MALWARE DETECTION

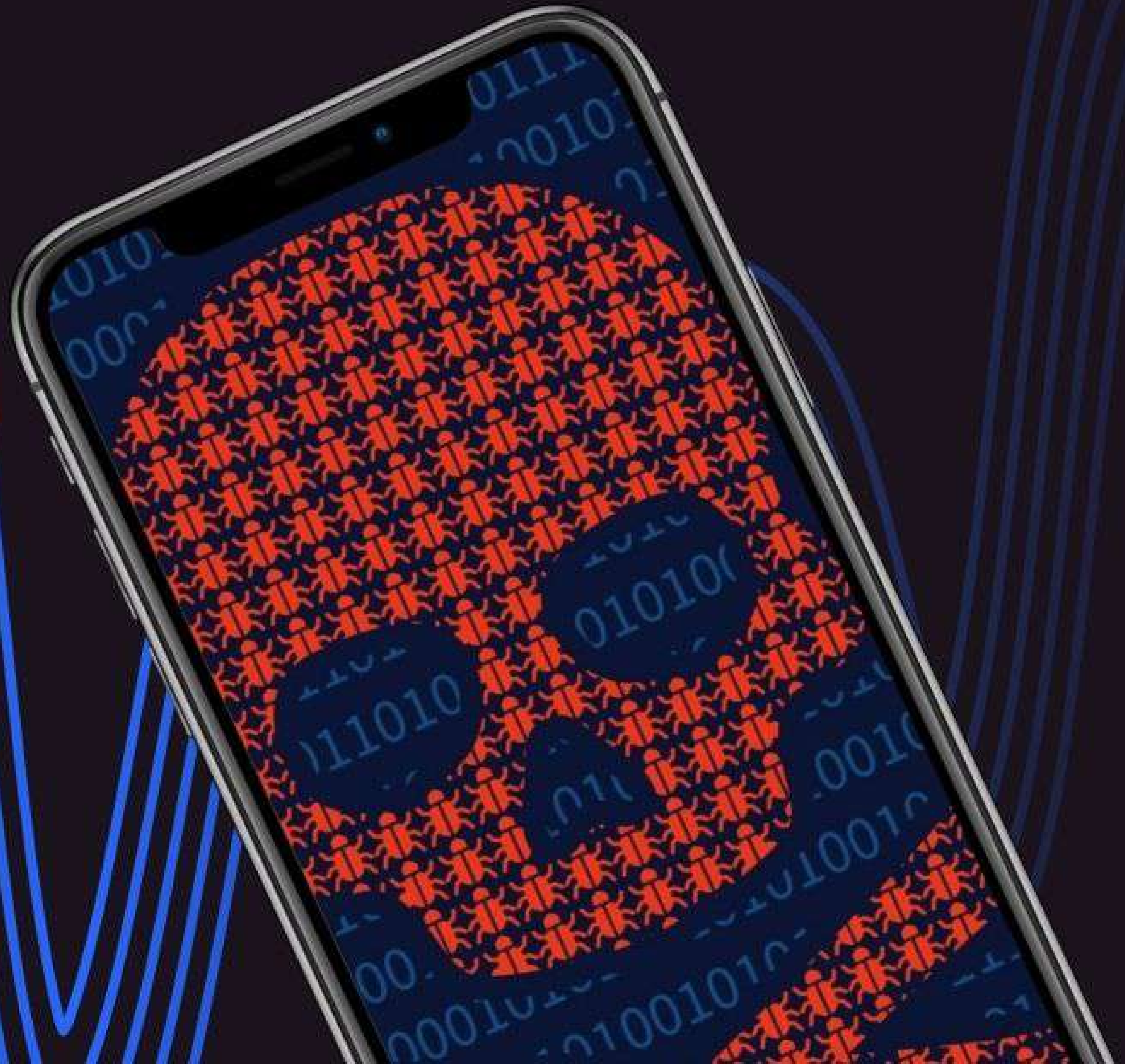
By Machine Learning Approach

WHAT IS MALWARE?

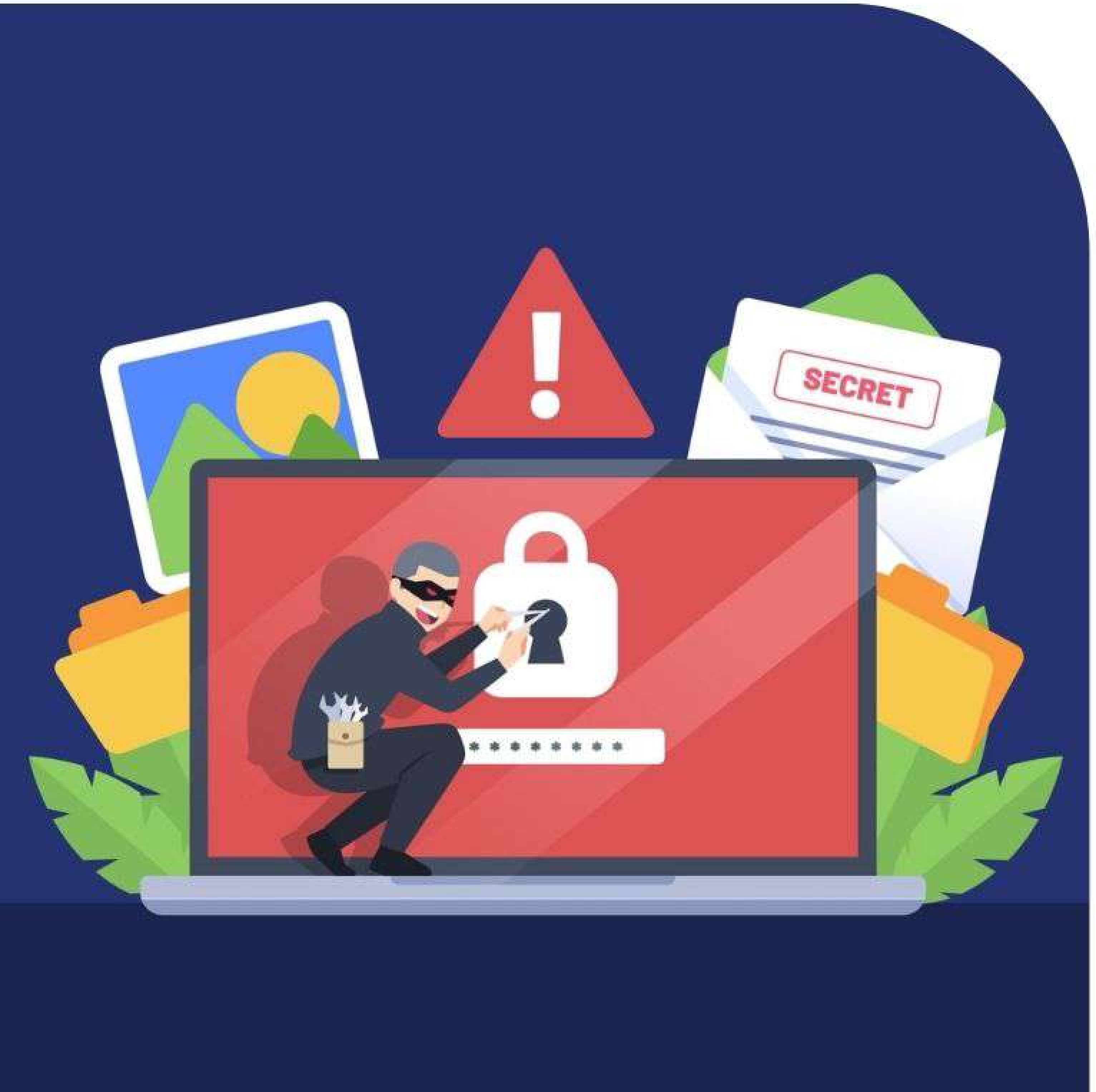
- Malware refers to malicious software designed to disrupt, damage, or gain unauthorized access to a computer system.

A wide variety of marwares exist like :

- Viruses
- Worms
- Adware
- Ransomware
- Trojan Horses

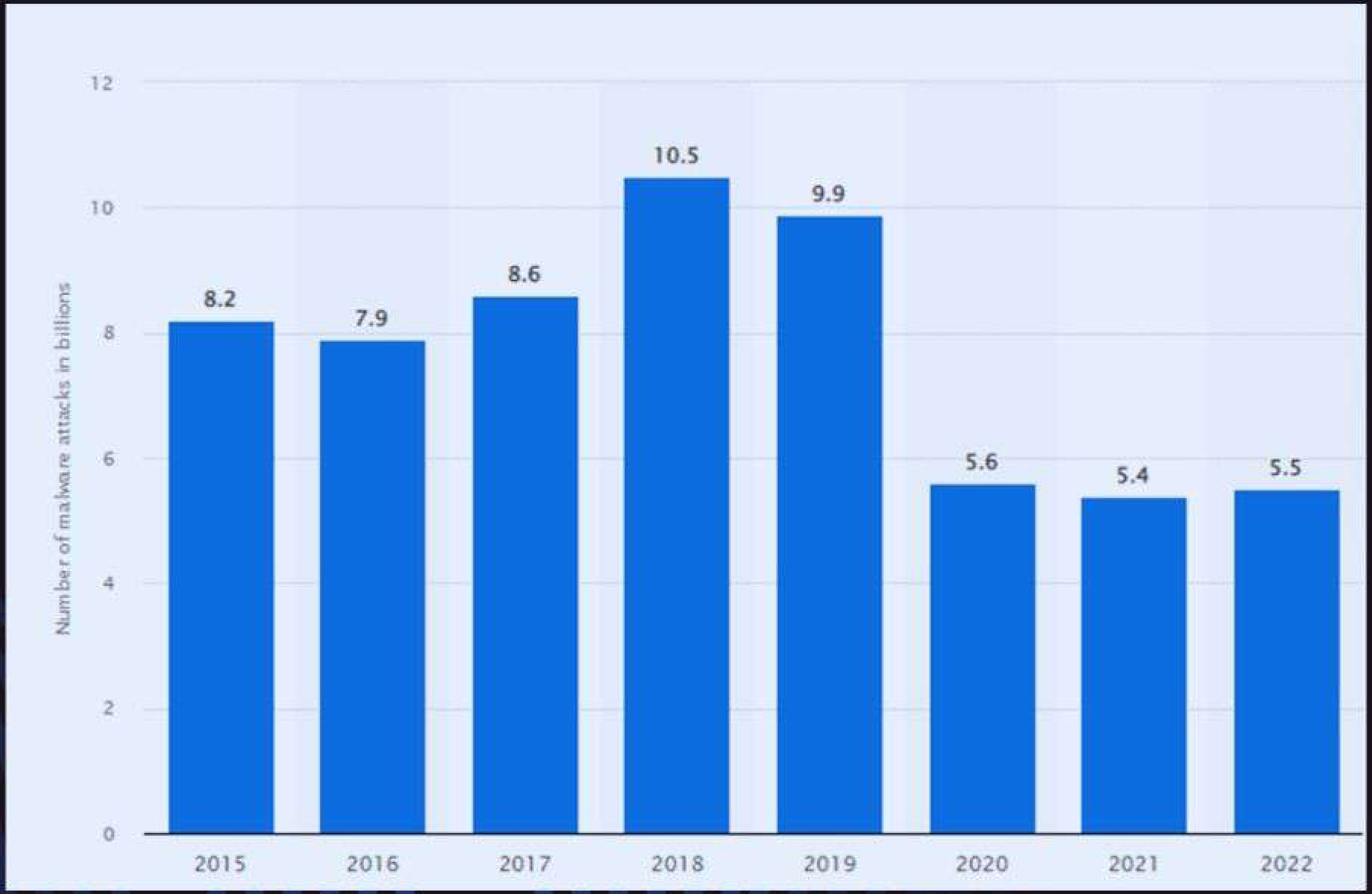


Abstract/ Analysis



- Current antivirus software's are effective against known viruses.
  - if new signature is introduced, then it is difficult to detect any malicious data.
- Signature-based detection is not that effective during zero-day attacks.
- Anti-virus organizations started applying machine learning and deep learning methods for malware analysis and detection.
- Machine learning methods can be used to create more effective anti-malware software.
  - Possible to detect unknown malwares.
- Approach used includes
  - a custom CNN model using Malimg and Malevis dataset.
  - an LSTM model with Dynamic API call sequence dataset.

