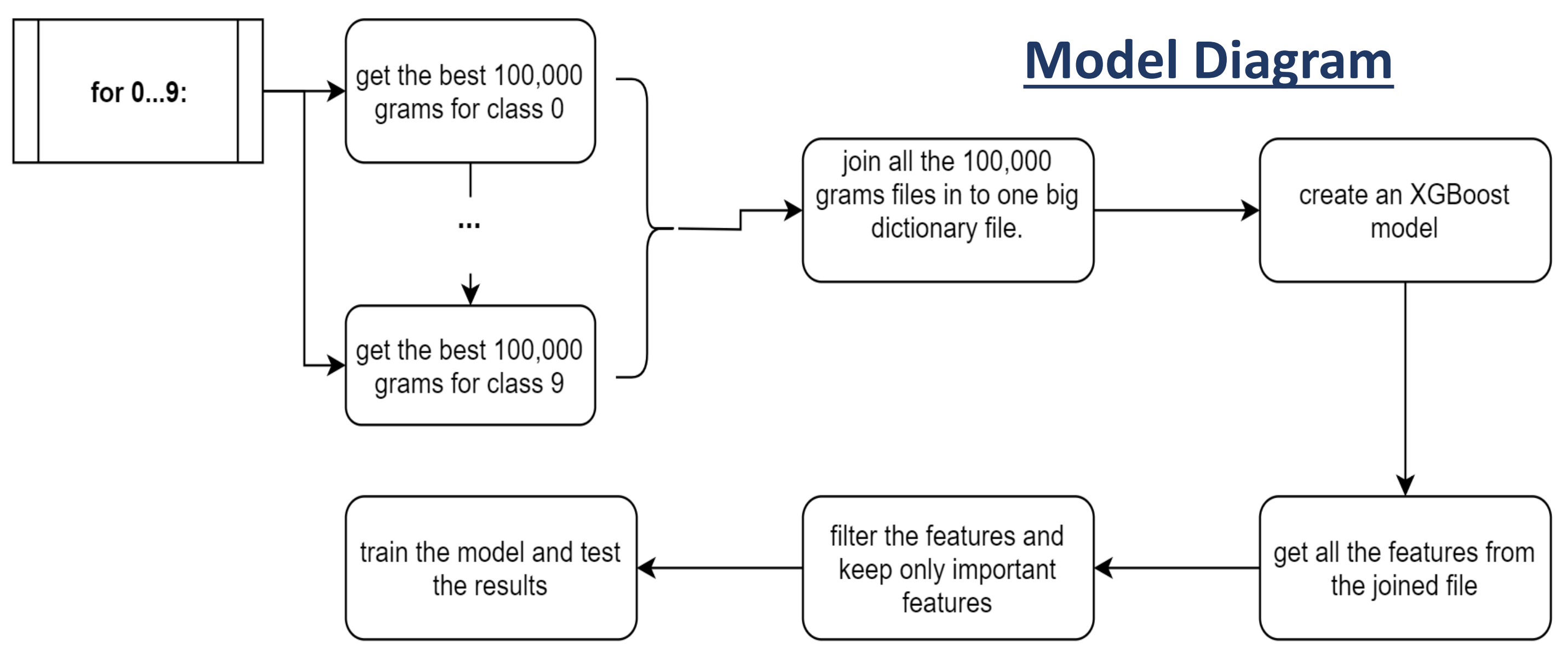
Malware Classification

Yarden Cohen and Haim Gil
Supervisor: Assaf Barak

Machine Learning using XGBoost

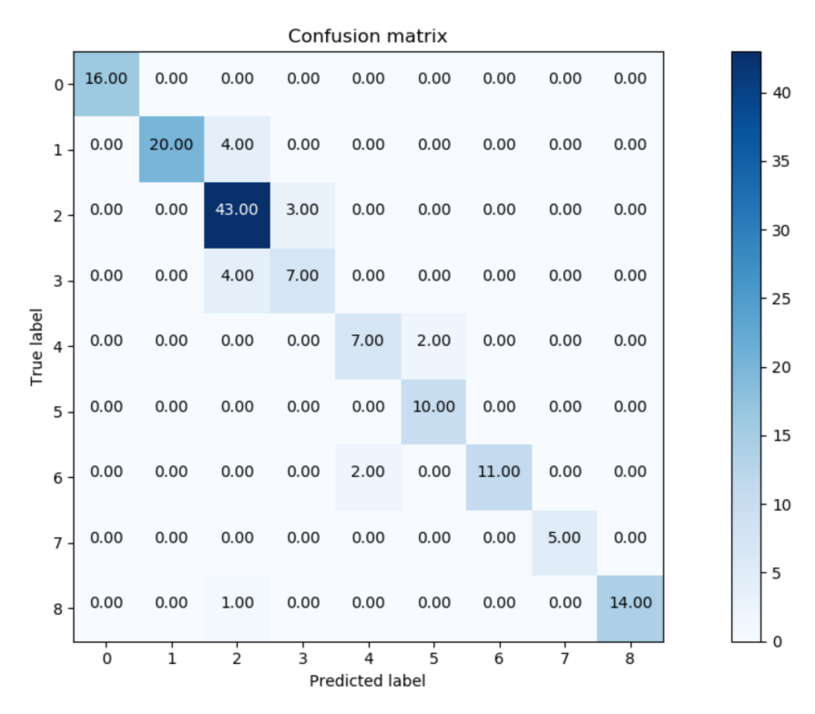
- We implemented a n-grams machine learning model between 8 classes of malware files and one class of benign files.

Model Diagram

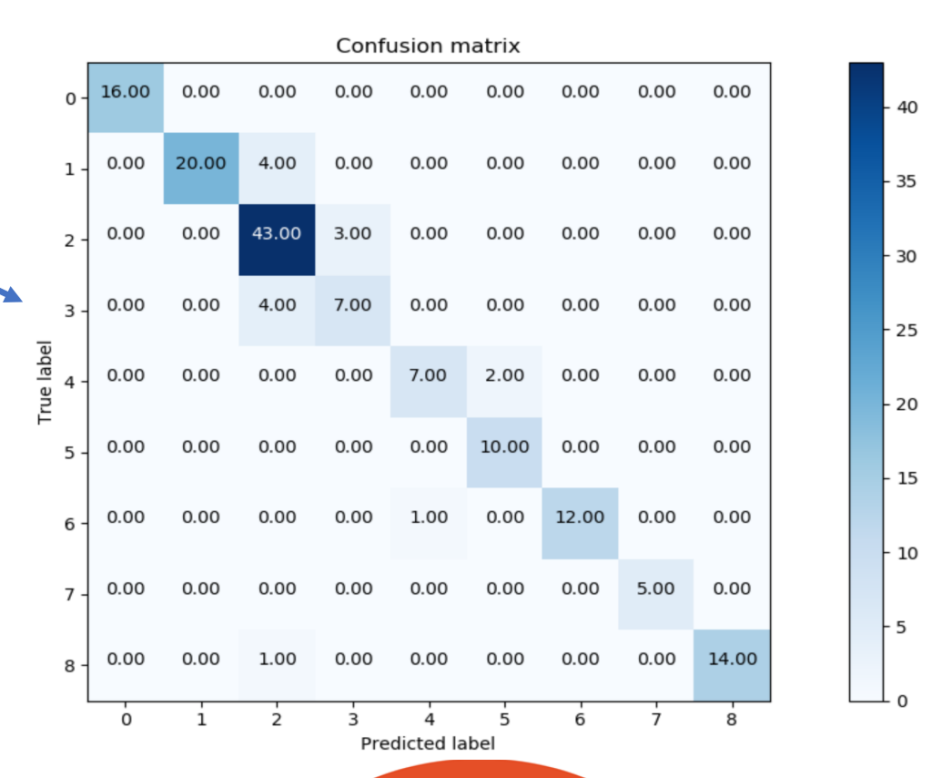

for 0…9:
get the best 100,000 grams for class 0
…
get the best 100,000 grams for class 9
join all the 100,000 grams files in to one big dictionary file.
create an XGBoost model
get all the features from the joined file
filter the features and keep only important features
train the model and test the results

- There are 4 steps in our model:
  1. Go through each class and keep the most 100,000 common features.
  2. Merge all of the saved features from all the classes.
  3. Get the features filtered according to their significance.
  4. Train and test the XGboost model with these features.

