Microsoft-PE-malware-detection

Using machine learning and Deep Learning Methods

Submitters:

Ohad Wolfman 316552496

Ohad Taizi 313465833

INTRODUCTION

- Malware detection is the process of ascertaining the presence of malware on a system or determining whether a program is malicious or harmless so that the system can be protected or recovered from any effects caused by the malicious code
- As the number of legitimate users of the Internet increases, so do the opportunities for cybercriminals to gain from manufacturing malware
- This is the reason that prompted the authors of the article we investigated to develop a model for predicting whether a PE file is malicious or benign by methods of deep learning and group learning.
- We implemented the idea in the models and tried to slightly improve the results, which we did manage to do eventually.
- We used the dataset of the research from Kaggle.

GETTING KNOW THE DATA

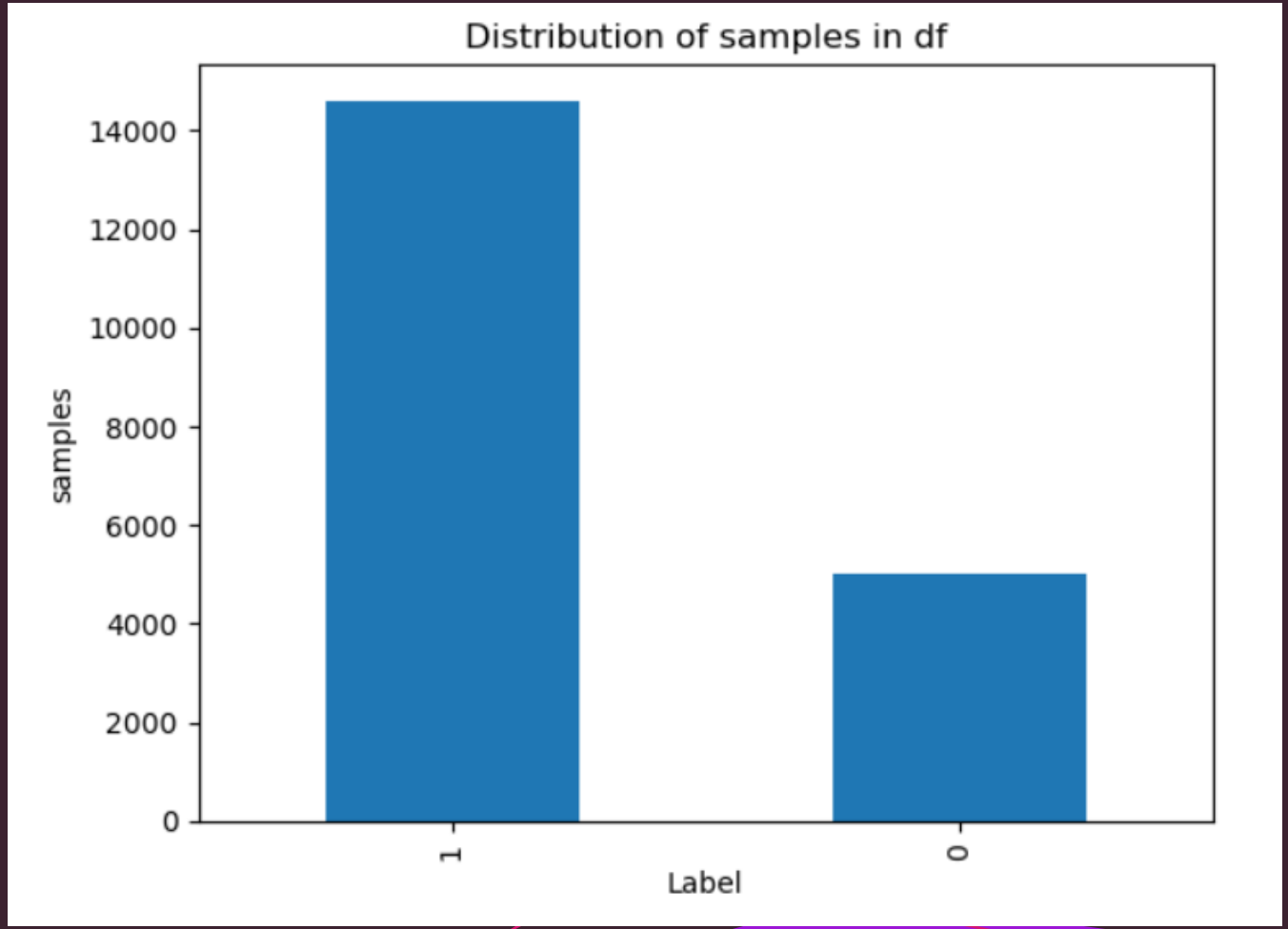
- The data contains 19611 rows, and 79 columns