WHY DETECT MALWARE?



- Malware is one of the most serious security threats on the Internet.
- Spam e-mails and denial of service attacks have malware as their underlying cause.
- it functions as an early warning system.
- Cyberattacks are on the rise, targeting governments, the military, and the public and private sectors.
- The growing dependency on digital systems has accelerated immensely during the COVID-19 pandemic..

Visualizing Malwares as Images

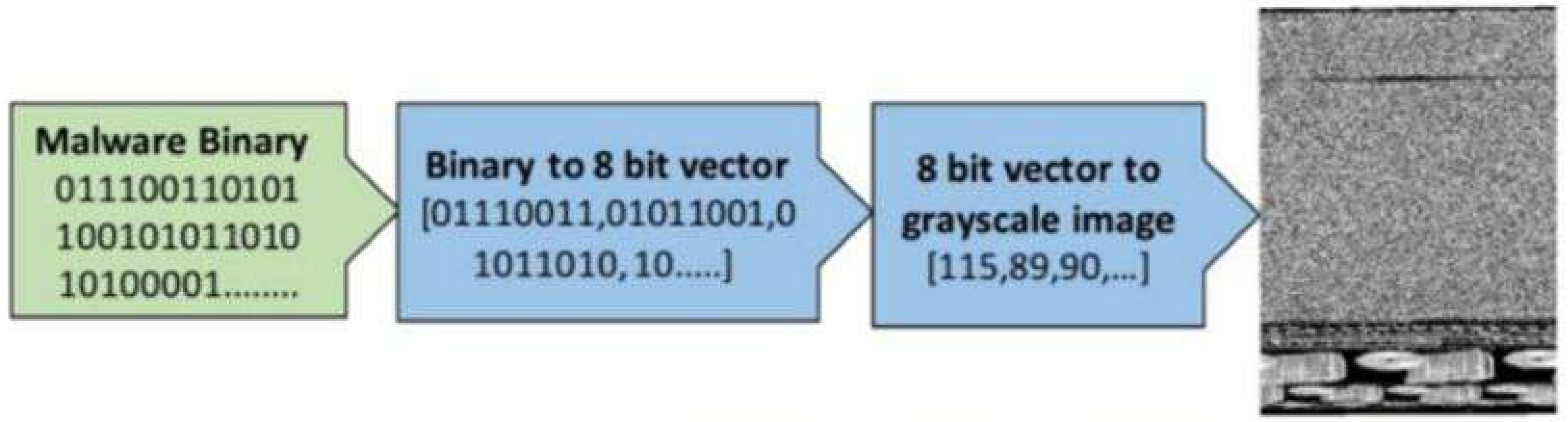
Image Based Approach

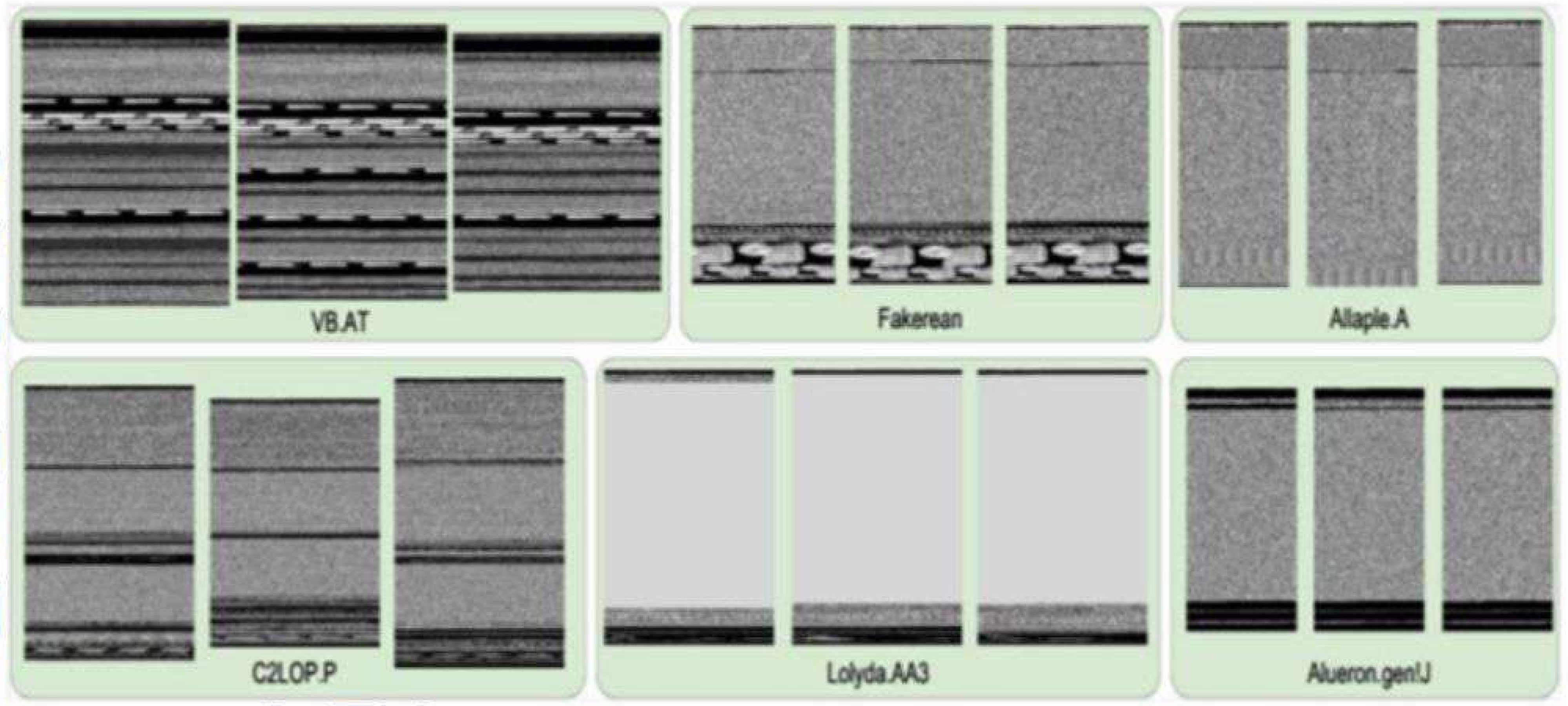
- Every byte has to be interpreted as one pixel in an image.
- The resulting array has to be organized as a 2-D array and visualized as a gray scale image.
- Values are in the range [0,255] (0:black, 255:white).
- A given malware binary file is first read in a vector of 8-bits unsigned integers.
- The binary value of each component of this vector is converted to its equivalent decimal value.
- It is then saved in a new decimal vector representative of the malware sample.
- Resulting decimal vector is reshaped to a 2D matrix and visualized as a grayscale image.

Why Should We Convert Malware to Images?

- Different sections of a binary can be easily differentiated.
- Some malware authors only change a small part of the code to produce new variants.
- Thus, if old malware is reused to create new binaries the resulting ones would be very similar.
- It is possible to detect the small changes while retaining the global structure of samples belonging to the same family.

HOW DOES IT WORK?



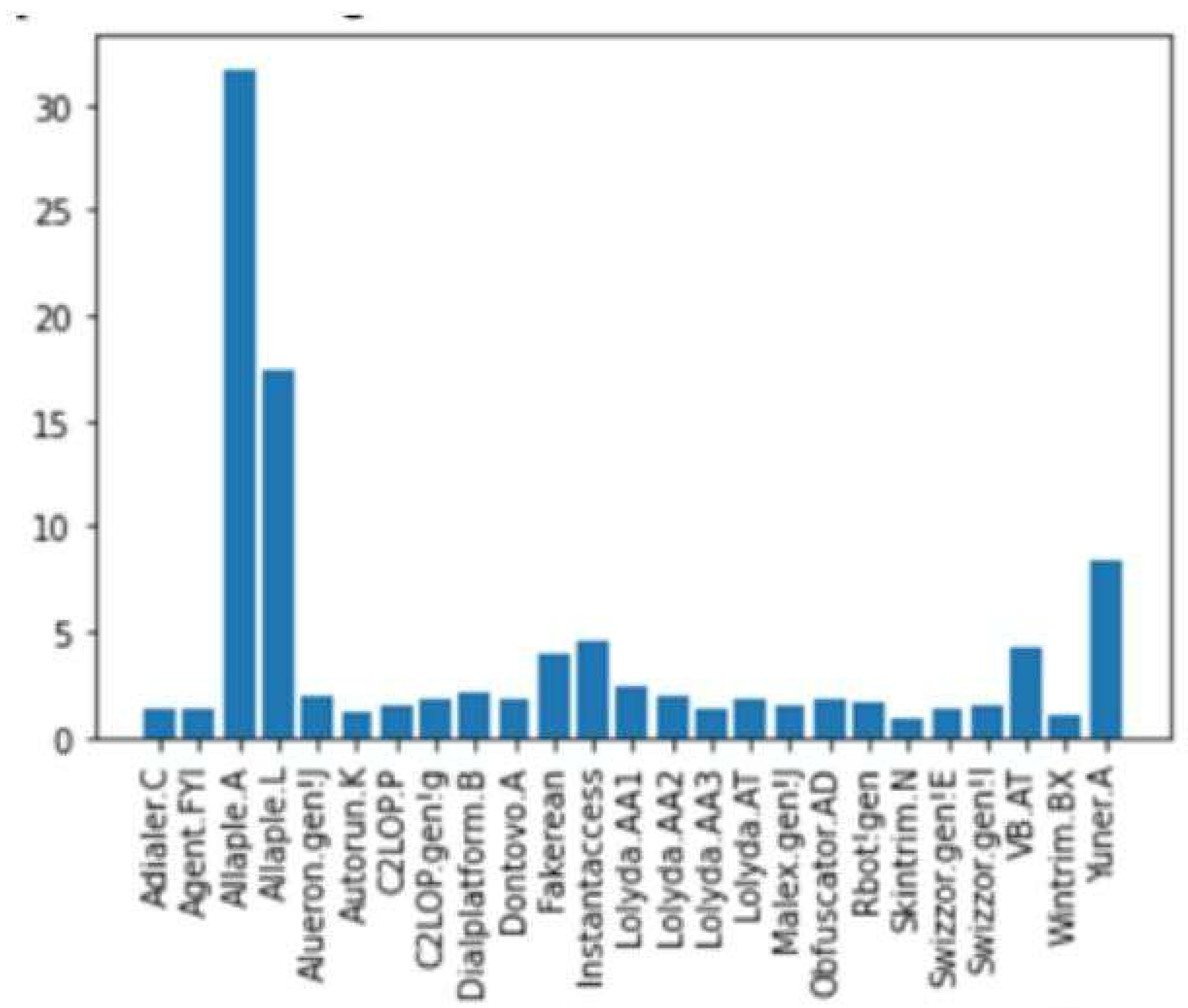


MALWARE VARIANTS BELONGING TO SAME FAMILY USUALLY HAVE SIMILAR TEXTURE (I.E. VISUAL APPEARANCE).

