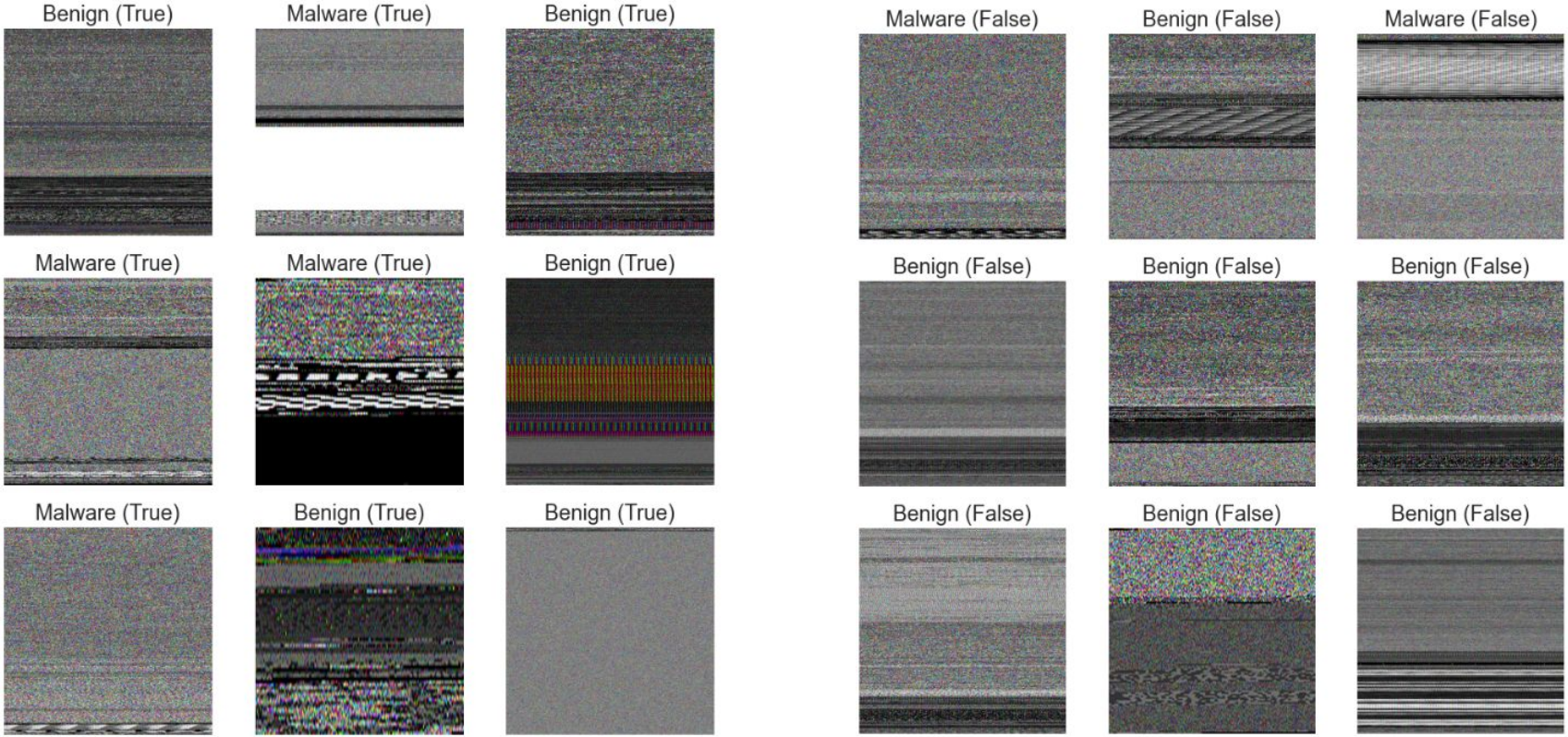
Testing dataset: 528 vs 720

Initial accuracy: 0.62

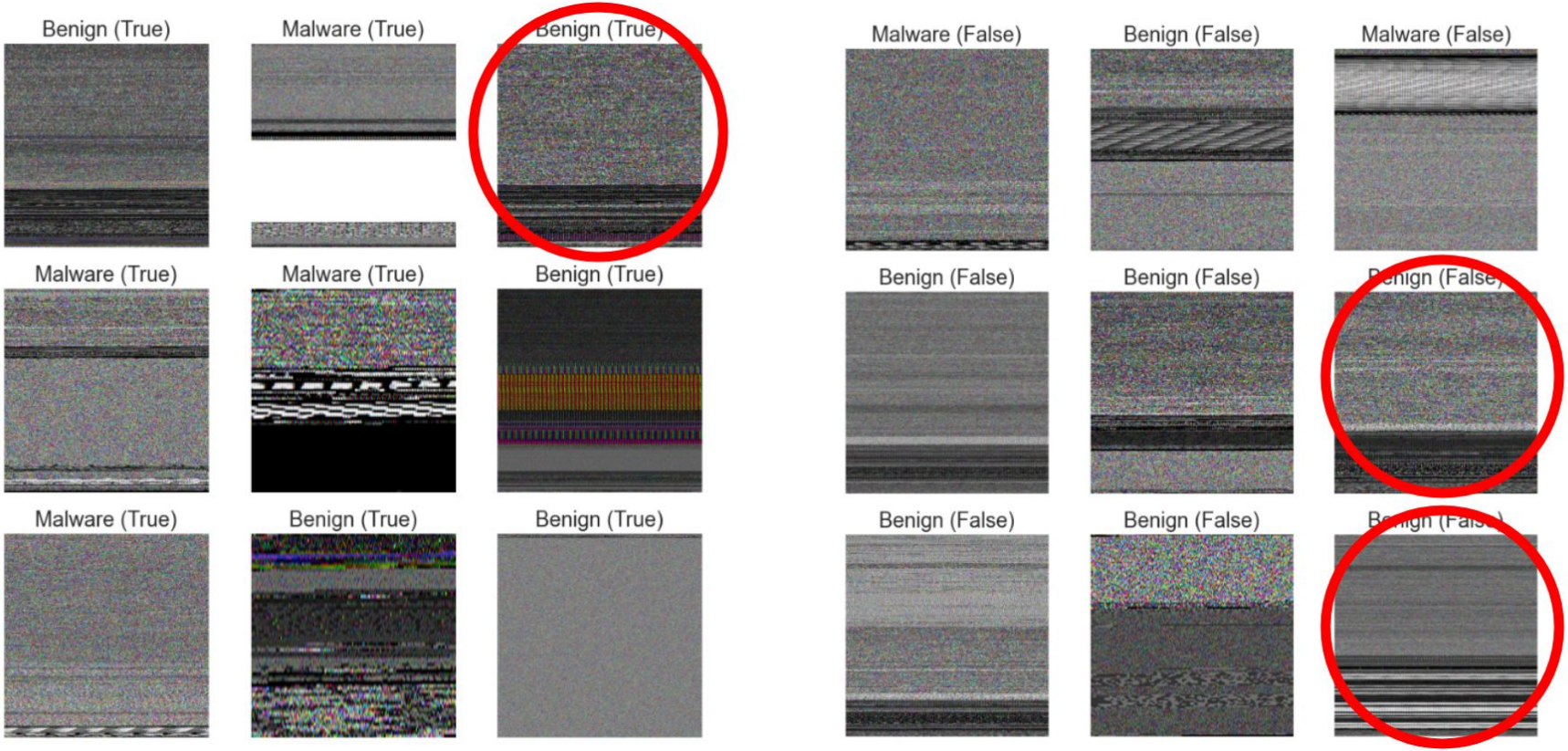
	Accuracy	Precision	Recall	F1-score
Training	0.9514	0.9508	0.9514	0.9511
Validation	0.9139	0.9201	0.9135	0.9168
Testing	0.9423	0.9468	0.9447	0.9457