0	Name	19611 non-null	object
1	e_magic	19611 non-null	int64
2	e_cblp	19611 non-null	int64
3	e_cp	19611 non-null	int64
4	e_crlc	19611 non-null	int64
5	e_cparhdr	19611 non-null	int64
6	e_minalloc	19611 non-null	int64
7	e_maxalloc	19611 non-null	int64
8	e_ss	19611 non-null	int64
9	e_sp	19611 non-null	int64
10	e_csum	19611 non-null	int64
11	e_ip	19611 non-null	int64
12	e_cs	19611 non-null	int64
13	e_lfarlc	19611 non-null	int64
14	e_ovno	19611 non-null	int64
15	e_oemid	19611 non-null	int64
16	e_oeminfo	19611 non-null	int64
17	e_lfanew	19611 non-null	int64
18	Machine	19611 non-null	int64
19	NumberOfSections	19611 non-null	int64
20	TimeDateStamp	19611 non-null	int64
21	PointerToSymbolTable	19611 non-null	int64
22	NumberOfSymbols	19611 non-null	int64
23	SizeOfOptionalHeader	19611 non-null	int64
24	Characteristics	19611 non-null	int64
25	Magic	19611 non-null	int64
26	MajorLinkerVersion	19611 non-null	int64
27	MinorLinkerVersion	19611 non-null	int64
28	SizeOfCode	19611 non-null	int64
29	SizeOfInitializedData	19611 non-null	int64
30	SizeOfUninitializedData	19611 non-null	int64
31	AddressOfEntryPoint	19611 non-null	int64
32	BaseOfCode	19611 non-null	int64
33	ImageBase	19611 non-null	int64
34	SectionAlignment	19611 non-null	int64
35	FileAlignment	19611 non-null	int64
36	MajorOperatingSystemVersion	19611 non-null	int64
37	MinorOperatingSystemVersion	19611 non-null	int64
38	MajorImageVersion	19611 non-null	int64
39	MinorImageVersion	19611 non-null	int64
40	MajorSubsystemVersion	19611 non-null	int64
41	MinorSubsystemVersion	19611 non-null	int64
42	SizeOfHeaders	19611 non-null	int64
43	CheckSum	19611 non-null	int64
44	SizeOfImage	19611 non-null	int64
45	Subsystem	19611 non-null	int64
46	DllCharacteristics	19611 non-null	int64
47	SizeOfStackReserve	19611 non-null	int64
48	SizeOfStackCommit	19611 non-null	int64
49	SizeOfHeapReserve	19611 non-null	int64
50	SizeOfHeapCommit	19611 non-null	int64
51	LoaderFlags	19611 non-null	int64
52	NumberOfRvaAndSizes	19611 non-null	int64
53	Malware	19611 non-null	int64
54	SuspiciousImportFunctions	19611 non-null	int64
55	SuspiciousNameSection	19611 non-null	int64
56	SectionsLength	19611 non-null	int64
57	SectionMinEntropy	19611 non-null	float64
58	SectionMaxEntropy	19611 non-null	int64
59	SectionMinRawsize	19611 non-null	int64
60	SectionMaxRawsize	19611 non-null	int64
61	SectionMinVirtualsize	19611 non-null	int64
62	SectionMaxVirtualsize	19611 non-null	int64
63	SectionMaxPhysical	19611 non-null	int64
64	SectionMinPhysical	19611 non-null	int64
65	SectionMaxVirtual	19611 non-null	int64
66	SectionMinVirtual	19611 non-null	int64
67	SectionMaxPointerData	19611 non-null	int64
68	SectionMinPointerData	19611 non-null	int64
69	SectionMaxChar	19611 non-null	int64
70	SectionMainChar	19611 non-null	int64
71	DirectoryEntryImport	19611 non-null	int64
72	DirectoryEntryImportSize	19611 non-null	int64
73	DirectoryEntryExport	19611 non-null	int64
74	ImageDirectoryEntryExport	19611 non-null	int64
75	ImageDirectoryEntryImport	19611 non-null	int64
76	ImageDirectoryEntryResource	19611 non-null	int64
77	ImageDirectoryEntryException	19611 non-null	int64
78	ImageDirectoryEntrySecurity	19611 non-null	int64

