ROBUST MALWARE DETECTION USING IMAGE PROCESSING

Project Lab

Author: Turtogtokh Altangerel

Supervisor: Ács Gergely

Introduction

Malware is one of the most serious security threats nowadays as it has shown steady growth in terms of not only volume but also complexity and diversity. To effectively deal with this threat, malware detection systems have to be improved drastically and most importantly be trained through the means that surpass human performance. This being the reason for this project's main goal suggests the use of machine learning so as to create an opportunity for a malware detector to learn efficiently and rapidly on its own.

My proposal to aid the counter measurement against malware is Robust Malware Detector. It is a malware detector based on Convolution Neural Network, commonly used for image processing, and is effective against adversarial malware, which fool non-robust detectors. To complete this project, main steps should be separated, executed and tested individually.

Main objectives

- Processing raw benign and malware files, essentially transforming a file to a ready-to-use image for the model while retaining its features
- Training a CNN model on the extracted dataset and testing its performance
- Improving the robustness of the trained model against generated adversarial images

Technologies used:

- Python (Interpreted, object-oriented, high-level programming language with dynamic semantics.)
- Keras (High-level deep learning API developed for implementing neural networks.)
- ART (Adversaryal robustness toolbox, used to train a model's robustness)
- matplotlib, visualkeras (Python libraries used for visualization purposes)

User Guide

There are four python source files, which can be downloaded and placed in the same directory.

sample_prepare.py implements the transformation of binary files into grey-scale images with corrected shape.

training.py implements the all the necessary actions from loading dataset, creating a model, training the model to saving the model. Also it visualizes performance measurements.

summary.py offers functions for plotting and visualizing.

adversarial_training.py implements the usage of ART classifier to improve the robustness of the model.

Getting started with the project

1) It is recommended to start with **sample_prepare.py**, where the directory path to raw binary files should be provided in the beginning.

2) After acquiring processed files, which will be saved as numpy arrays, use **training.py** to train the CNN model. The visualization of the model as well as its performance on the dataset will be displayed and saved.

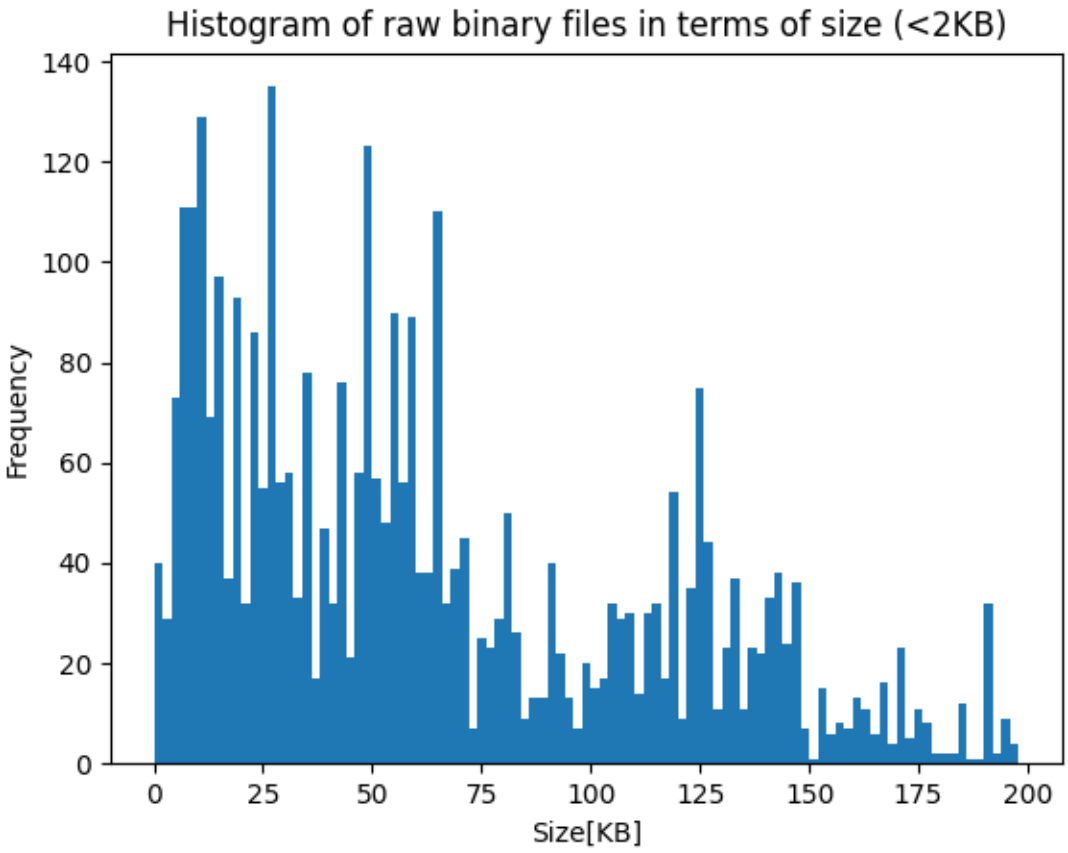
3) Lastly, in order to improve the accuracy of the model against malwares hidden as benign, the user can use **adversarial_training.py**, where the trained model should be provided. The trained model can be one acquired in the previous step or **trained_model.h5**, which can be found in this repository.