MalIMG and MaleVIS Datasets

MalIMG Dataset

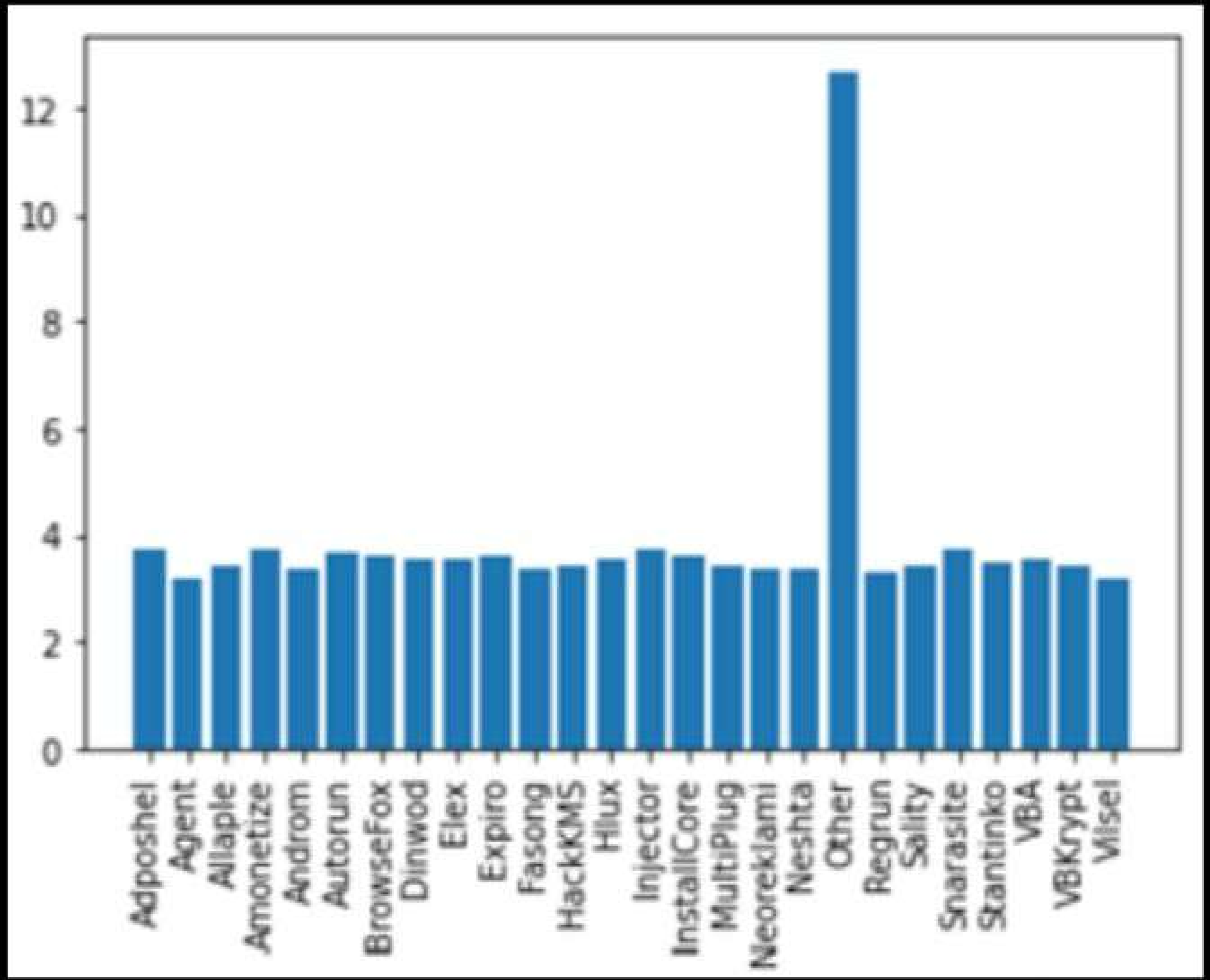
- This dataset has a total of 9,339 malware samples that are represented as grayscale images.
- Each malware sample in the dataset belongs to one of the 25 malware families/classes.
- We randomly select 90% of malware samples in a family for training and the remaining 10% for testing.



MalIMG and MaleVIS Datasets

MaleVIS Dataset

- This dataset has a total of 14,226 evenly distributed malware samples that are represented as RGB images.
- Each malware sample in the dataset belongs to one of the 25 malware families/classes.
- We randomly select 90% of malware samples in a family for training and the remaining 10% for testing.
- The MaleVIS dataset also contains a 26th class called “other” which contains non malicious software.



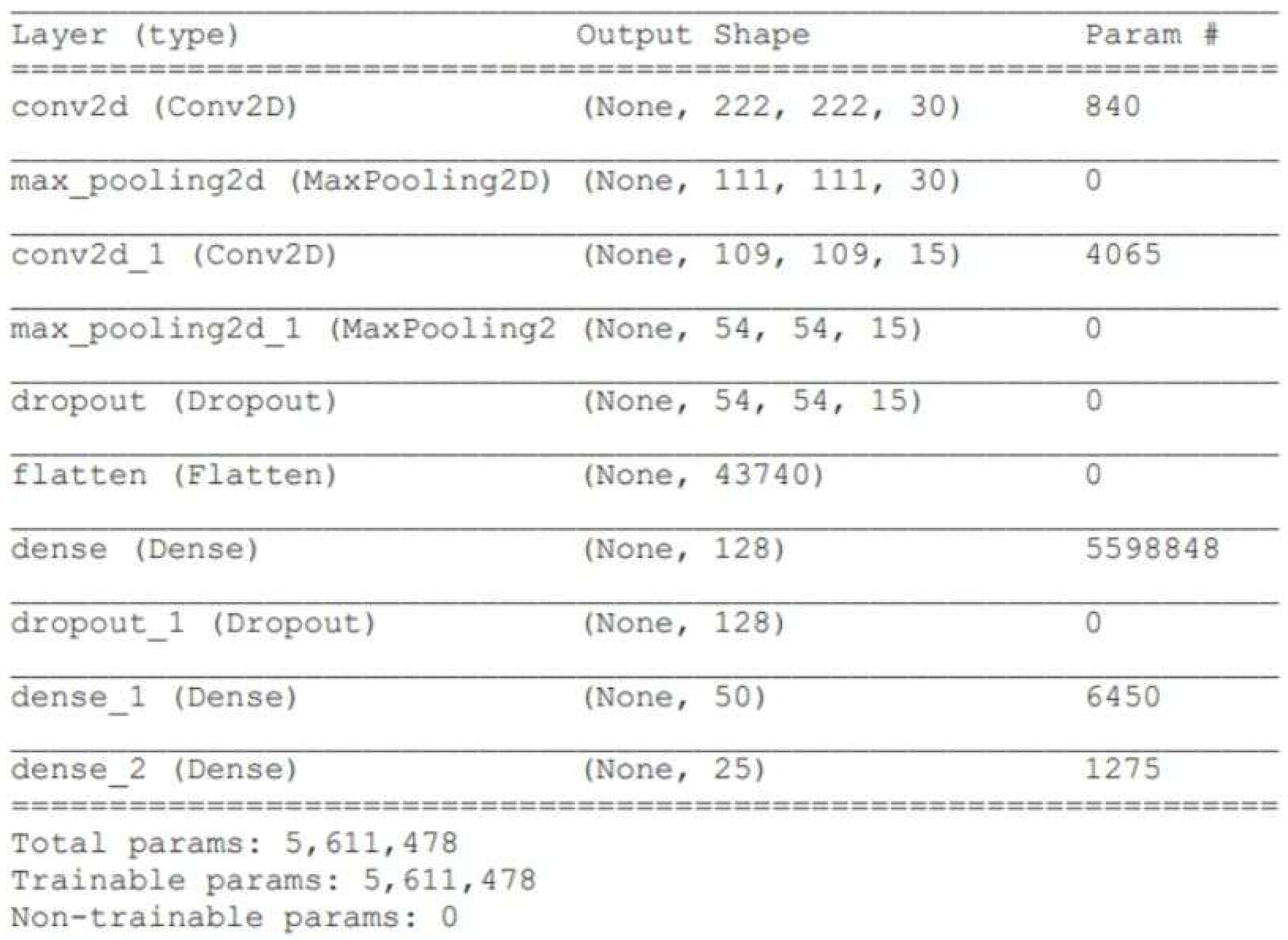
OVERVIEW OF THE CNN ARCHITECTURES USED

- We plan on comparing the results of 2 different implementations of CNNs: the first is a pre-exiting CNN taken from a Towards Data Science article, and the second one is our model.

- Basic CNN from Towards Data Science:
- 2 convolution layers
- 2 maxpooling layers
- 2 dropout layers
- 2 dense layers
- 1 final softmax layer for classification

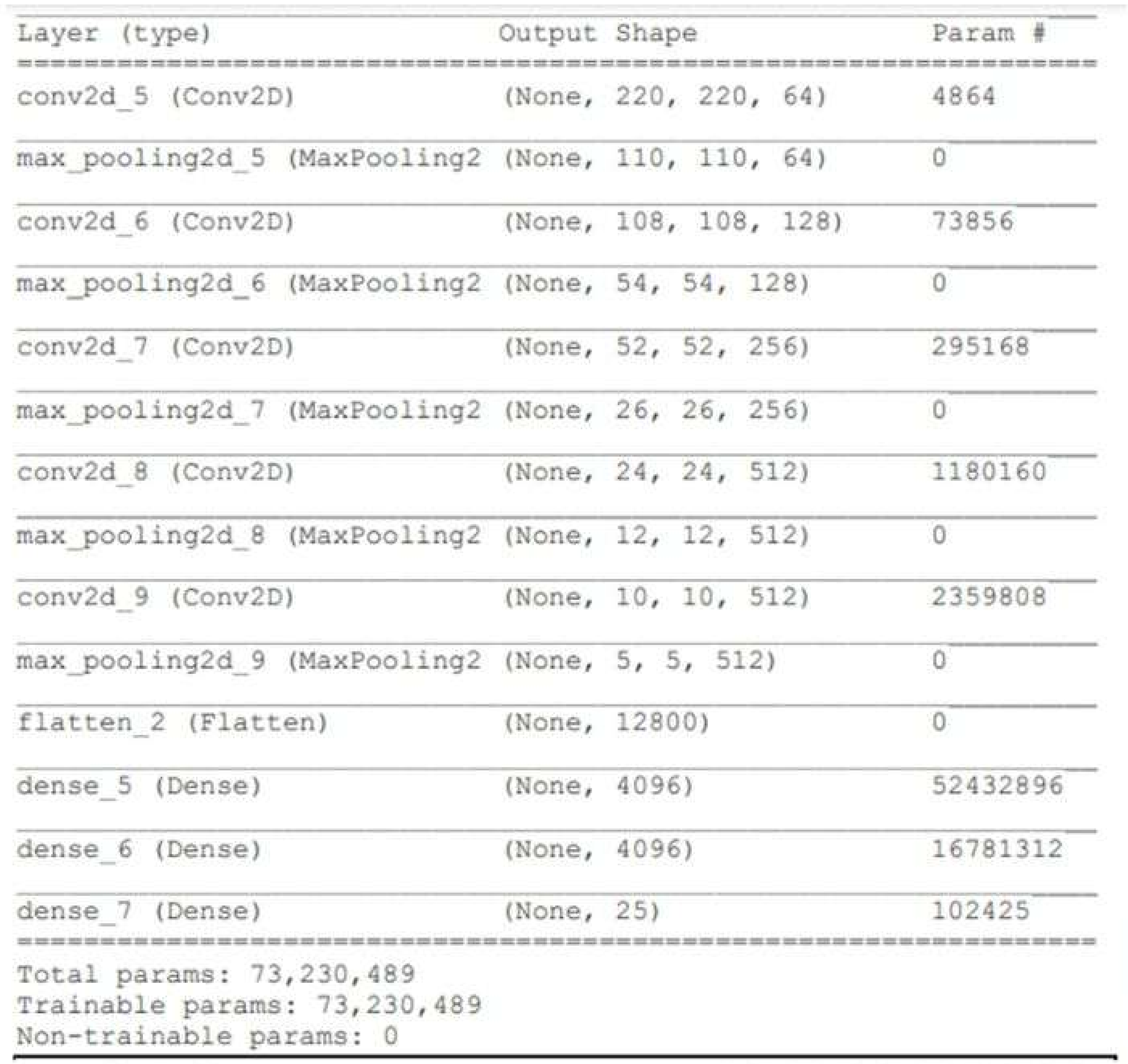
- Our Custom CNN:
- 5 convolution layers
- 5 maxpooling layers
- 2 dense layers
- 1 final softmax layer for classification