dtypes: float64(1), int64(77), object(1)

GETTING KNOW THE DATA

- Although The data is a bit biased to the “ Malware” cluster, we have enough data such that the model will be able to learn all the labels well

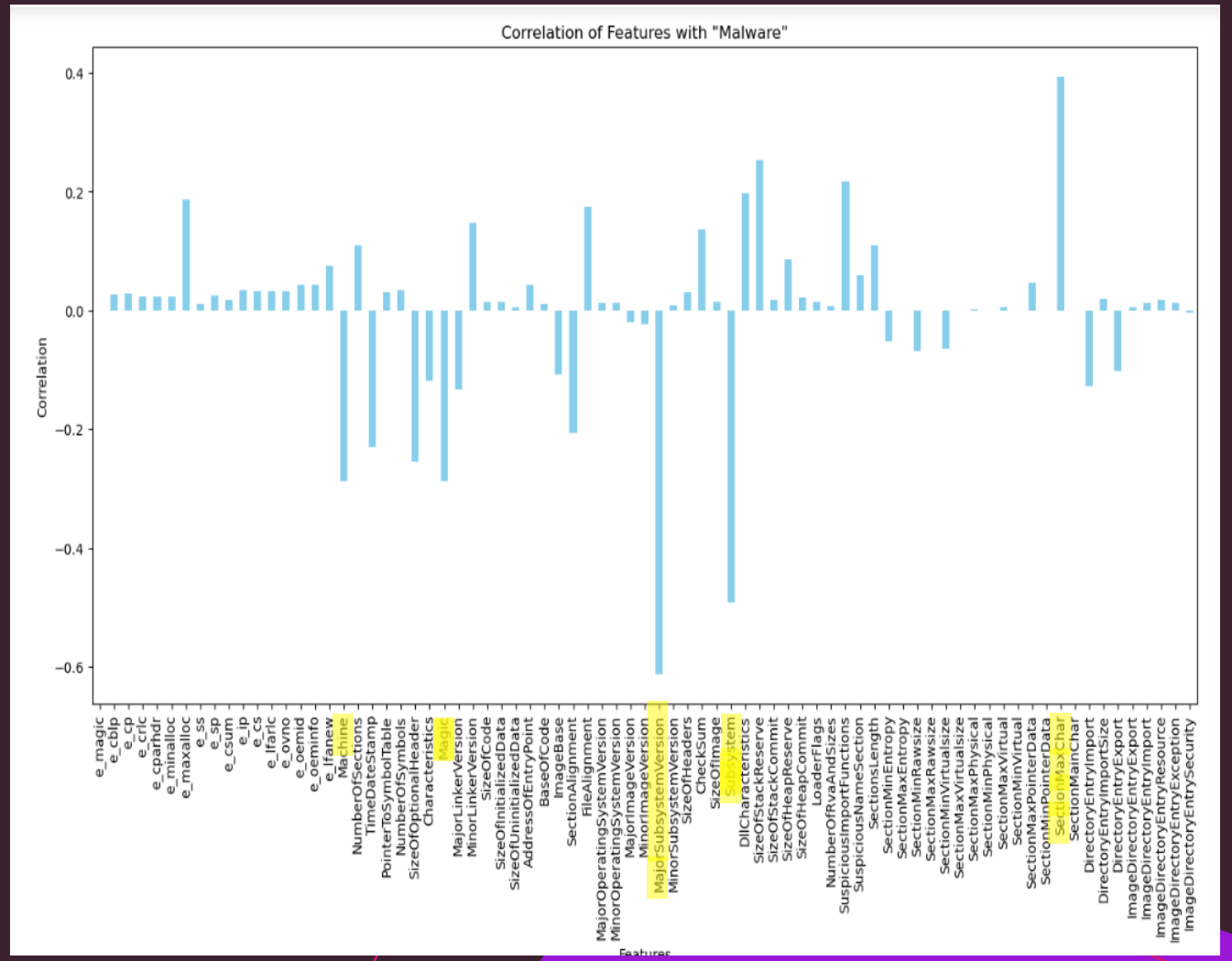


GETTING KNOW THE DATA

- We would like to check the correlation of the features to the label: “ Malware”, but still the correlations aren’t compromising

```
Top 5 Features with Highest Correlation to 'Malware':
SectionMaxChar                0.393282
SizeOfStackReserve            0.251791
SuspiciousImportFunctions     0.216656
DllCharacteristics            0.197023
e_maxalloc                    0.186079
Name: Malware, dtype: float64

Top 5 Features with Lowest Correlation to 'Malware':
MajorSubsystemVersion    -0.611621
Subsystem                -0.492813
Magic                    -0.287414
Machine                  -0.287413
SizeOfOptionalHeader     -0.255692
Name: Malware, dtype: float64
```



DIMENSIONALITY REDUCTION

- Following the research work - we used PCA to reduce the number of columns to 55, as determined by the researchers.
- Before that, we carried out our own research and found that in advance we could not refer to 4 columns ('Name', 'Machine', 'TimeDateStamp', and the target label 'Malware' that mustn't be reduced) that represent general or not significant information so that it does not constitute an impact on the data