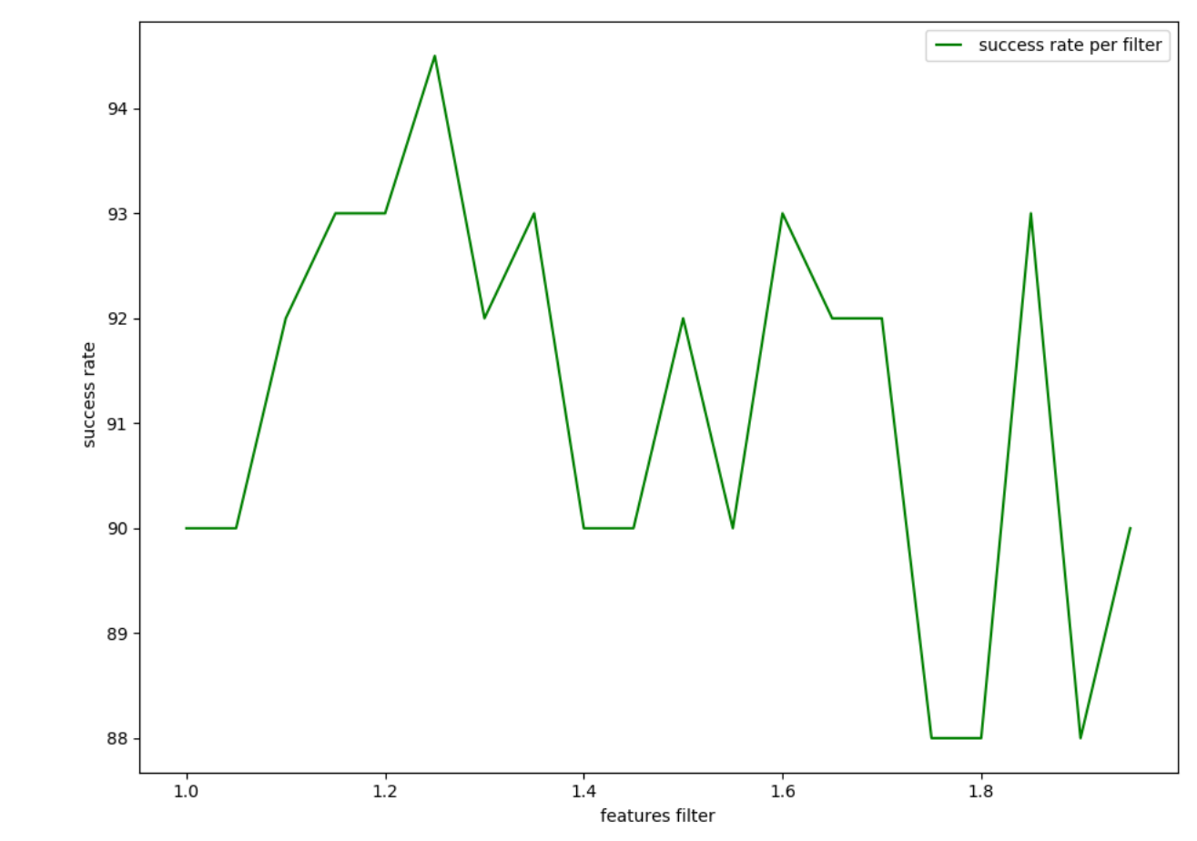
- We got 88-89% with this confusion matrix.