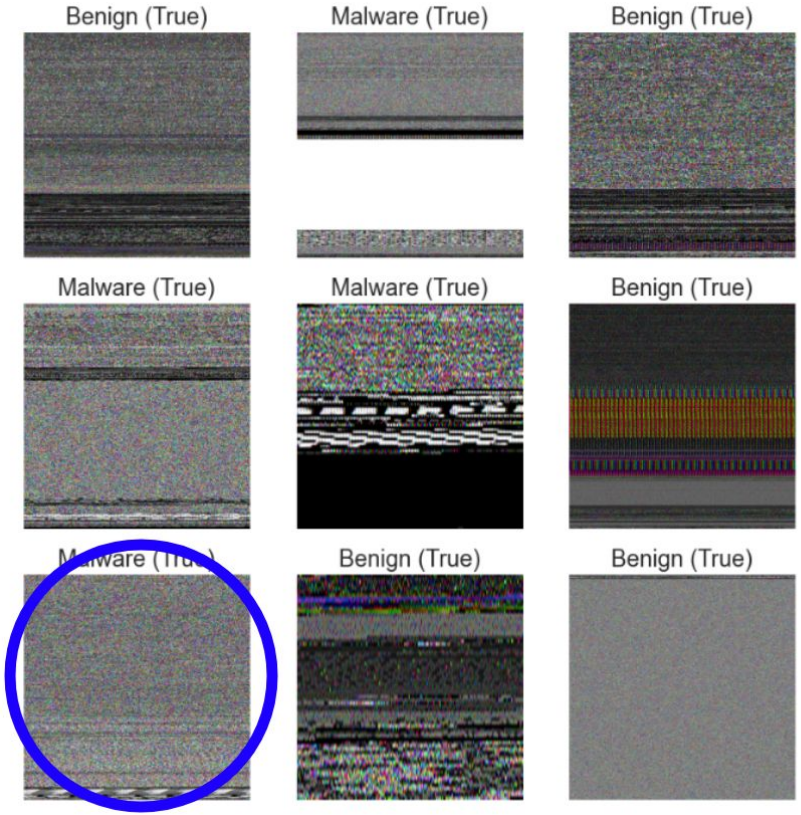
Experiment #2: MobileNetV2 + Malevis BC + more benign

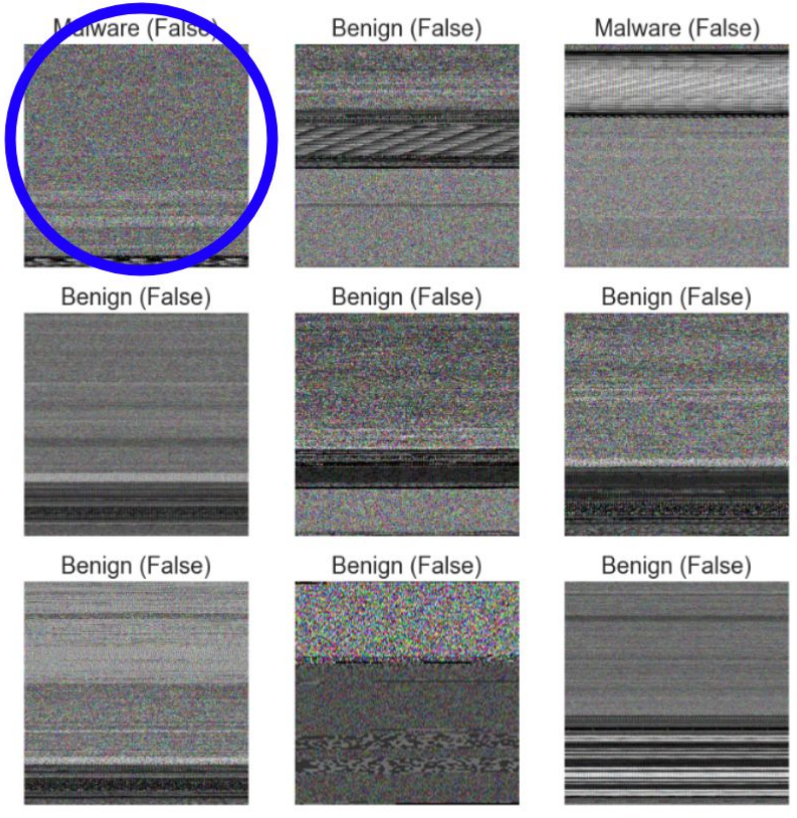


Experiment #2: MobileNetV2 + Malevis BC + more benign



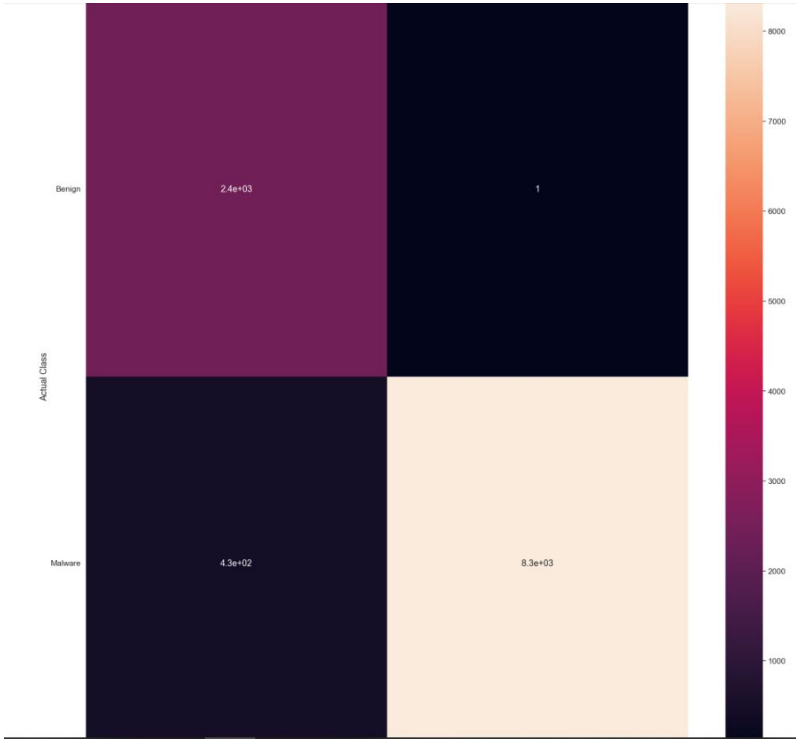
Experiment #2: MobileNetV2 + Malevis BC + more benign