Summary:
Basic CNN from Towards Data Science

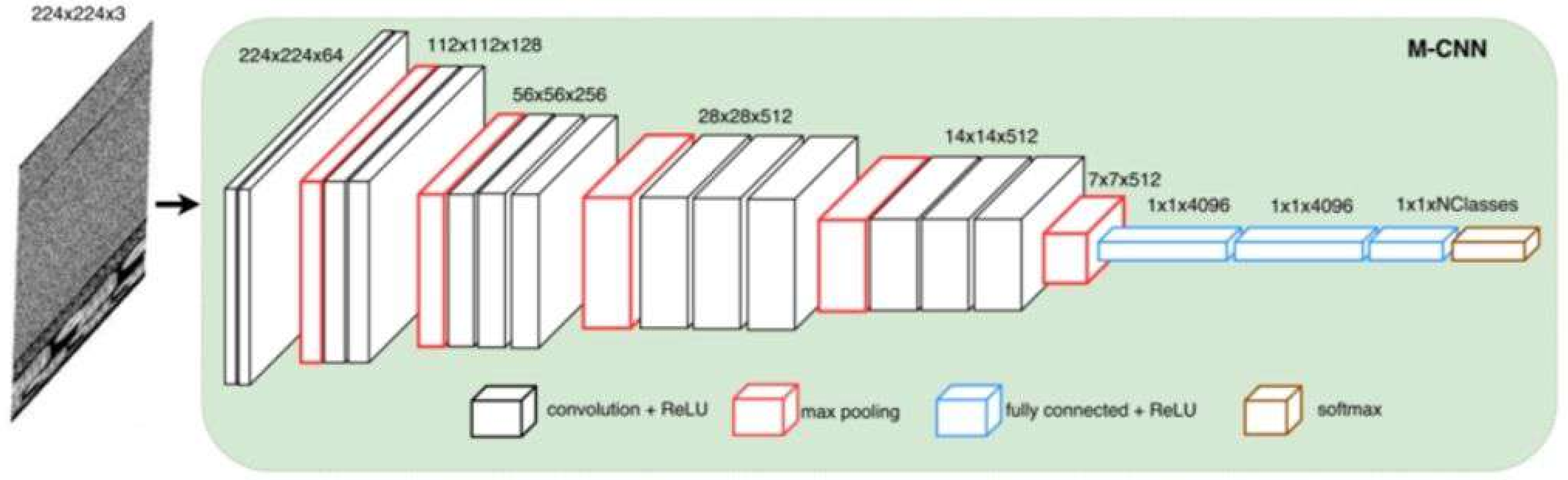


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 30)      840
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 30)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 15)      4065
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 15)        0
_________________________________________________________________
dropout (Dropout)            (None, 54, 54, 15)        0
_________________________________________________________________
flatten (Flatten)            (None, 43740)             0
_________________________________________________________________
dense (Dense)                (None, 128)               5598848
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1275
=================================================================
Total params: 5,611,478
Trainable params: 5,611,478
Non-trainable params: 0
```

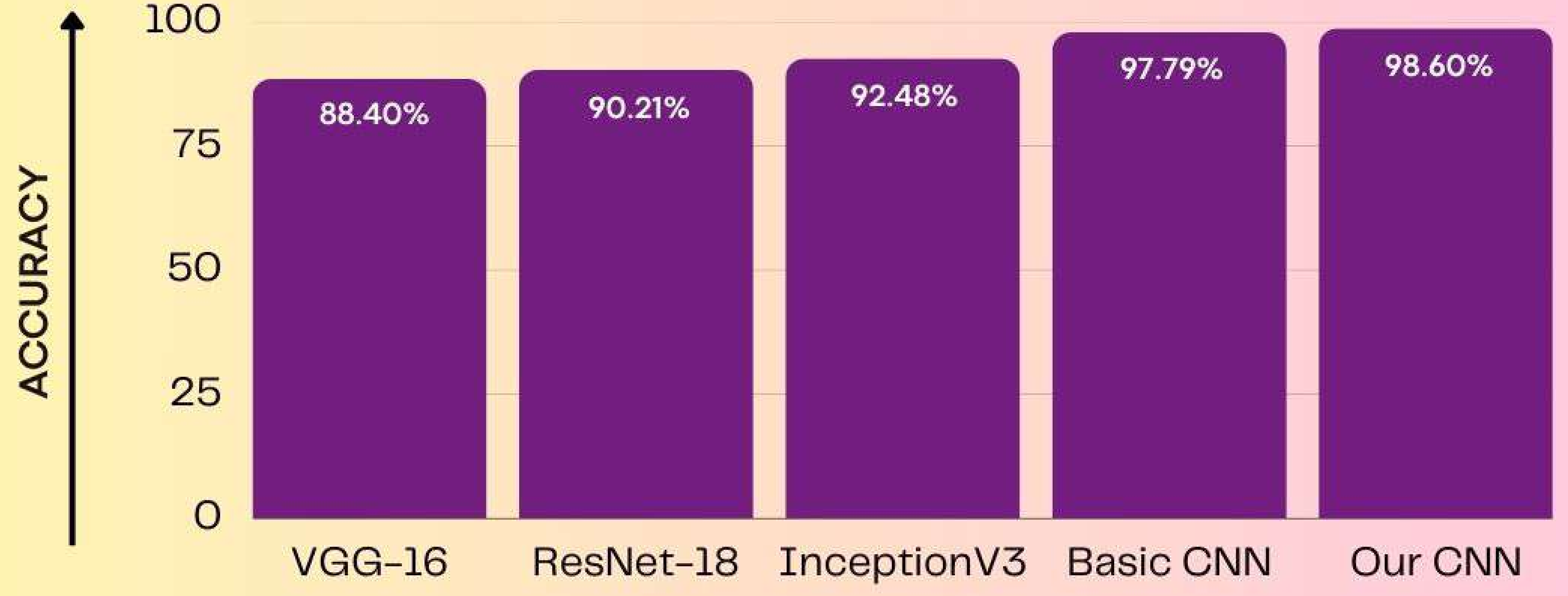
Summary:
Our Custom CNN



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 220, 220, 64)      4864
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 110, 110, 64)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 108, 108, 128)     73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 54, 54, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 52, 52, 256)       295168
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 26, 26, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 24, 24, 512)       1180160
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 12, 12, 512)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 512)       2359808
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_5 (Dense)              (None, 4096)              52432896
_________________________________________________________________
dense_6 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_7 (Dense)              (None, 25)                102425
=================================================================
Total params: 73,230,489
Trainable params: 73,230,489
Non-trainable params: 0
```