```
print(df_pca.shape)
df_pca.head()
```

```
(19611, 56)
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	...	PC47	PC48	PC49	PC50
0	-0.121848	-0.109631	0.677826	1.666928	1.618763	-0.550915	0.724746	-0.201950	-0.829159	-0.149451	...	0.065499	0.062655	-0.133934	0.168381
1	-0.294837	-0.135038	0.612546	-0.003972	0.179251	0.200744	0.551912	0.277117	0.068057	-1.479715	...	0.201665	-0.068814	-0.228309	0.154042
2	-0.296454	-0.122306	0.340623	-0.655986	-1.082754	0.059316	-0.495534	-0.429369	0.711304	-0.653683	...	-0.023068	-0.009834	0.101132	-0.124272
3	-0.282446	-0.128331	1.502726	-0.618458	-0.902702	-0.011136	-0.228185	-0.291438	0.214832	1.020817	...	-0.105812	0.055736	0.166189	-0.151427
4	-0.085061	0.102229	0.840554	-0.730703	-1.359444	-0.167428	-0.583728	-0.231280	0.236922	1.143307	...	-0.152038	-0.007882	0.119603	-0.060771

PRE-PROCESSING

- In the last stage we split the data to trains and tests sets

```
X_train, X_test, y_train, y_test = train_test_split(df_pca.drop("Malware", axis=1), df["Malware"],
                                                    test_size=0.2, random_state=42)
print("X_train size:", X_train.shape)
print("y_train size:", y_train.shape,"\n")
print("X_test size:", X_test.shape)
print("y_test size:", y_test.shape)
```