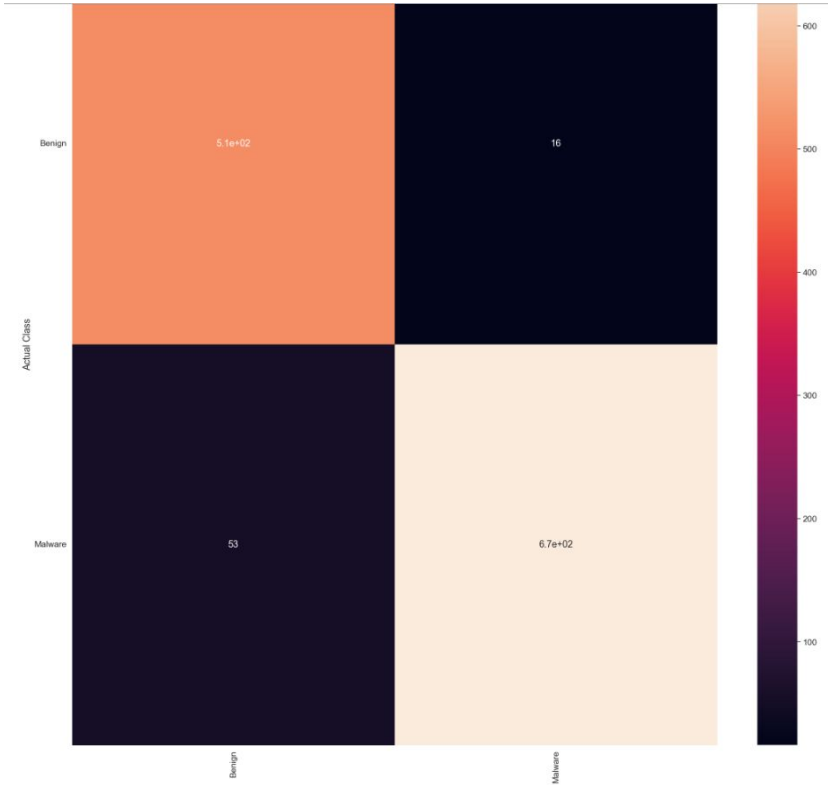


Experiment #2: MobileNetV2 + Malevis BC + more benign

Confusion matrix - training

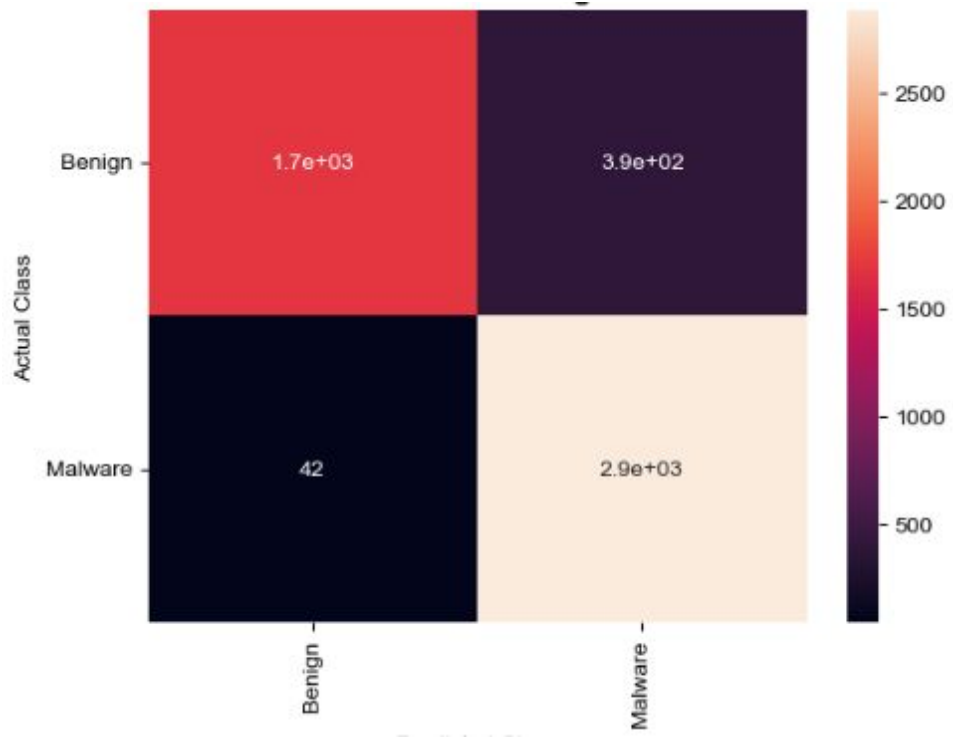


Confusion matrix - testing



Experiment #2: MobileNetV2 + Malevis BC + more benign

Confusion matrix - validation



Experiment #3: MobileNetV2 + SORRY BC

Description: SORRY dataset splitted into two classes

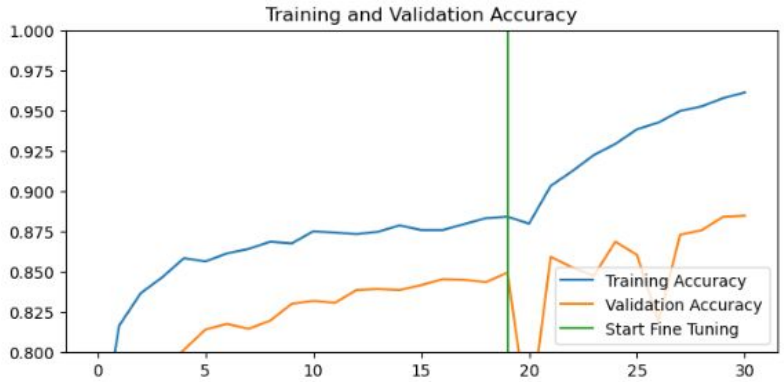