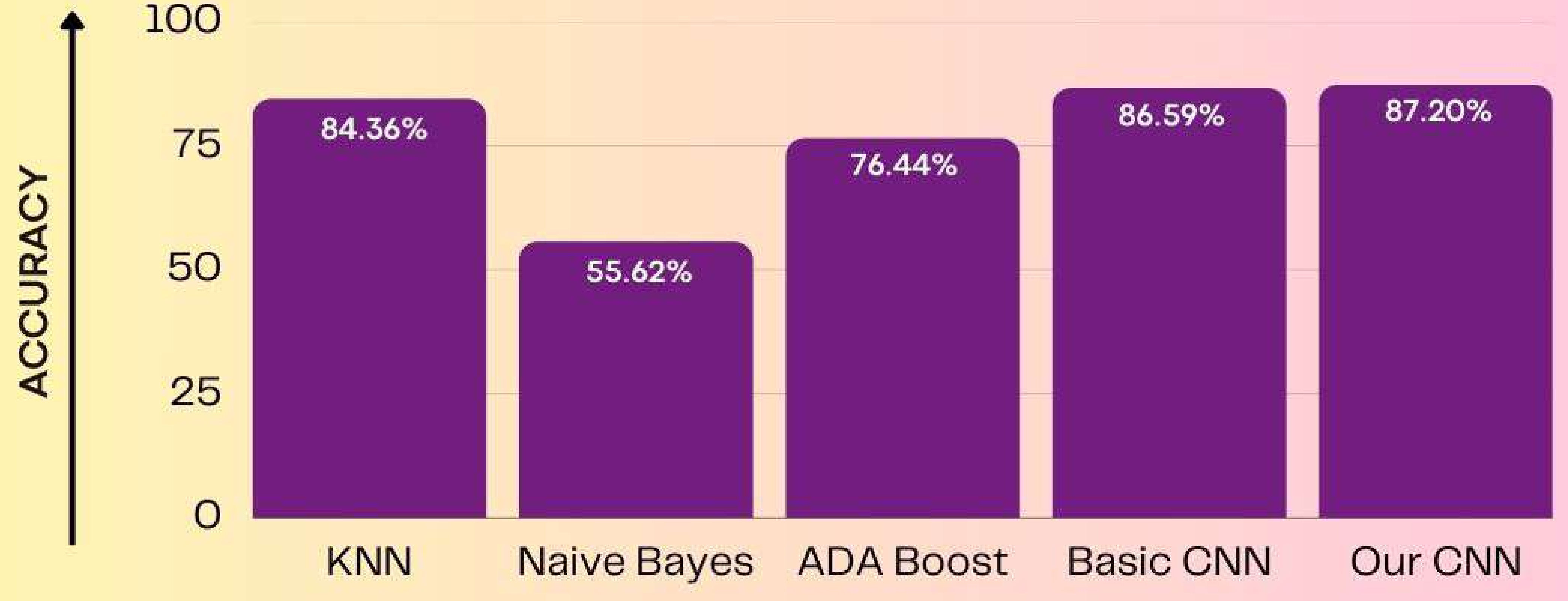


PERFORMANCE IN COMPARISION TO EXISTING MODELS



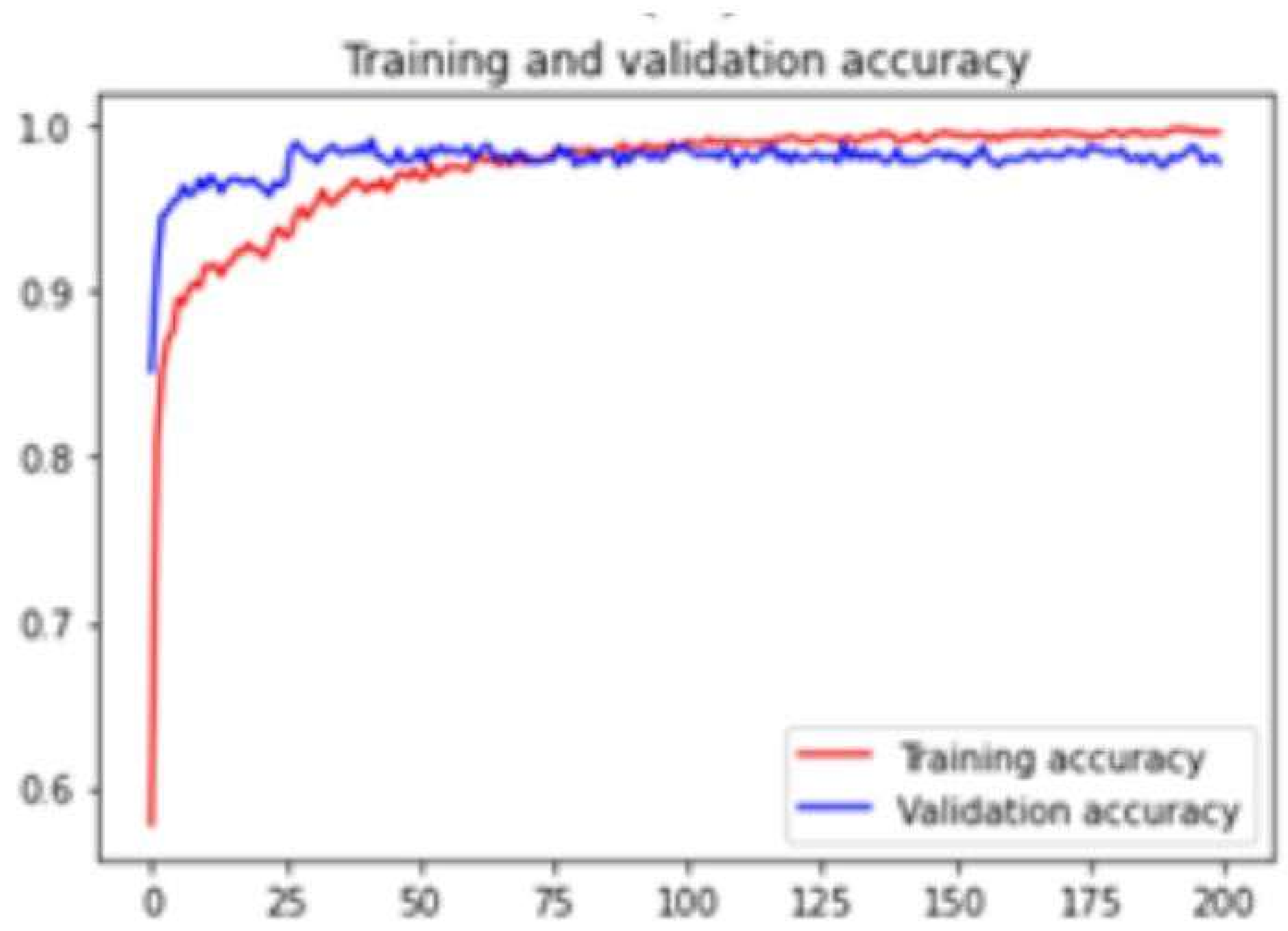
MALIMG DATASET

PERFORMANCE IN COMPARISION TO EXISTING MODELS

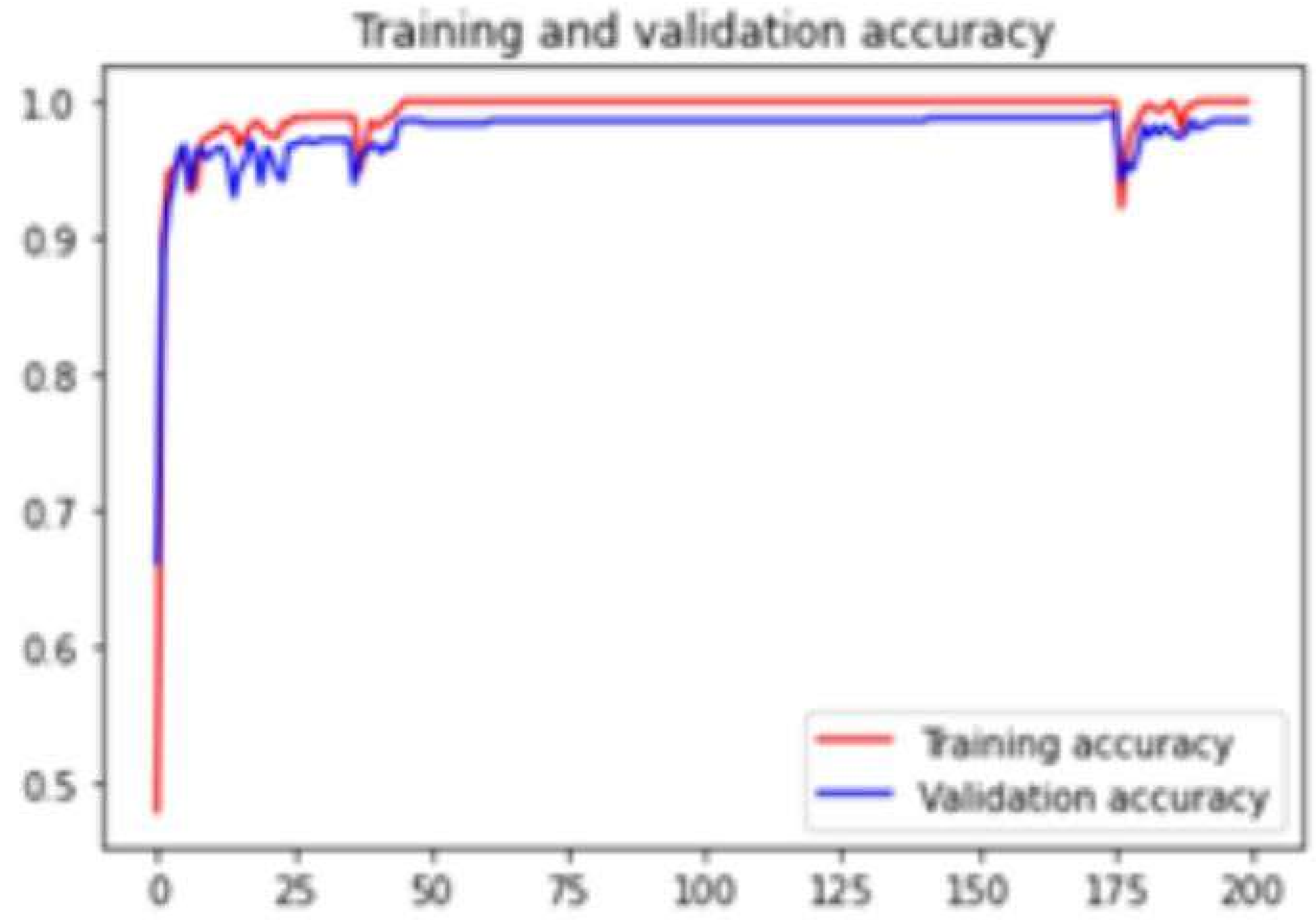


MALEVIS DATASET

MalIMG

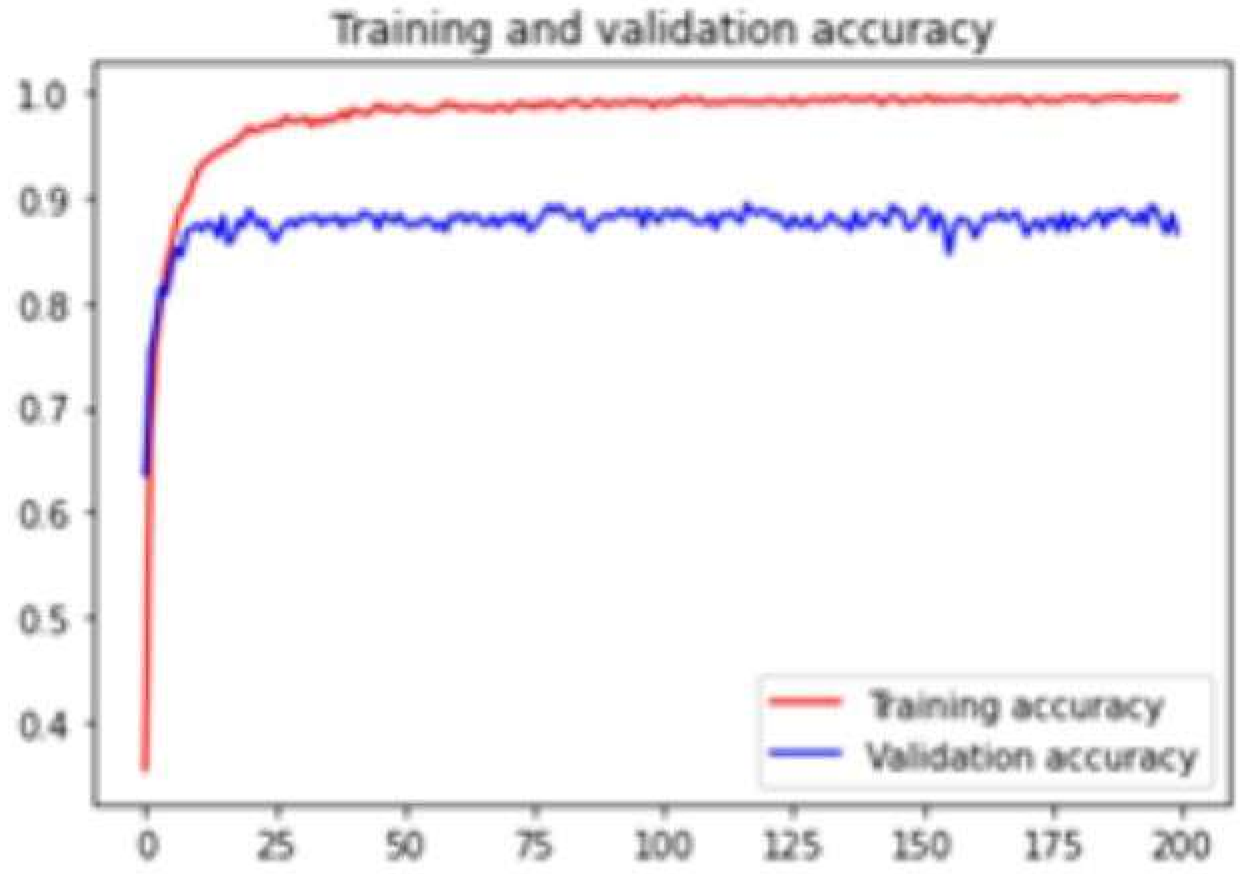


Basic CNN

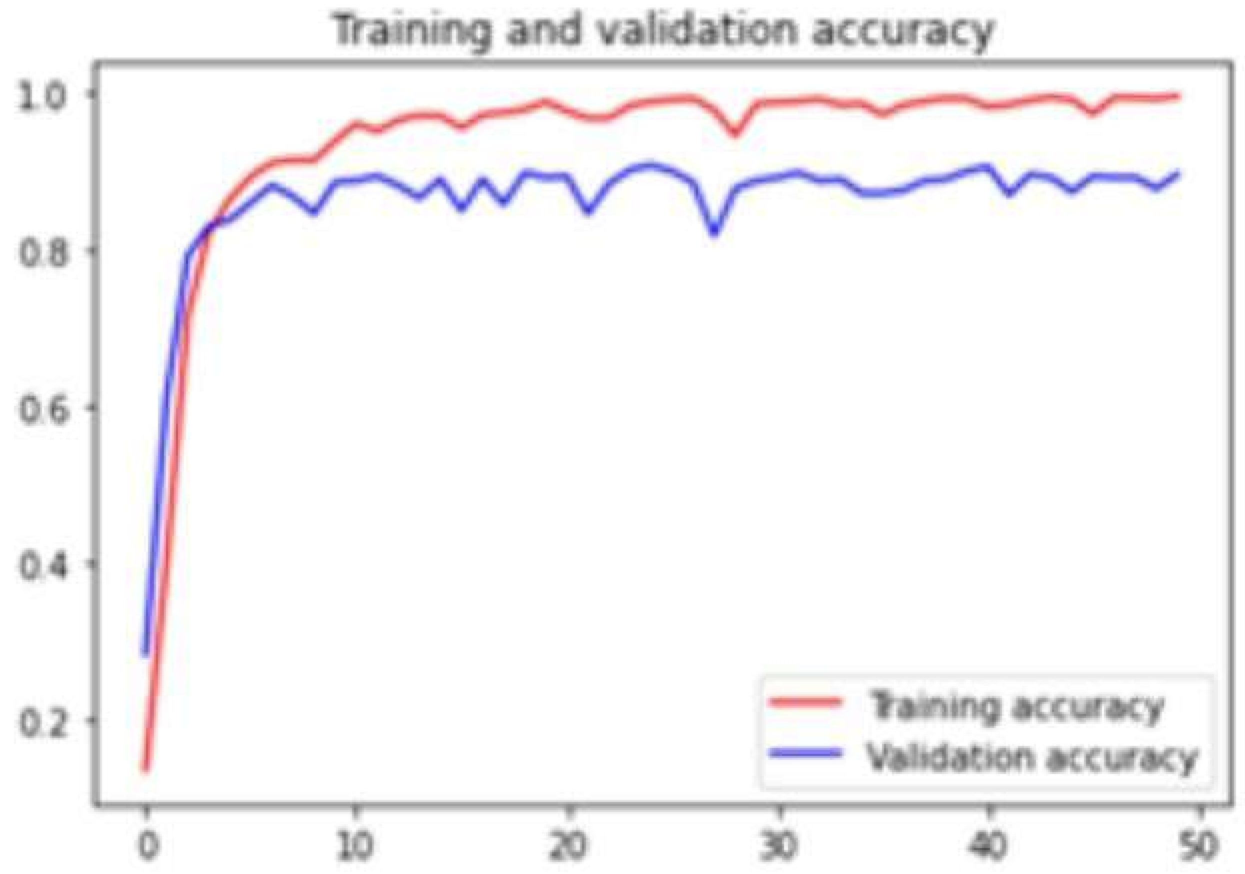


Our CNN

MaleVIS



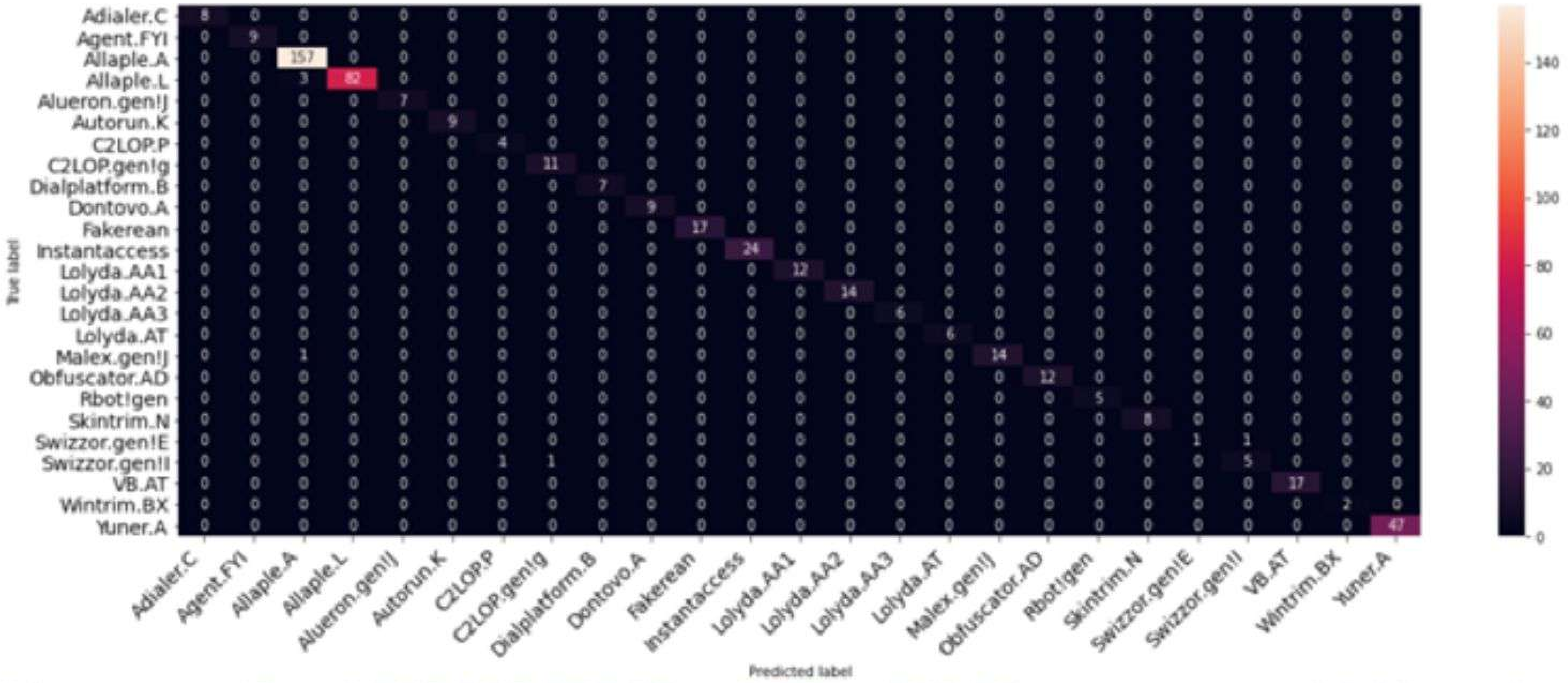
Basic CNN



Our CNN

Confusion Matrix

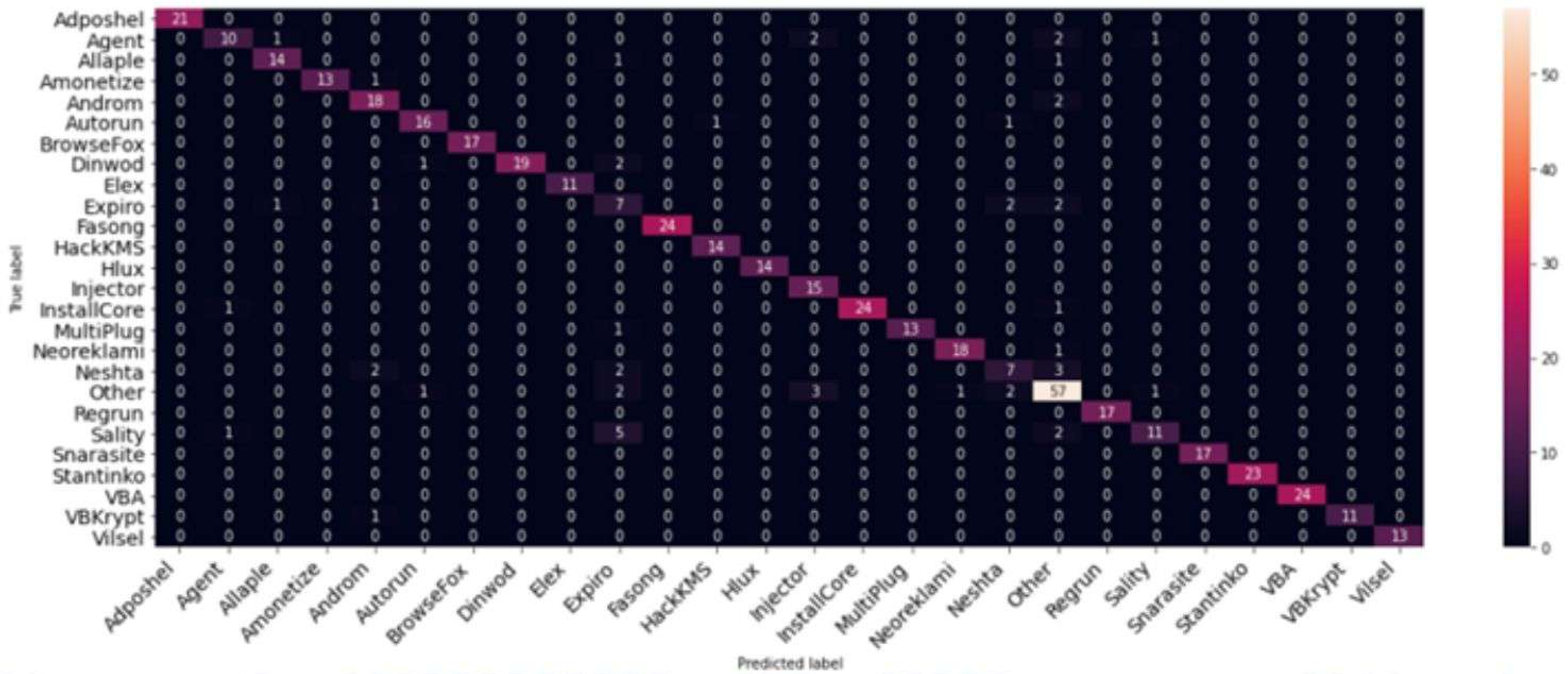
mallMG



	Adialer.C	Agent.FYI	Allaple.A	Allaple.L	Alueron.gen!J	Autorun.K	C2LOP.P	C2LOP.gen!g	Dialplatform.B	Dontovo.A	Fakerean	Instantaccess	Lolyda.AA1	Lolyda.AA2	Lolyda.AA3	Lolyda.AT	Malex.gen!J	Obfuscator.AD	Rbot!gen	Skintrim.N	Swizzor.gen!E	Swizzor.gen!I	VB.AT	Wintrim.BX	Yuner.A
Adialer.C	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Agent.FYI	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Allaple.A	0	0	157	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Allaple.L	0	0	3	82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Alueron.gen!J	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Autorun.K	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2LOP.P	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2LOP.gen!g	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Dialplatform.B	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Dontovo.A	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fakerean	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Instantaccess	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0
Lolyda.AA1	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0
Lolyda.AA2	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0
Lolyda.AA3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0
Lolyda.AT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0
Malex.gen!J	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0
Obfuscator.AD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0
Rbot!gen	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
Skintrim.N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0
Swizzor.gen!E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
Swizzor.gen!I	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
VB.AT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0
Wintrim.BX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
Yuner.A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47

Confusion Matrix

maleVIS



	Adposhel	Agent	Allaple	Amonetize	Androm	Autorun	BrowseFox	Dinwod	Elex	Expiro	Fasong	HackKMS	Hlux	Injector	InstallCore	MultiPlug	Neoreklami	Neshta	Other	Regrun	Sality	Snarasite	Stantinko	VBA	VBKrypt	Vilsel
Adposhel	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Agent	0	10	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	1	0	0	0	0	0