```
X_train size: (15688, 55)
y_train size: (15688,)

X_test size: (3923, 55)
y_test size: (3923,)
```

ML MODELS

- In the first stage the researchers implemented 5 machine-learning models:
  1. Naive Bayes
  2. Decision Tree
  3. Random Forest
  4. AdaBoost
  5. Gradient Boosting
- We ran these models on the new df (with the 55 new features after the PCA dimensional reduction)
- Since we want to detect malware, we decided to present the scores in a recall metric, which is more relevant in these cases.

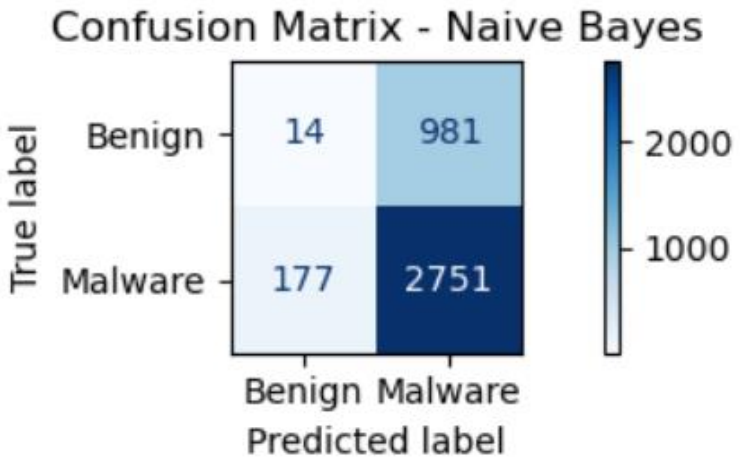
ML MODELS – RECALL SCORES

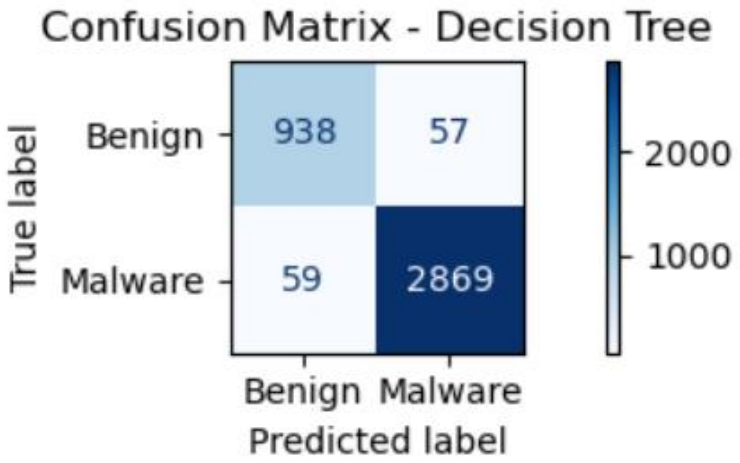
	Our scores	Researcher's scores
Naive Bayes	0.94	0.99
Decision Tree	0.98	0.95
Random Forest	0.991	0.98
AdaBoost	0.981	0.95
Gradient Boosting	0.988	0.93

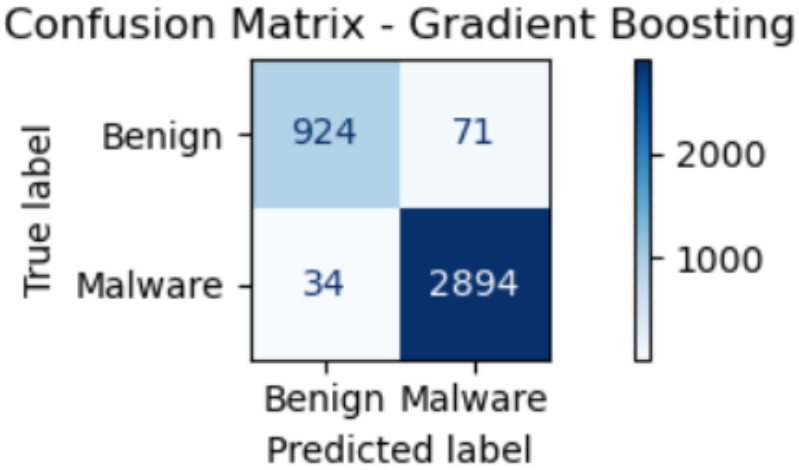
- Note – we saw that Naïve bayes returns a recall score of 0.06045, which means almost always wrong, so we decided that whatever this model says – we will take the opposite. Which changed the recall score to 0.94

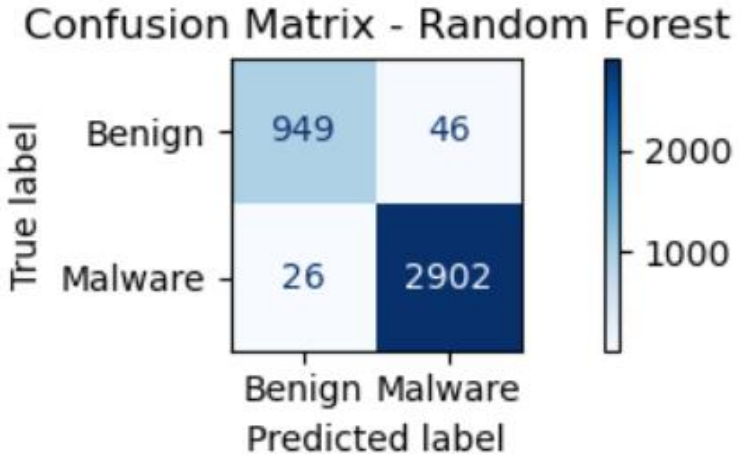
```
for idx, (clf_name, clf) in enumerate(classifiers):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    if clf_name == 'Naive Bayes':
        print(y_pred)
        y_pred = (y_pred+1)%2
        print(y_pred)
```

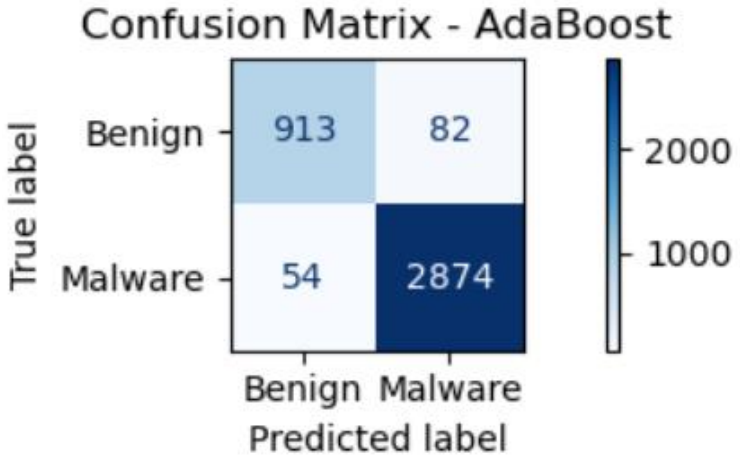