Classes: **Benign**, **Malware**

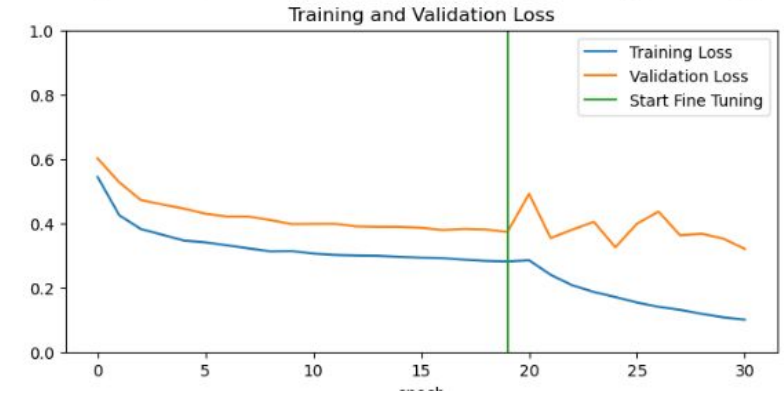
Training dataset: 2378 vs 7000

Validation dataset: 2097 vs 2797

Testing dataset: 511 vs 705

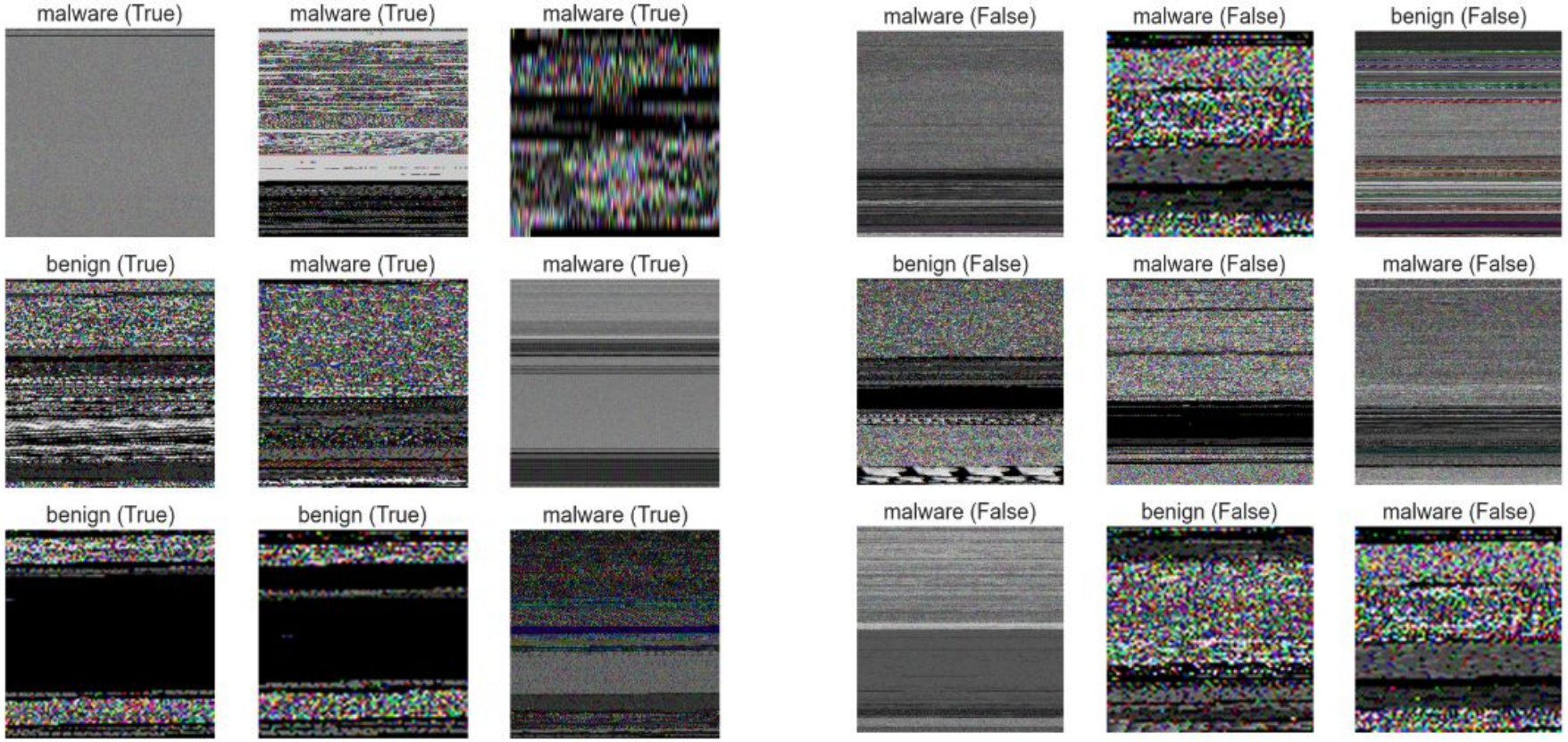
Initial accuracy: 0.49



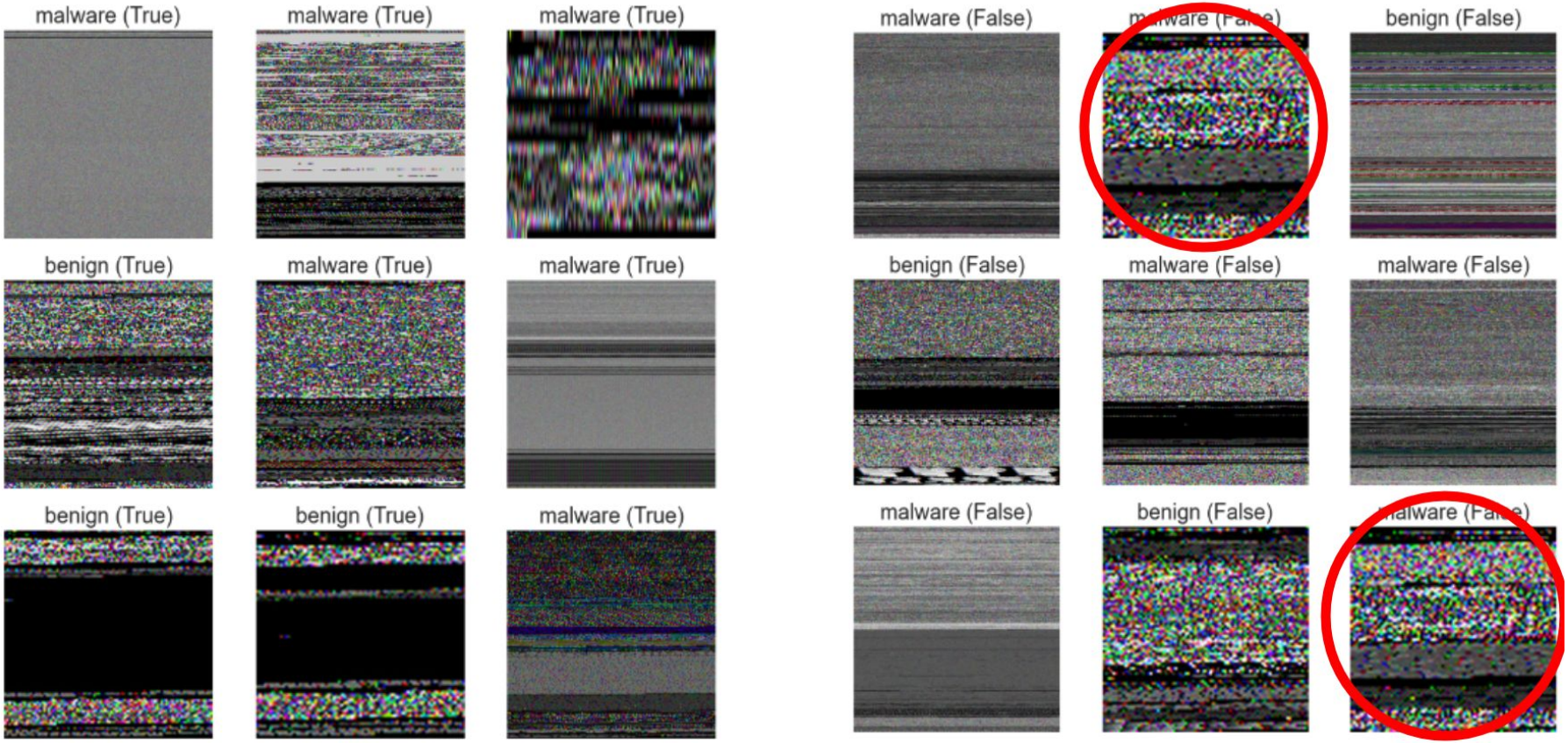


	Accuracy	Precision	Recall	F1-score
Training	0.9576	0.9055	0.9645	0.9341
Validation	0.8860	0.8784	0.8831	0.8808
Testing	0.8865	0.8804	0.8855	0.8830