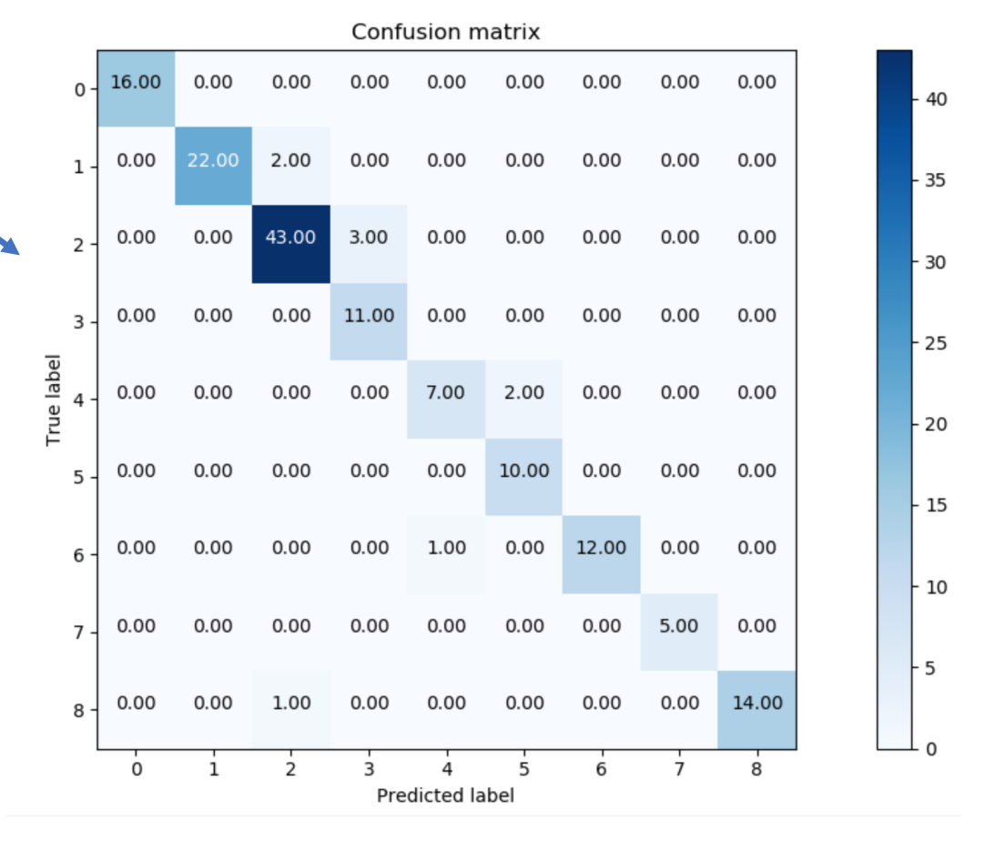
- As seen, we have 100% with the binary classification. We adjusted to play with our parameters and discovered that the filter size of the features has most impact on our results.
- We found that the RandomClassifier has a property, called "feature importance" and after RF.fit() we can get an array with the importance of each feature.
- We found the mean of this array and checked for each feature to see if its importance is smaller than the mean We deleted this feature. This raised our results to 90% with this confusion matrix.



- We asked ourselves how much to multiply the mean to get the best results. So we printed a graph of the results of our model per x (when x is how much to multiply the mean in the filter) and got this result.



- Maximum in filter = 1.25 got 94% success.
- Meaning take for all the features, if this feature's importance is smaller than 1.25 multiply the mean of all features importance – delete this feature.
- That filter brought us this confusion matrix.