ML MODELS – CONF-MATRIX











DEEP LEARNING MODELS

- The next stage was implementing 3 DL models:
  1. MLP with 1 hidden layer with 50 neurons
  2. MLP with 2 hidden layers with (40,40) neurons
  3. 1D CNN model with 40 neurons and 60 filters

- in all experiments we used sparse categorical cross-entropy loss and Adam optimizer.
- 80% of the data was used for training and 20% for testing. The results are shown in Tables 4–6. We trained Dense-1 and Dense-2 models for 200 epochs, while the 1D-CNN model were trained for 50 epochs
- We checked with different learning rates but the changes were small.

'ADAM' OPTIMIZER

- Adam (Adaptive Moment Estimation) helps minimize the loss function by iteratively adjusting the learning rate for each parameter in the model based on past gradients by fast convergence (if the loss is high – the adjustment in the learning rate will be much more significant than if the loss is low)

DL MODELS - RESULTS

	Our scores	Researcher's scores
Mlp- 1 hidden layer	0.977	0.983
Mlp- 2 hidden layers	0.978	0.988
1D CNN	0.978	0.978

ENSEMBLE LEARNING

- In the last stage they implemented an ensemble learning model by implementing the previous 3 dl models as the first stage, and on top of these results – machine learning models were implemented as the final stage.