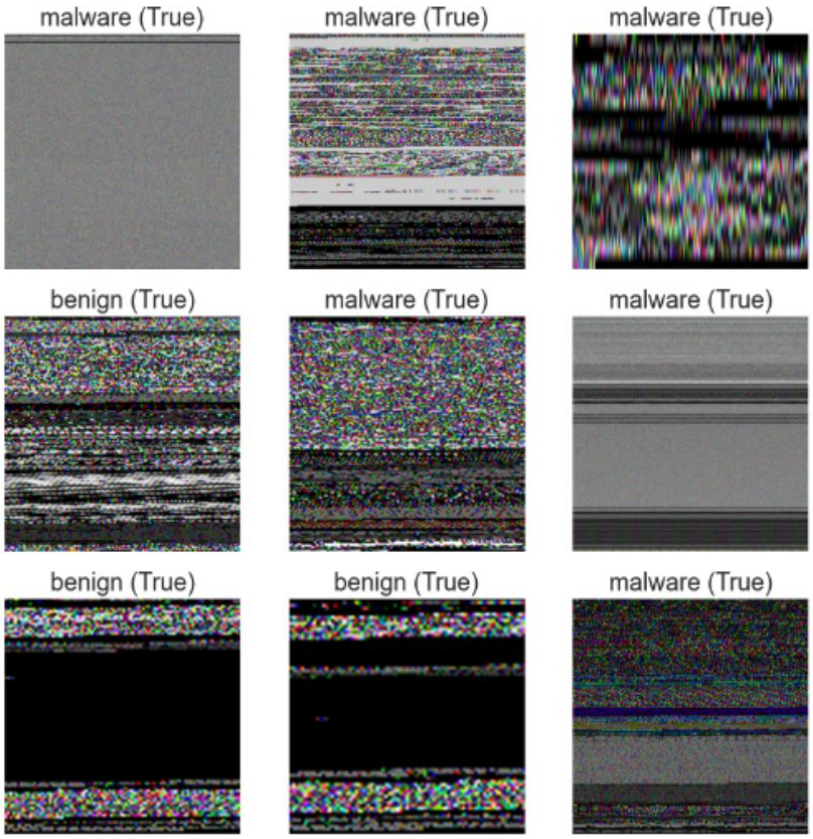
Experiment #3: MobileNetV2 + SORRY BC

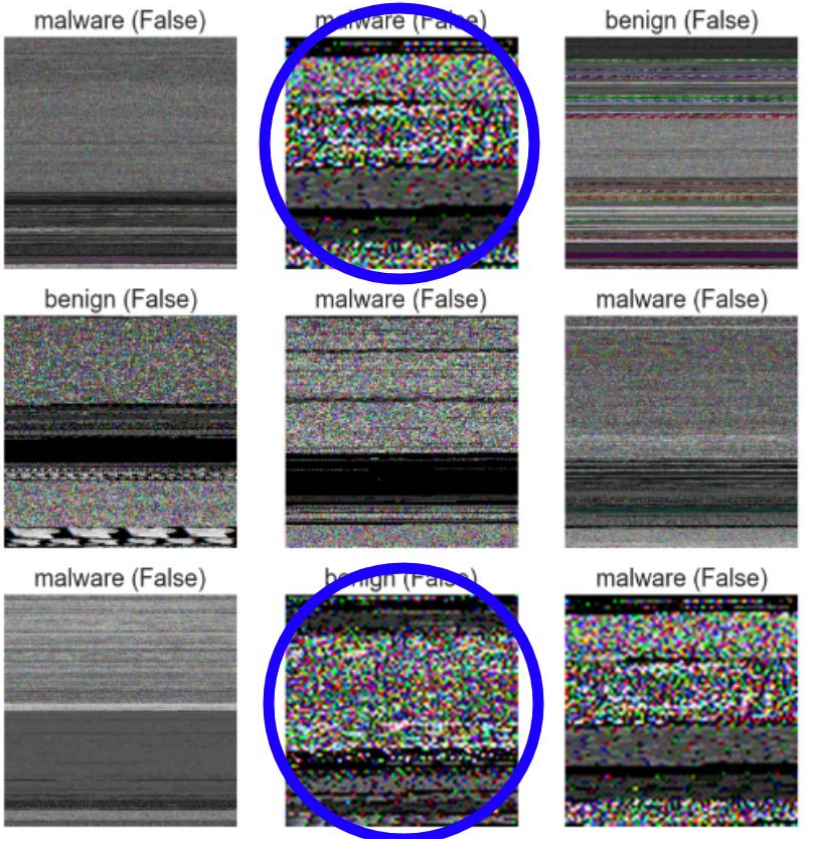


Experiment #3: MobileNetV2 + SORRY BC



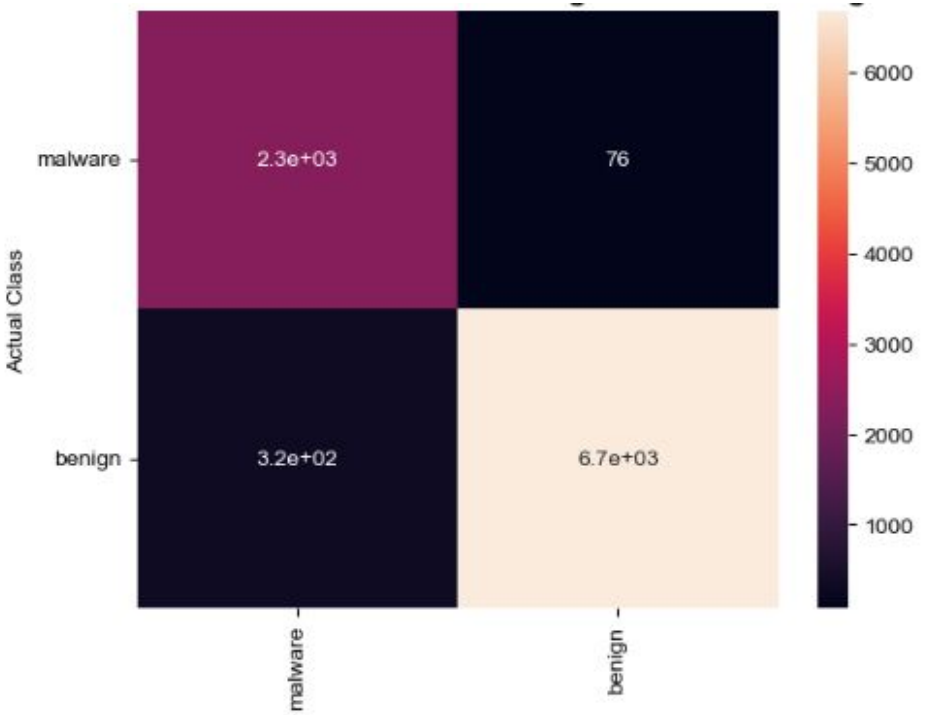
Experiment #3: MobileNetV2 + SORRY BC



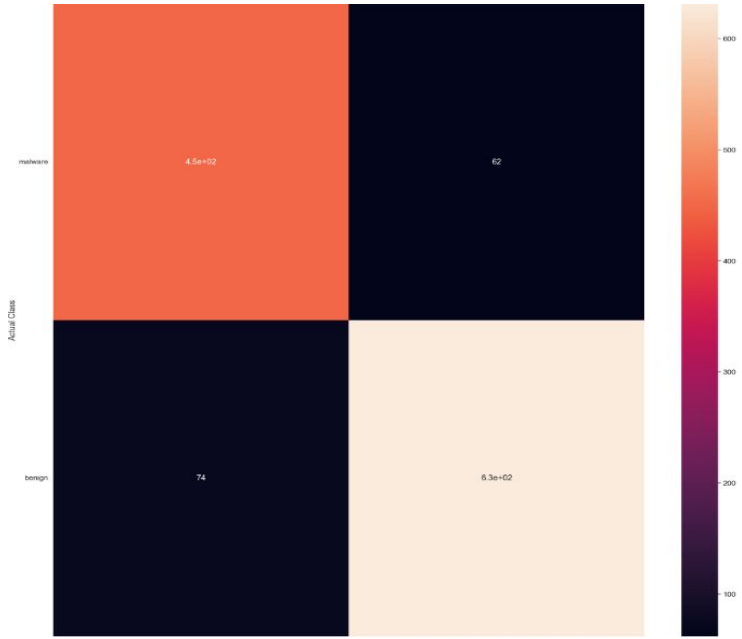


Experiment #3: MobileNetV2 + SORRY BC

Confusion matrix - training

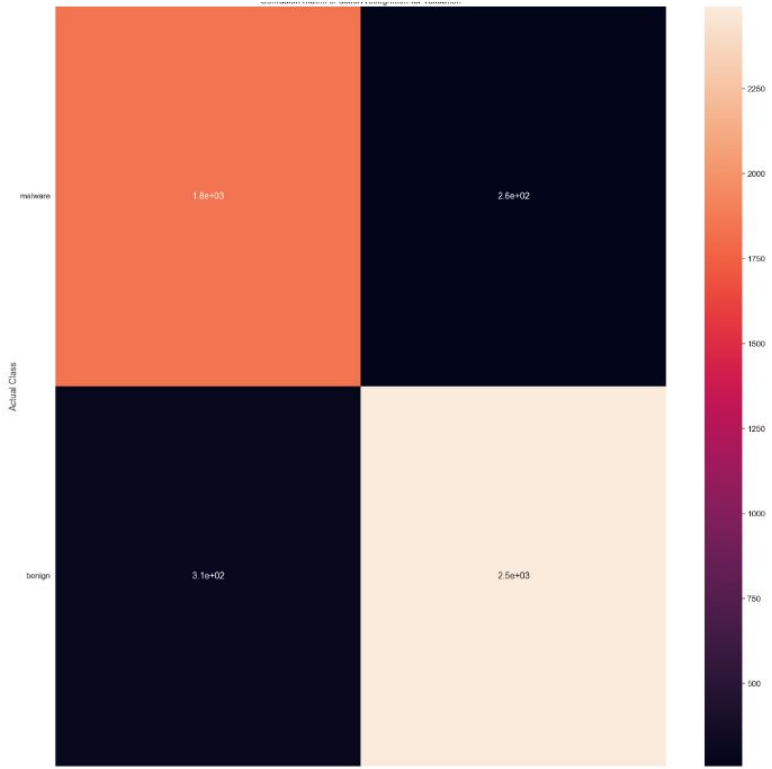


Confusion matrix - testing

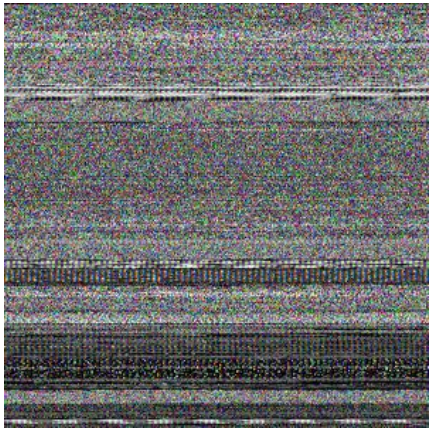


Experiment #3: MobileNetV2 + SORRY BC

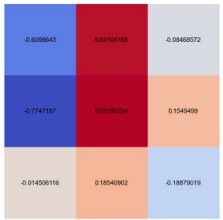
Confusion matrix - validation



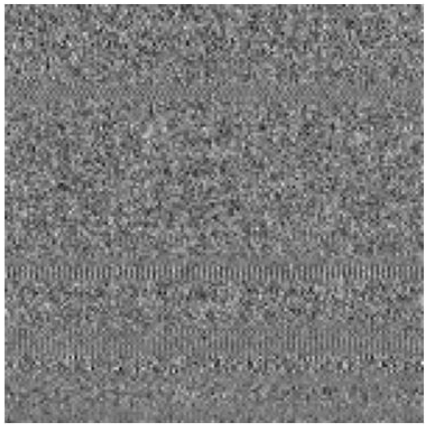
Learned filters examples #1



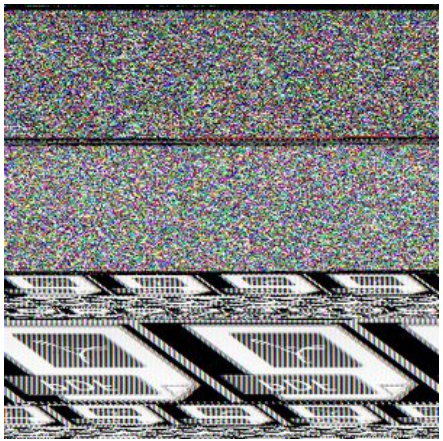
*



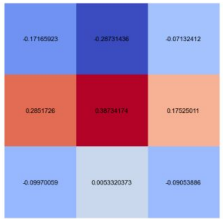
=

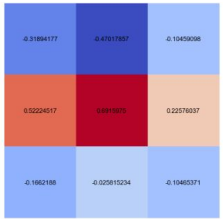


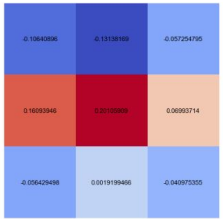
Learned filters examples #2



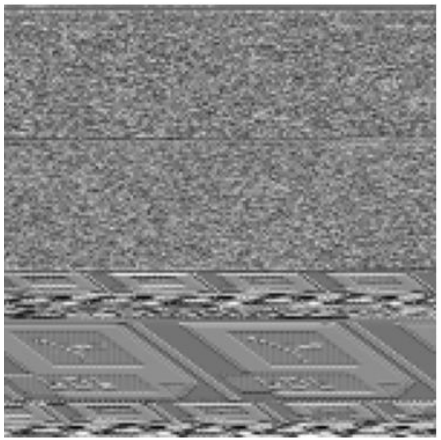
*







=



Experiment #4: RF + SORRY BC

Description: Extracted **EMBER** features from binaries used in the SORRY dataset