Allaple	0	0	14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Amonetize	0	0	0	13	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Androm	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
Autorun	0	0	0	0	0	16	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
BrowseFox	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Dinwod	0	0	0	0	0	1	0	19	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Elex	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Expiro	0	0	1	0	1	0	0	0	0	7	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0
Fasong	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
HackKMS	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hlux	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0
Injector	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0
InstallCore	0	1	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	1	0	0	0	0	0	0	0
MultiPlug	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0
Neoreklami	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	1	0	0	0	0	0	0	0
Neshta	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0	0	0	7	3	0	0	0	0	0	0	0
Other	0	0	0	0	0	1	0	0	0	2	0	0	0	3	0	0	1	2	57	0	1	0	0	0	0	0
Regrun	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0
Sality	0	1	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	2	0	11	0	0	0	0	0
Snarasite	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0
Stantinko	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0
VBA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0
VBKrypt	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0
Vilsel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13

Conclusion

- We noticed that both CNNs perform very similarly on the MalIMG dataset with our custom CNN having the highest final average accuracy score.
- We also noticed that even though the results were relatively close for the MaleVIS dataset, they were very far in accuracy from the MalIMG accuracy numbers which is most probably due to the "other" class in the MaleVIS dataset.
- We can see from the Confusion matrices that the "other" class is responsible for the most false positives.
- Another reason being that the MaleVIS images are RGB.

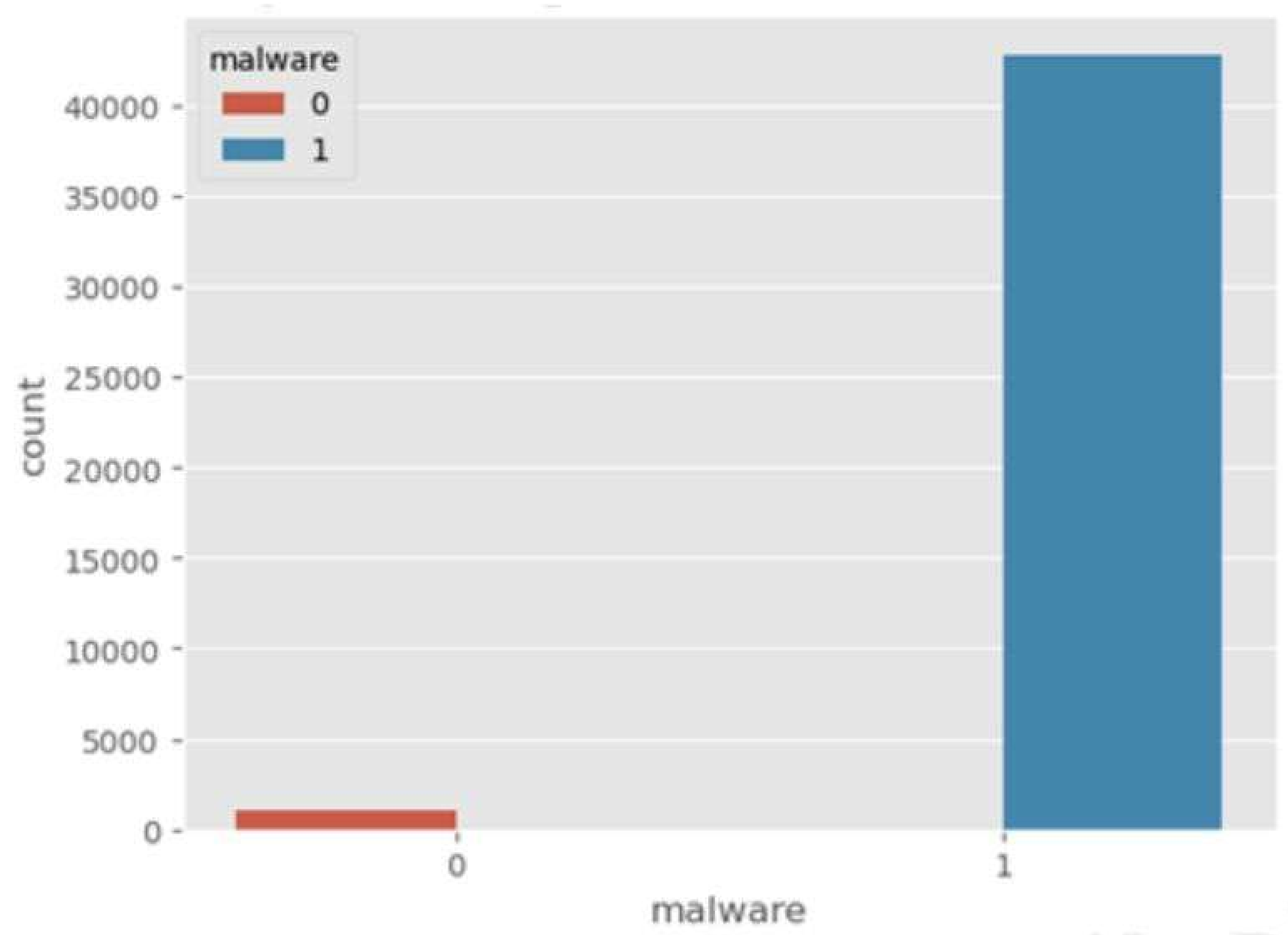
Long Short-Term Memory (LSTM)

Dynamic API call sequence

- Analysing the results of the implementations of CNN-LSTM.
- Reference taken from a paper by Weizhong Qiang, Lin Yang, Hai Jin, entitled: "Efficient and Robust Malware Detection Based on Control Flow Traces Using Deep Neural Networks".
- Pre-process the dataset in a way that it becomes easy for testing and if there is a new malware thrown at our model, it will detect it more efficiently.