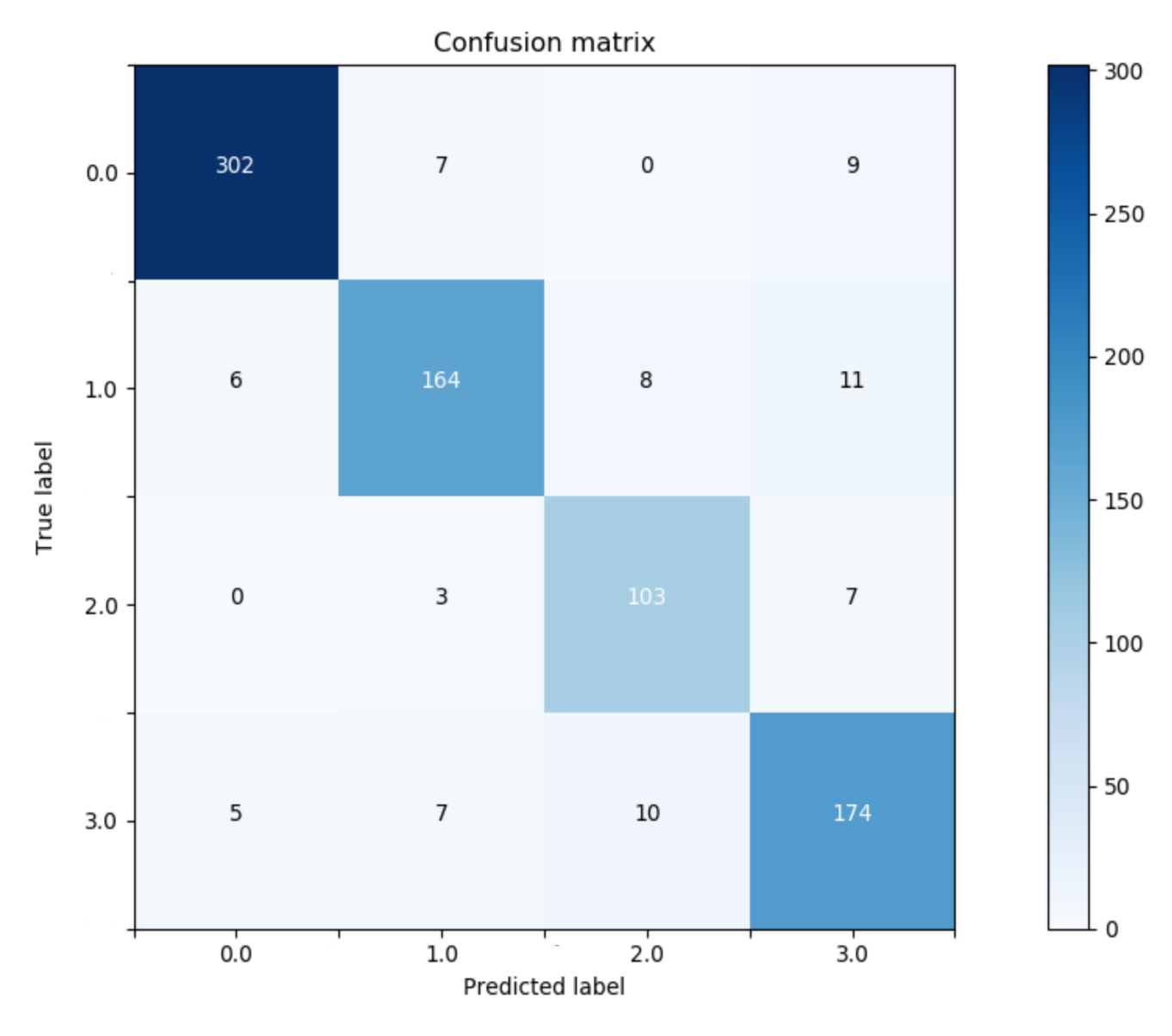


Deep Learning using PyTorch

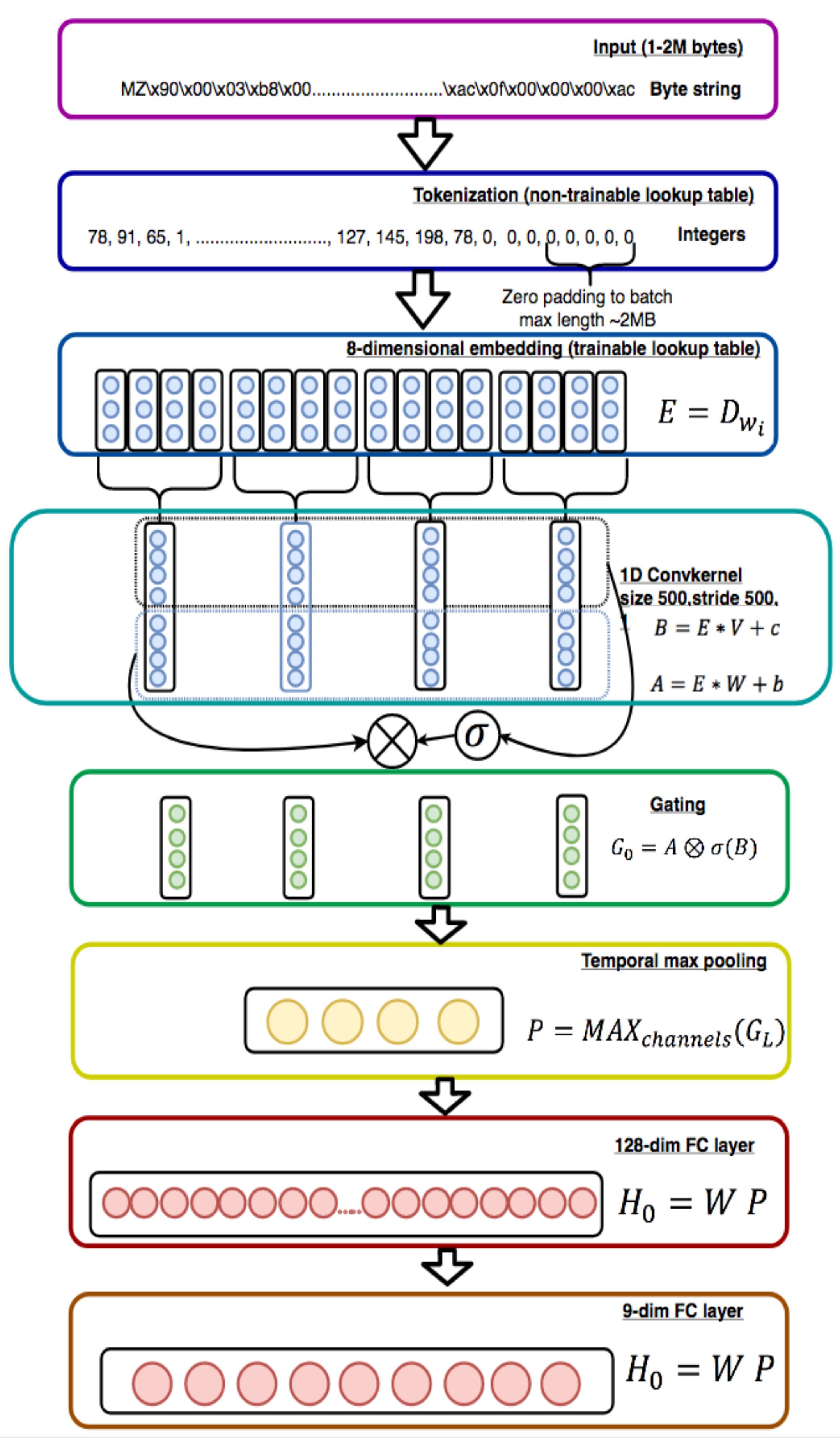
- We implemented deep learning model according to Edward Raff's paper: Malware Detection by Eating a Whole EXE. We expanded this model to classify between 3 classes of malware and one class of benign, achieving 90% accuracy.
- We used an embedding layer to map each byte to a fixed length feature vector.
- Training the embedding jointly with the convolution allows even our shallow network to activate for a wider breadth of input patterns. This also gives it a degree of robustness in the face of minor alterations in byte values. Prior work using byte n-grams lack this quality, as they are dependent on exact byte matches.
- **Improvements**:
- Initially, each epoch took two hours to learn the features so we added GPU support to our model using CUDA to reduce the running time to 8-9 minutes.

Future work:
We would like to run our model on a cluster with 10-20 machines to reduce the running time even more. To do this, we need to support it in our code. Additionally, we would like to integrate with Spark. Spark keeps data in-memory using RDD. It is very efficient when dealing with machine learning algorithms because memory is much faster than disk which will lead to significant shorting during runtime.

Confusion Matrix:



Model Diagram


Input (1-2M bytes)
MZ\x90\x00\x03\xb8\x00……………………..\xac\x0f\x00\x00\x00\xac
Byte string
Tokenization (non-trainable lookup table)
78, 91, 65, 1, ………………………, 127, 145, 198, 78, 0, 0, 0, 0, 0, 0, 0, 0
Integers
Zero padding to batch max length ~2MB
8-dimensional embedding (trainable lookup table)
$E = D_{w_i}$
1D Convkernel size 500,stride 500,
$B = E * V + c$
$A = E * W + b$
σ
Gating
$G_0 = A \otimes \sigma(B)$
Temporal max pooling
$P = MAX_{channels}(G_L)$
128-dim FC layer
$H_0 = W P$
9-dim FC layer
$H_0 = W P$