Classes: **Benign**, **Malware**

Training dataset: 2246 vs 3024

Testing dataset: 1538 vs 1976

Accuracy	Precision	Recall	F1-score
0.9999+	0.9999+	0.9999+	0.9999+

Experiment #4: RF + SORRY BC

ransomware.exe

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZÉ.....♦.  ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ╕.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00  ............°...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..▌..┤.═!╕.L═!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  F5 CD 13 12 B1 AC 7D 41 B1 AC 7D 41 B1 AC 7D 41  ⌠═..▒¼}A▒¼}A▒¼}A
00000090  05 30 8C 41 BD AC 7D 41 05 30 8E 41 3E AC 7D 41  .0îA╜¼}A.0ÄA>¼}A
000000A0  05 30 8F 41 AC AC 7D 41 D2 F1 7E 40 A7 AC 7D 41  .0ÅA¼¼}A╥±~@°¼}A
000000B0  D2 F1 78 40 8B AC 7D 41 D2 F1 79 40 93 AC 7D 41  ╥±x@ï¼}A╥±y@ô¼}A
000000C0  6C 53 B6 41 BA AC 7D 41 B1 AC 7C 41 D8 AC 7D 41  lS╢A│¼}A▒¼|A╪¼}A
000000D0  DF F1 74 40 B0 AC 7D 41 DF F1 82 41 B0 AC 7D 41  ▀±t@\¼}A▀±éA\¼}A
000000E0  DF F1 7F 40 B0 AC 7D 41 52 69 63 68 B1 AC 7D 41  ▀±⌂@\¼}ARich▒¼}A
000000F0  00 00 00 00 00 00 00 00 50 45 00 00 00 00 08 00  ........PE......
00000100  CE 20 71 58 00 00 00 00 00 00 00 00 E0 00 02 01  ╬ qX........α...
00000110  0B 01 0E 00 00 EC 01 00 00 32 01 00 00 00 00 00  .....∞...2......
00000120  52 56 00 00 00 10 00 00 00 00 02 00 00 00 40 00  RV............@.
00000130  00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00  ................
00000140  06 00 00 00 00 00 00 00 00 90 03 00 00 04 00 00  .........É......
00000150  00 00 00 00 00 00 40 81 00 00 10 00 00 10 00 00  ......@ü........
00000160  00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 5C DF 02 00 78 00 00 00  ........\▀..x...
00000180  00 80 03 00 DC 01 00 00 00 00 00 00 00 00 00 00  .C..▬...........
```

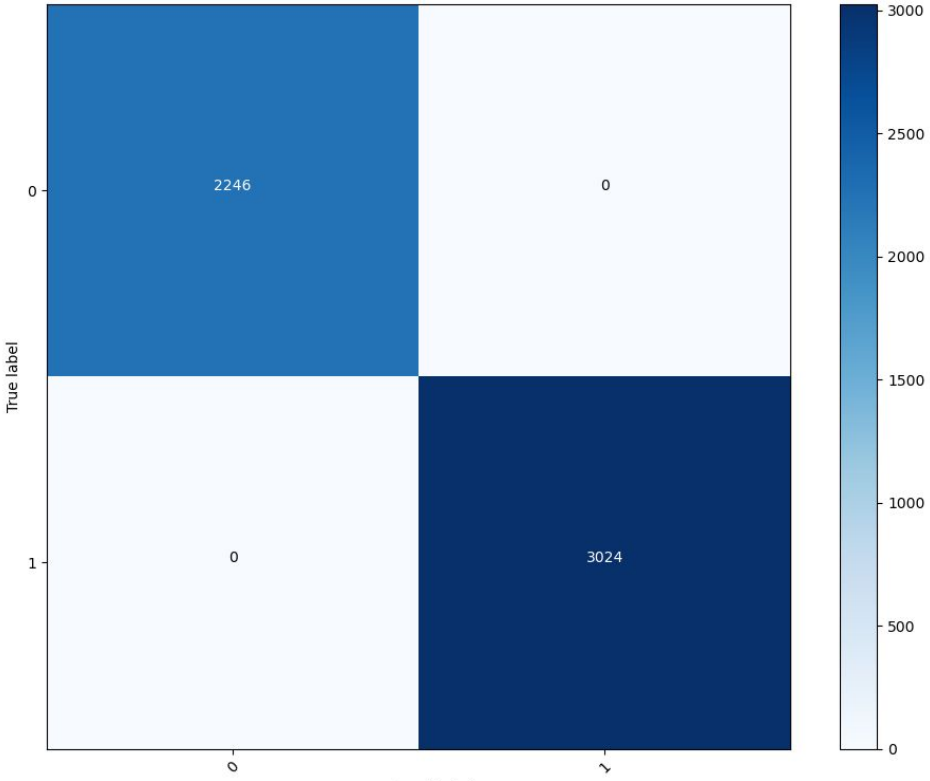