API Call Sequences Dataset

- The dataset contains 42,797 malware API call sequences and 1,079 goodware API call sequences.
- Each API call sequence is composed of the first 100 non-repeated consecutive API calls associated with the parent process
- Malware samples collected from VirusShare.
- Goodware samples collected from both portablepps.com and a 32-bit Windows 7 Ultimate directory.



OVERVIEW OF THE CNN-LSTM ARCHITECTURE USED

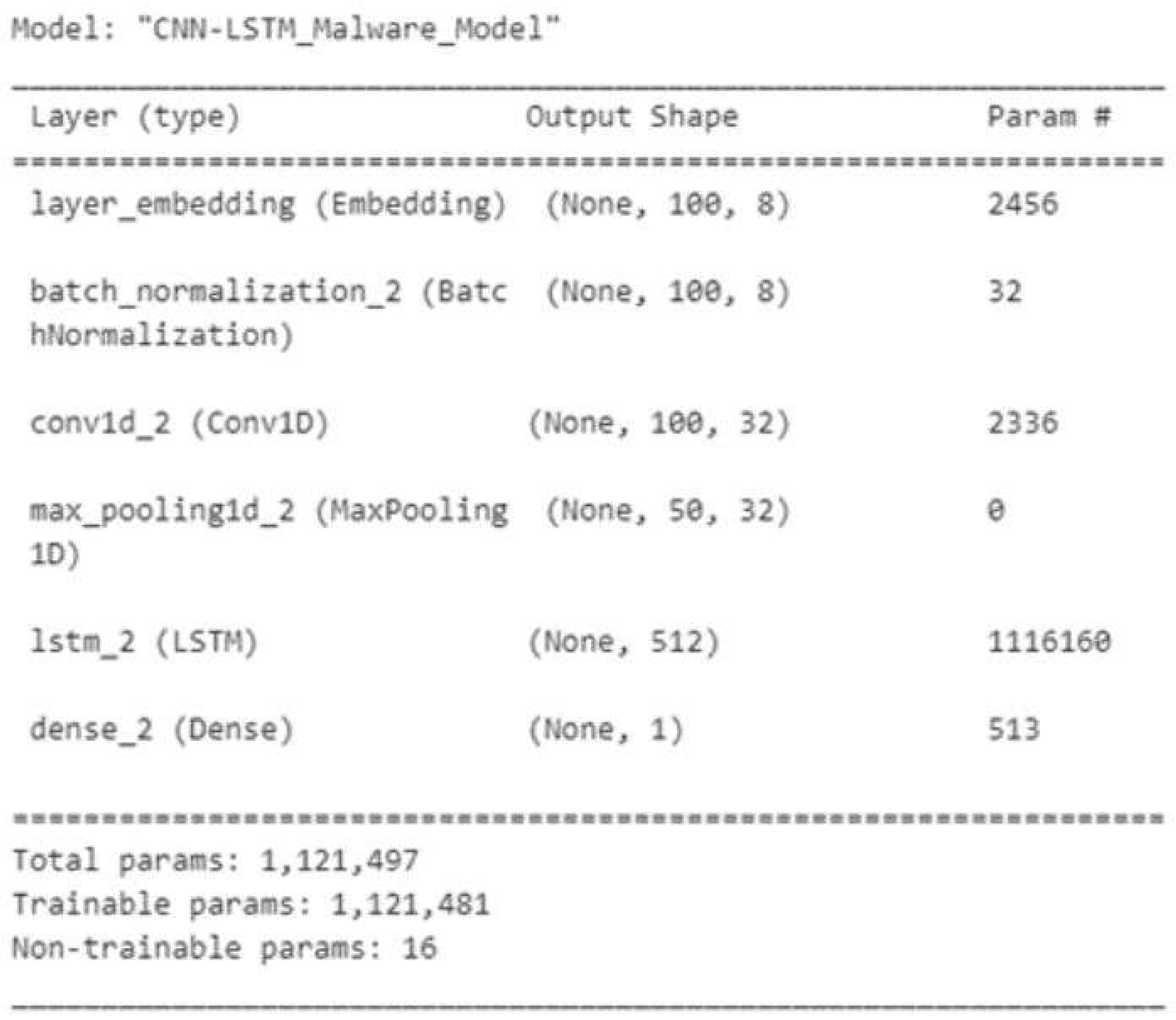
- We plan on Pre-Processing the dataset with the help of a normalization technique. For this purpose, MinMaxScaler was used.

- It s a normalization technique used to scale numeric features to a range of [0, 1]. It transforms each feature such that the minimum value of the feature is 0 and the maximum value is 1.

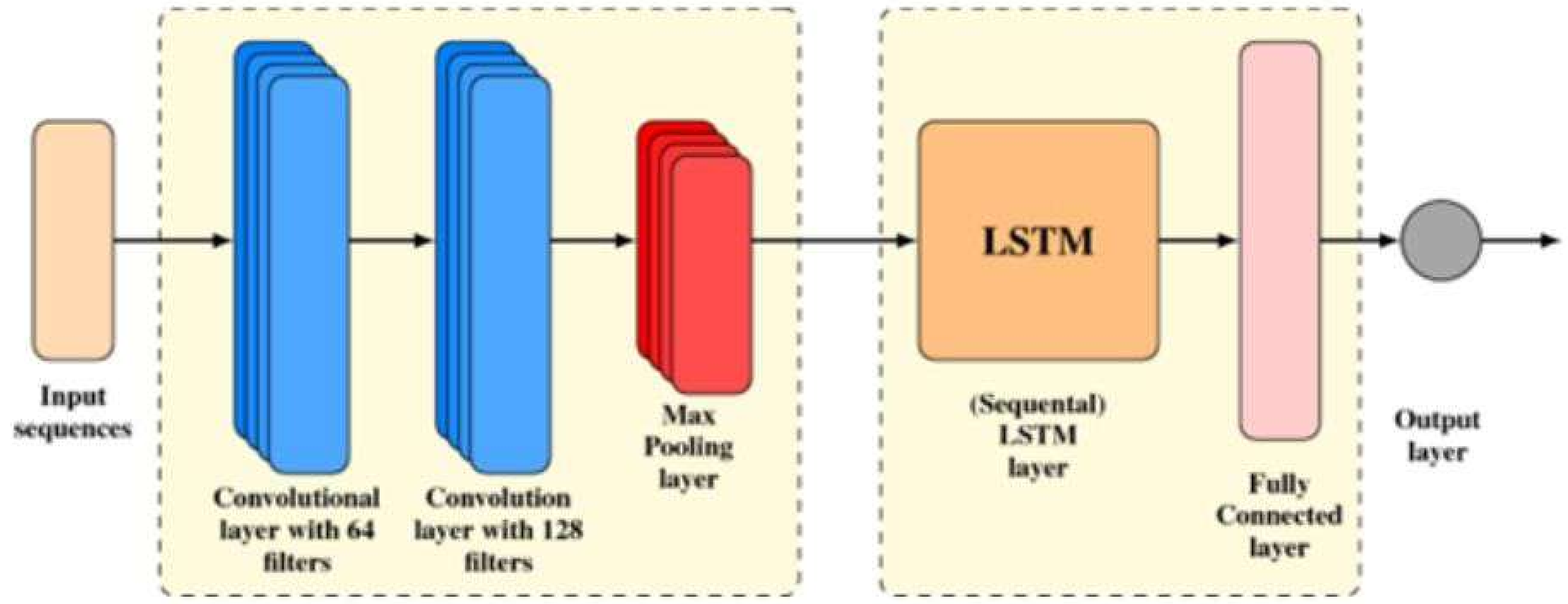
- It can work well with LSTM machine learning models for malware detection, especially when dealing with input sequences of different lengths.

- Our CNN-LSTM Model:
- 1 Embedding layer
- 1 BatchNormalization layer
- 1 Convolution layer
- 1 Maxpooling layer
- 1 LSTM layer
- 1 Dense layer

Summary:
CNN-LSTM Architecture

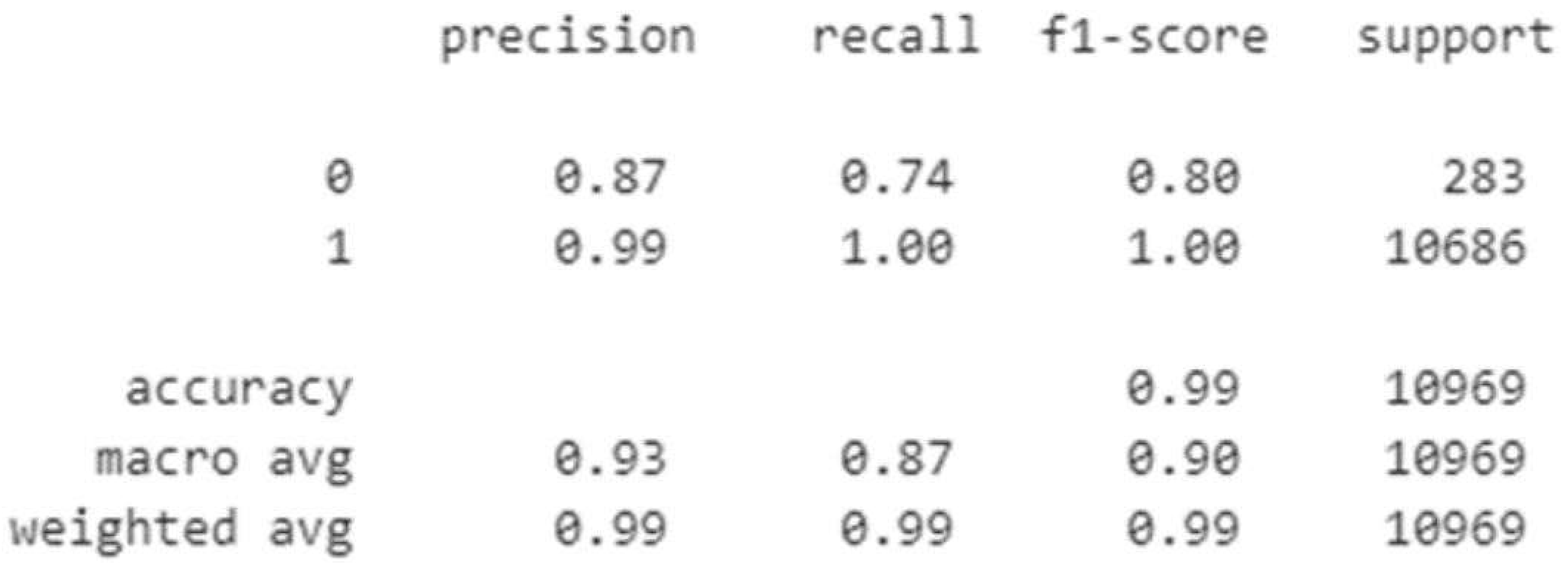


```
Model: "CNN-LSTM_Malware_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer_embedding (Embedding)  (None, 100, 8)           2456

 batch_normalization_2 (Batc  (None, 100, 8)           32
 hNormalization)

 conv1d_2 (Conv1D)           (None, 100, 32)           2336

 max_pooling1d_2 (MaxPooling  (None, 50, 32)           0
 1D)

 lstm_2 (LSTM)               (None, 512)               1116160

 dense_2 (Dense)             (None, 1)                 513

=================================================================
Total params: 1,121,497
Trainable params: 1,121,481
Non-trainable params: 16
_________________________________________________________________
```