```
def train_models(X_train, y_train):
    # Define base classifiers
    dense1 = Sequential([
        Dense(55),
        PReLU(),
        Dropout(0.3),
        Dense(2, activation='softmax')
    ])

    dense2 = Sequential([
        Dense(64),
        PReLU(),
        Dropout(0.3),
        Dense(32),
        PReLU(),
        Dropout(0.3),
        Dense(2, activation='softmax')
    ])

    D1_CNN = Sequential([
        Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(55, 1)),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=64, kernel_size=3, activation='relu'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(55),
        Dropout(0.5),
        Dense(2, activation='softmax')
    ])
```

```python
def test_ensemble(X_test, y_test, new_features):
    # Define final-stage classifiers
    classifiers = {
        'Decision Tree': DecisionTreeClassifier(),
        'SVM (Linear Kernel)': SVC(kernel='linear', probability=True),
        'SVM (RBF Kernel)': SVC(kernel='rbf', probability=True),
        'Random Forest': RandomForestClassifier(),
        'AdaBoost': AdaBoostClassifier(),
        'Extra Trees': ExtraTreesClassifier(),
        'KNN': KNeighborsClassifier(),
        'Gaussian NB': GaussianNB(),
        'LDA': LinearDiscriminantAnalysis(),
        'QDA': QuadraticDiscriminantAnalysis(),
        'Logistic Regression': LogisticRegression(),
        'Passive Aggressive': PassiveAggressiveClassifier(),
        'Ridge Classifier': RidgeClassifier(),
        'SGD': SGDClassifier(),
        'Isolation Forest': IsolationForest()
    }
```

ML MODELS – RECALL SCORES

	Our scores	Researcher's scores
Decision Tree	0.99981	0.989
Random Forest	0.99981	0.984
Extra Trees	0.99981	1
KNN	0.98266	0.986
LDA	0.97705	0.98
AdaBoost	0.97673	0.982
SVM RBF	0.97654	0.982
Ridge Classifier	0.97654	0.981
SVM Linear	0.97642	0.979
Logistic Regression	0.97642	0.981
SGD	0.97508	0.979
Passive Aggressive	0.97444	0.978
Gaussian NB	0.97291	0.972
QDA	0.96577	0.973

ENSEMBLE LEARNING

- On top of these data, we tried to flip the order of the stages, and use ml models, and send the output features to dl models but it didn't beat our previous model.
- In addition we tried to run on the 3 models that achieved the best scores (Random forest, Extra trees, Decision tree) with grid search to find the best parameters. But again the scores didn't improve

```
param_grid_dt = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2', None]
}

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

param_grid_et = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

```
Best parameters for Decision Tree: {'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_samples_split': 2}
Best parameters for Random Forest: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 200}
Best parameters for Extra Trees: {'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 100}
Best score for Decision Tree: 0.9778174029699187
Best score for Random Forest: 0.9834266601082945
Best score for Extra Trees: 0.9832991294309392
```

CONCLUSIONS

- We explored using machine learning and neural networks to improve malware detection possibilities. Our system achieved better accuracy than traditional methods by automatically learning from data but requires labeled data which may not always be available.
- Our scores do not beat the researchers' best model score, since according to the article the researchers stated that they reached an accuracy of 1.
- However we gained results which is almost certain identification – 99.98%.