```
"general": {
  "file_size": 33334,
  "vsize": 45056,
  "has_debug": 0,
  "exports": 0,
  "imports": 41,
  "has_relocations": 1,
  "has_resources": 0,
  "has_signature": 0,
  "has_tls": 0,
  "symbols": 0
},
"header": {
  "coff": {
    "timestamp": 1365446976,
    "machine": "I386",
    "characteristics": [ "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": {
```



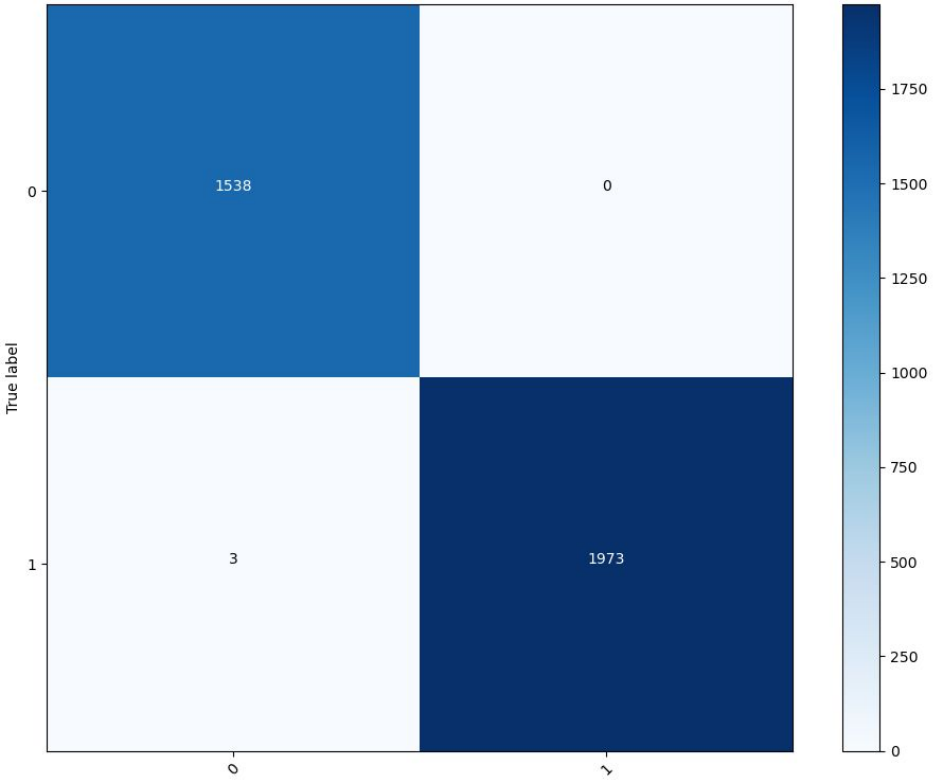
```
1  5.25543699041008949 3e-03,4.07524732872843742
2  2.37422380596399307 3e-02,4.28666360676288604
3  2.16538887470960617 1e-02,3.49039374850690364
4  2.35935356467962265 0e-02,3.93240945413708686
5  3.37147772312164306 6e-01,1.05743430105447776
6  2.43206527084112167 4e-02,4.75543458014726638
7  6.34814426302909851 1e-02,3.02734365686774253
8  2.43423692882061004 6e-02,3.06607829406857490
9  1.31217852234840393 1e-01,9.92983672767877578
10 2.37663136795163154 6e-03,5.42483106255531311
11 6.94573298096656799 3e-02,4.41568717360496521
12 3.70266258716583252 0e-01,1.12708173692226409
13 2.43206527084112167 4e-02,4.75543458014726638
```

Experiment #4: RF + SORRY BC

Confusion matrix - training

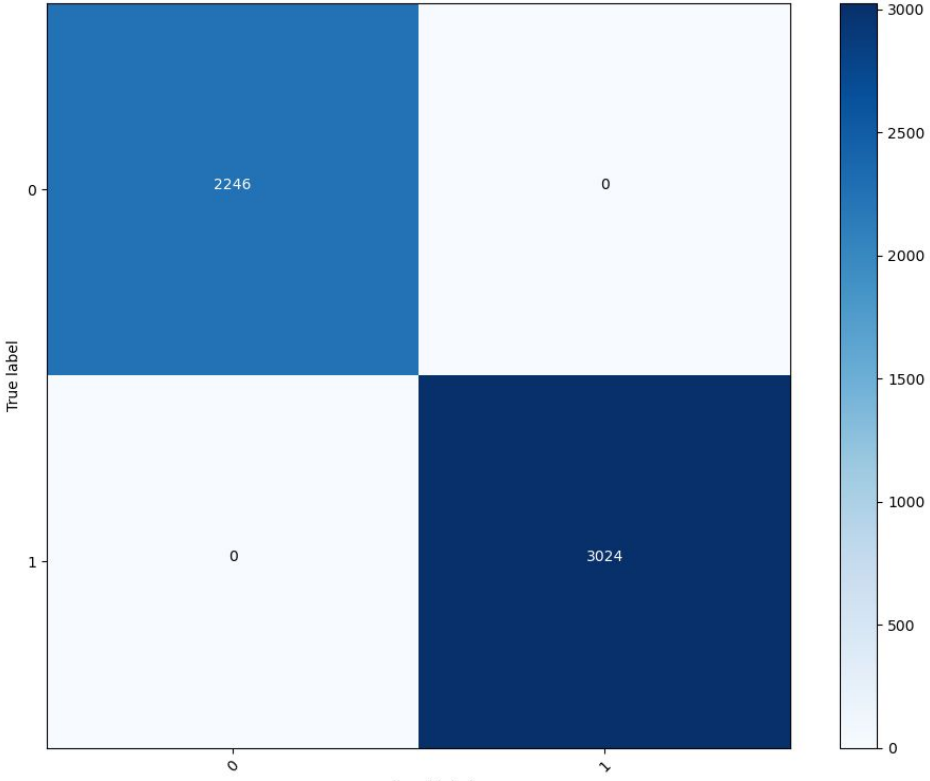


Confusion matrix - testing

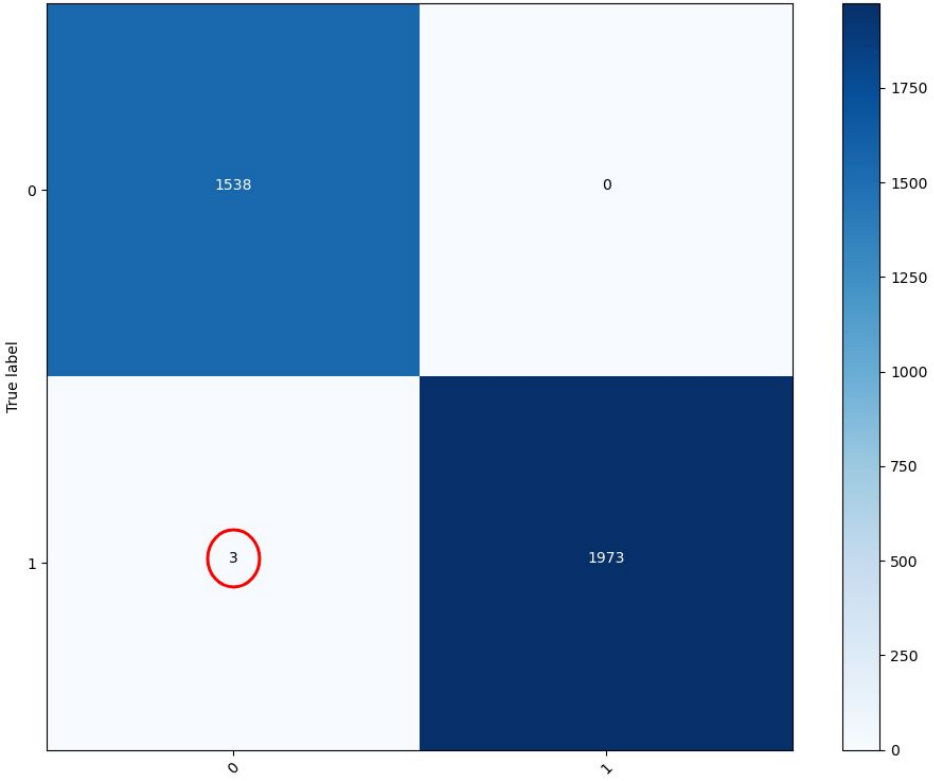


Experiment #4: RF + SORRY BC

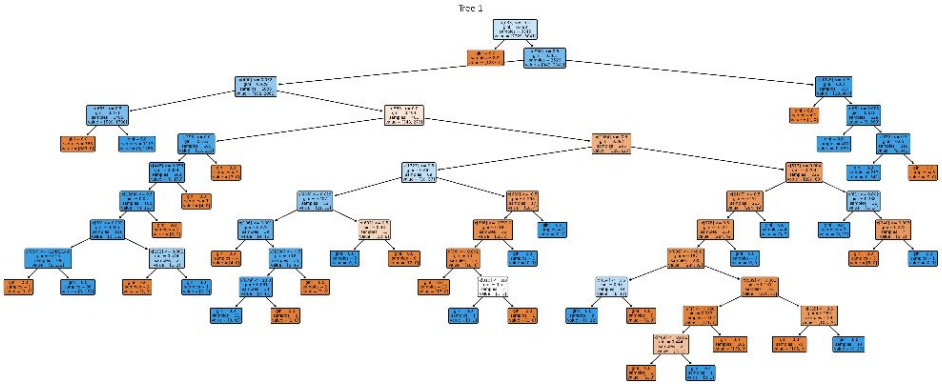
Confusion matrix - training

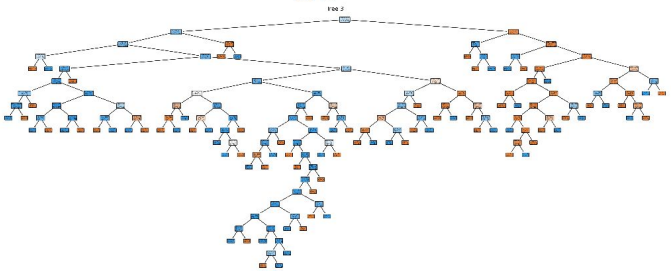


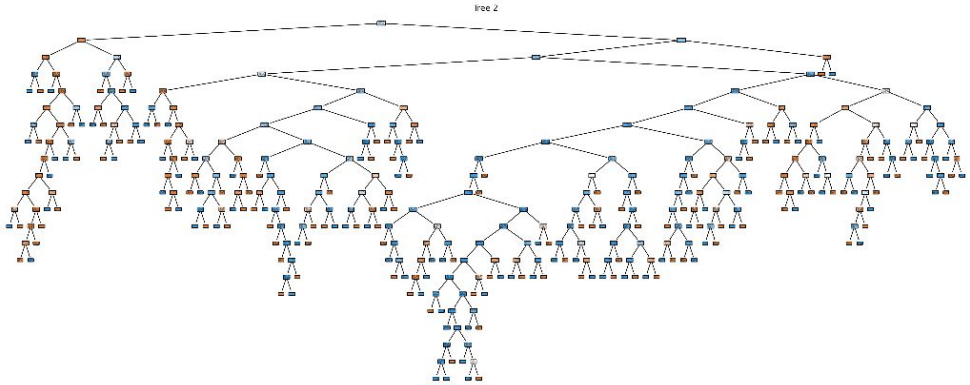
Confusion matrix - testing

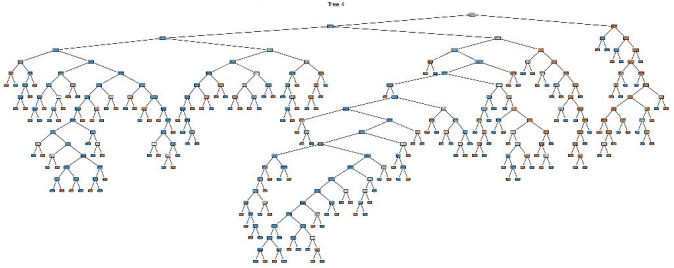


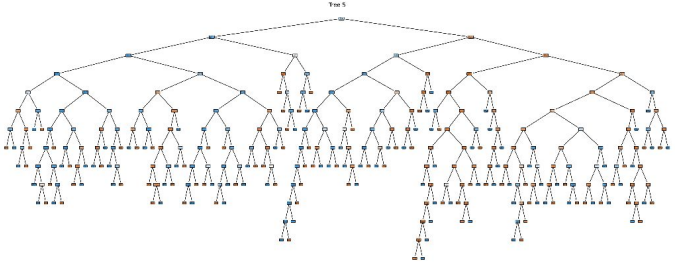
Experiment #4: RF + SORRY BC



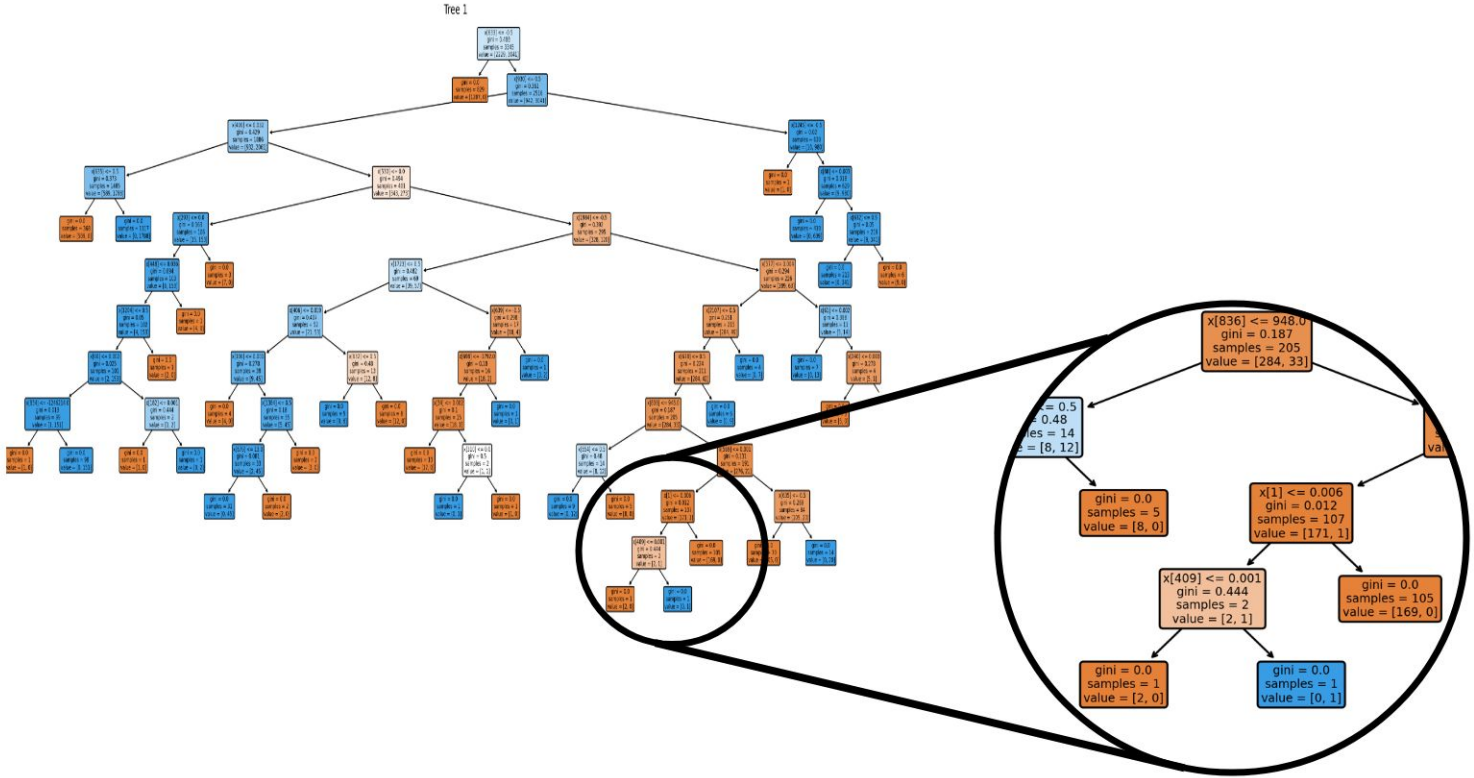








Experiment #4: RF + SORRY BC

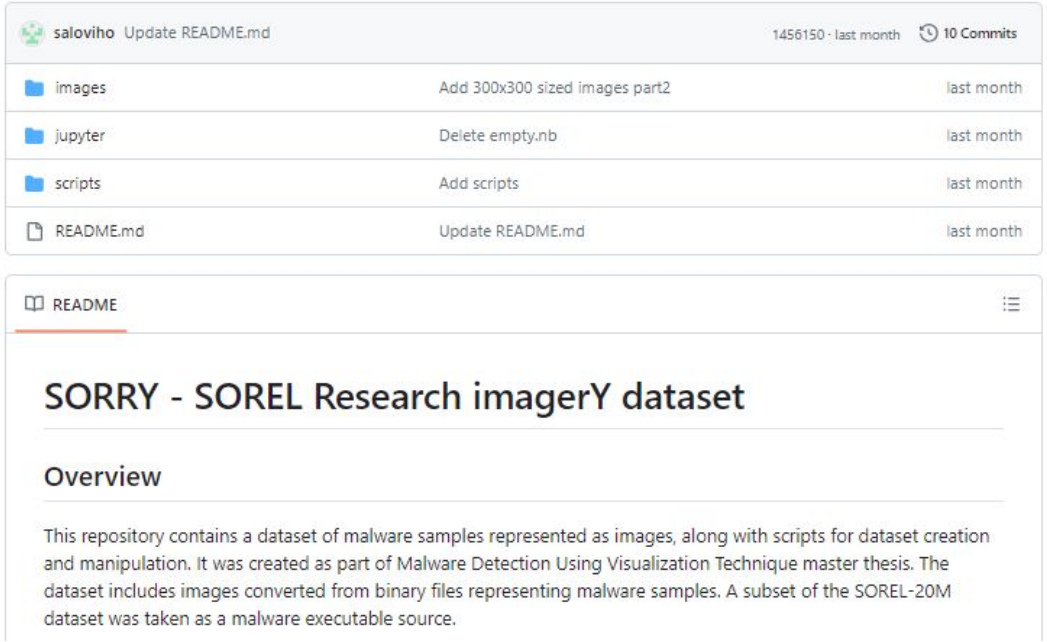


Conclusion

- SORRY dataset was created as part of this final thesis
- CNN is capable of detecting malware, by its visual representation
- RF outperformed CNN classifier and was much faster to train
- As a result of the experiment, it was found that PE features have more descriptive power than RGB byteplot
- Solution is to visualize PE related features or use combination of both methods

	Accuracy	Precision	Recall	F1-score
SORRY BC (RF)	**0.9999+**	**0.9999+**	**0.9999+**	**0.9999+**
Malevis BC	0.9139	0.9201	0.9135	0.9168
SORRY BC	0.8860	0.8784	0.8831	0.8808
Malevis MCC	0.8040	0.8241	0.8936	0.8574

Github



All source codes and malware images can be found at:

https://github.com/saloviho/SORRY

Thanks for your attention

Questions?

Sources used

1. https://decoded.avast.io/threatresearch/avast-q4-2023-threat-report/
2. https://usa.kaspersky.com/about/press-releases/2024_kaspersky-finds-one-in-three-cyber-incidents-are-due-to-ransomware
3. https://blog.checkpoint.com/research/check-point-research-2023-the-year-of-mega-ransomware-attacks-with-unprecedented-impact-on-global-organizations/
4. https://software.informer.com/Stories/how-to-get-rid-of-adware.html