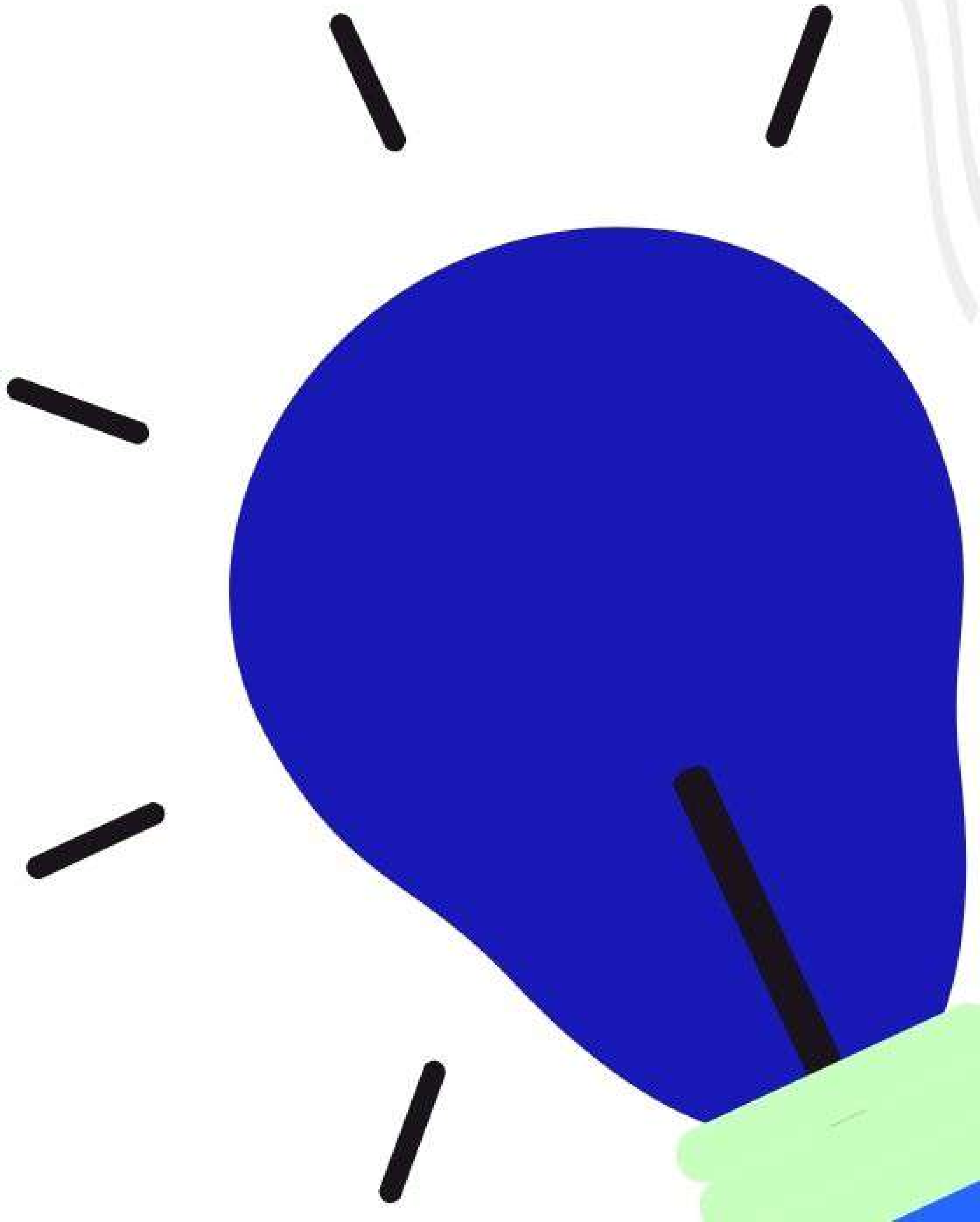


PERFORMANCE METRICS

- Validation Accuracy and other performance parameters

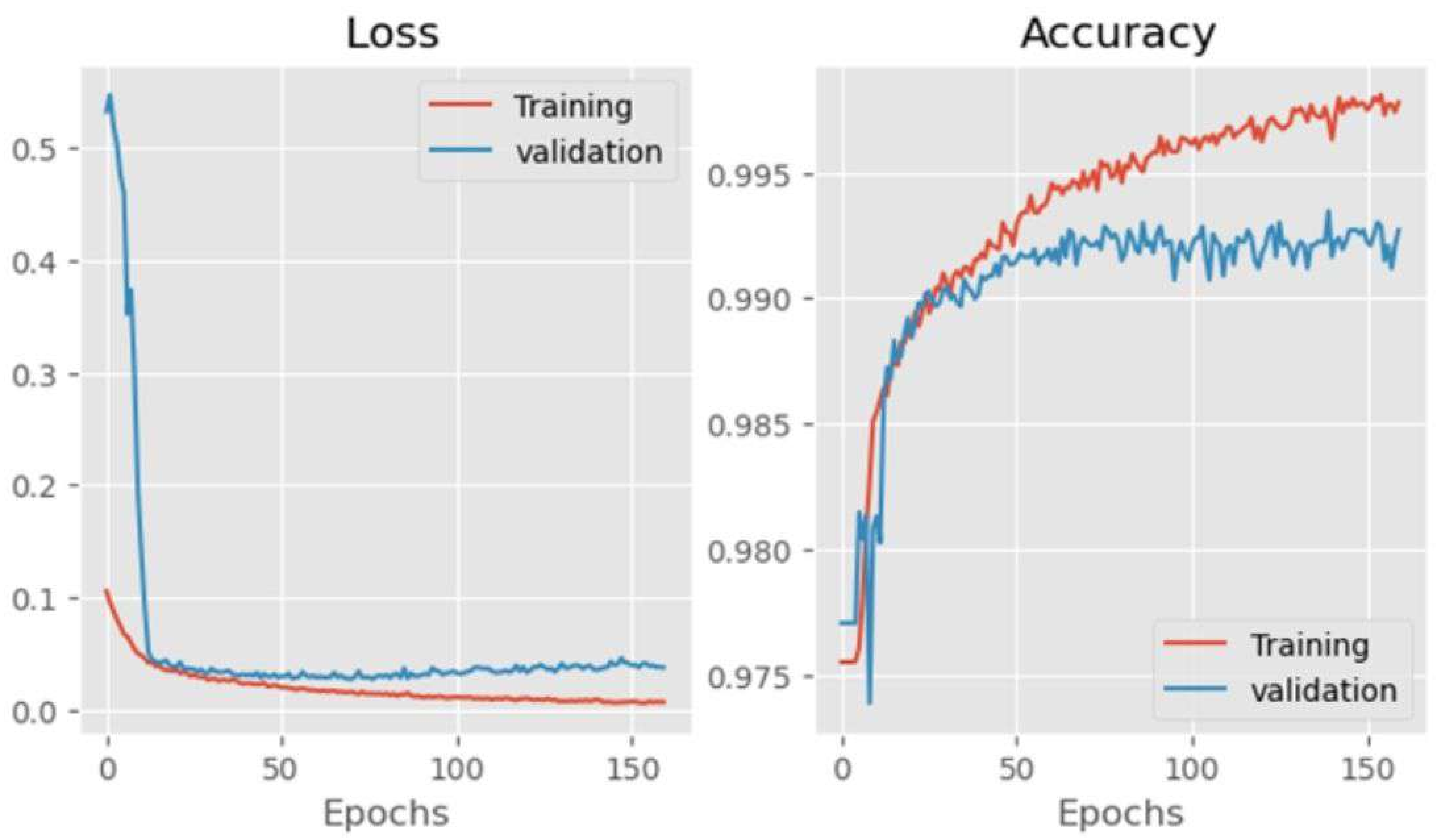


```
              precision    recall  f1-score   support

           0       0.87      0.74      0.80       283
           1       0.99      1.00      1.00     10686

    accuracy                           0.99     10969
   macro avg       0.93      0.87      0.90     10969
weighted avg       0.99      0.99      0.99     10969
```



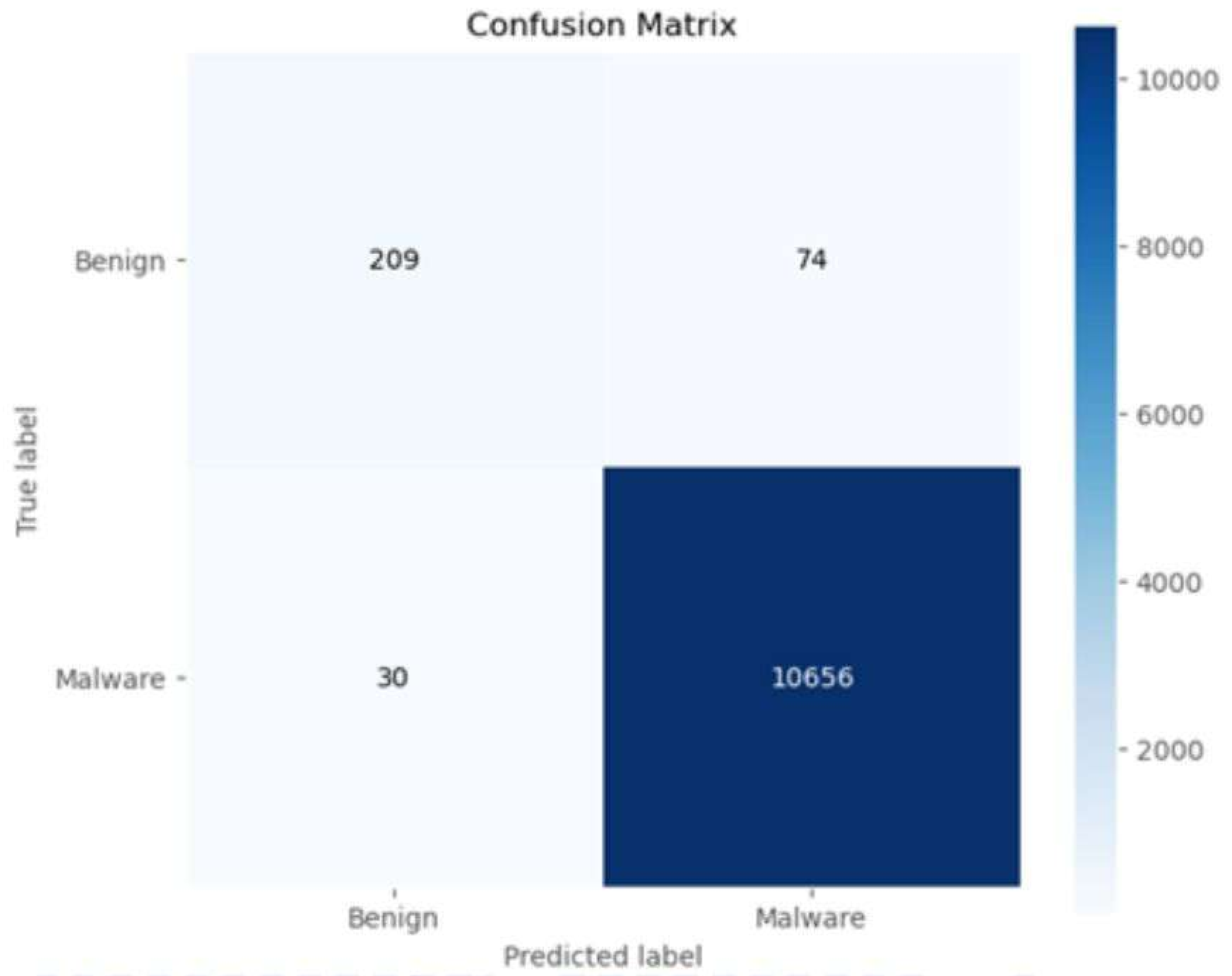
API Call Sequences Dataset

Results of the training and validation accuracy evolution per epoch and loss for the last training iteration.



Confusion Matrix

API Call Sequences Dataset



Conclusion

- The CNN-LSTM model achieved high accuracy (0.99) in detecting malware using the dynamic API call sequence dataset.
- The model had a high precision (0.99) and recall (0.99) for the weighted average.
- For the macro average, precision (0.93) and recall (0.87) were slightly lower, but still relatively high, indicating that the model performed well in detecting both classes.
- Overall, the high accuracy and precision of the model suggest that it could be effective in detecting malware in real-world scenarios.