5. https://www.bleepingcomputer.com/news/security/the-week-in-ransomware-august-4th-2017-globeimposter-notpetya-and-more/
6. Kinser, Jason. (2018). Image Operators: Image Processing in Python. 10.1201/9780429451188.
7. https://www.youtube.com/watch?v=Yoo4GWiAx94
8. Dale Purves. Neuroscience. Sinauer Associates, Sunderland, Mass, 2001.
9. Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics, 36*, 193-202.
10. https://en.wikipedia.org/wiki/AlexNet
11. Charu C. Aggarwal. Neural Networks and Deep Learning: A Textbook. Springer International Publishing, 2018
12. https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/
13. https://glassboxmedicine.com/2019/05/26/classification-sigmoid-vs-softmax/
14. Byers, Benjamin & Sheta, Alaa. (2023). Design of Convolutional Neural Networks for Fish Recognition and Tracking. 22. 1-9.
15. Maung, Saw. (2019). Convolutional Neural Network based Recognition of Myanmar Text Warning Sign for Mobile Platform. International Journal of Engineering Research and. V8. 10.17577/IJERTV8IS010102.
16. Hyrum S. Anderson and Phil Roth. EMBER: An Open Dataset for Training Static PE Malware Machine, 2018
17. Bozkir, Ahmet & Cankaya, Ahmet & Aydos, Murat. (2019). Utilization and Comparision of Convolutional Neural Networks in Malware Recognition. 10.1109/SIU.2019.8806511.
18. Richard Harang and Ethan M. Rudd (2020). SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection
19. Thomas Panas. 2008. Signature visualization of software binaries. In Proceedings of the 4th ACM symposium on Software visualization (SoftVis '08). Association for Computing Machinery, New York, NY, USA, 185–188. https://doi.org/10.1145/1409720.1409749
20. P. Trinius, T. Holz, J. Göbel and F. C. Freiling, "Visual analysis of malware behavior using treemaps and thread graphs," 2009 6th International Workshop on Visualization for Cyber Security, Atlantic City, NJ, USA, 2009, pp. 33-38, doi: 10.1109/VIZSEC.2009.5375540.
21. A. Azab and M. Khasawneh, "MSIC: Malware Spectrogram Image Classification," in *IEEE Access*, vol. 8, pp. 102007-102021, 2020, doi: 10.1109/ACCESS.2020.2999320.
22. InSeon Yoo. 2004. Visualizing windows executable viruses using self-organizing maps. In Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security (VizSEC/DMSEC '04). Association for Computing Machinery, New York, NY, USA, 82–89. https://doi.org/10.1145/1029208.1029222
23. Khan, Riaz Ullah & Zhang, Xiaosong & Kumar, Rajesh. (2019). Analysis of ResNet and GoogleNet models for malware detection. Journal of Computer Virology and Hacking Techniques. 15. 10.1007/s11416-018-0324-z.
24. D. A. Quist and L. M. Liebrock, "Visualizing compiled executables for malware analysis," *2009 6th International Workshop on Visualization for Cyber Security*, Atlantic City, NJ, USA, 2009, pp. 27-32, doi: 10.1109/VIZSEC.2009.5375539.