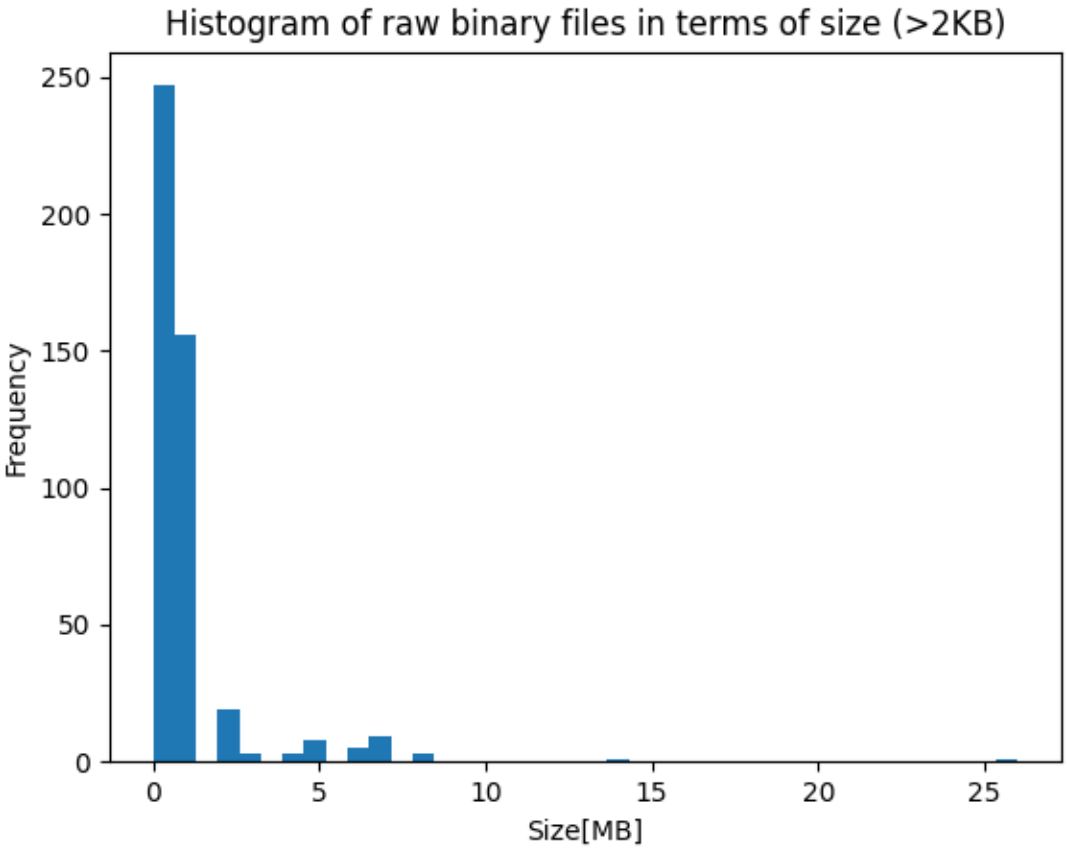
Demonstration

To describe the usage of the project precisely, this section demonstrates the process step by step.

Dataset processing

We start with dataset of almost 4000 binary files, which has equal numbers of malware and benign files. However, the range of the sizes of the files is high, as illustrated below:





The above plots depict that some files are relatively small in size while some are quite large. Therefore, we need to resize the image to a fixed shape, which is the same as the input shape of the CNN model. In our implementation, the input shape is 128*128 pixels.

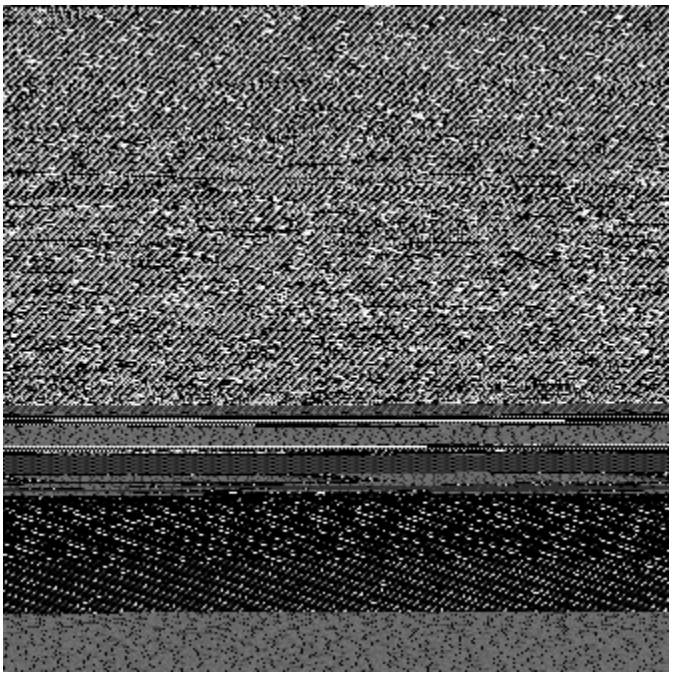
sample_prepare.py resizes the binary files by averaging, which is a great method since we need to retain the features of the images. For an example, a file of size 32768 bytes, which is twice larger than the needed size, can be downsized with a pooling size of 2 bytes. In this way, each neighboring pixel in a pair is averaged and taken as a single pixel.

sample_prepare.py is able to reshape any file to any requested size while retaining most information as possible.

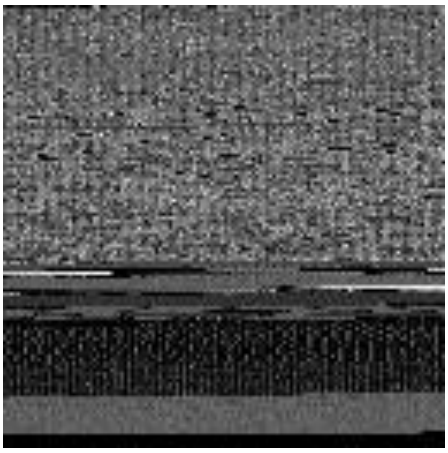
Also, it is worth mentioning that the images are re-shaped to fit the CNN model, which requires a 4-D image as an input, and normalized pixel values to a range from 0 to 1, which results in easier computations.

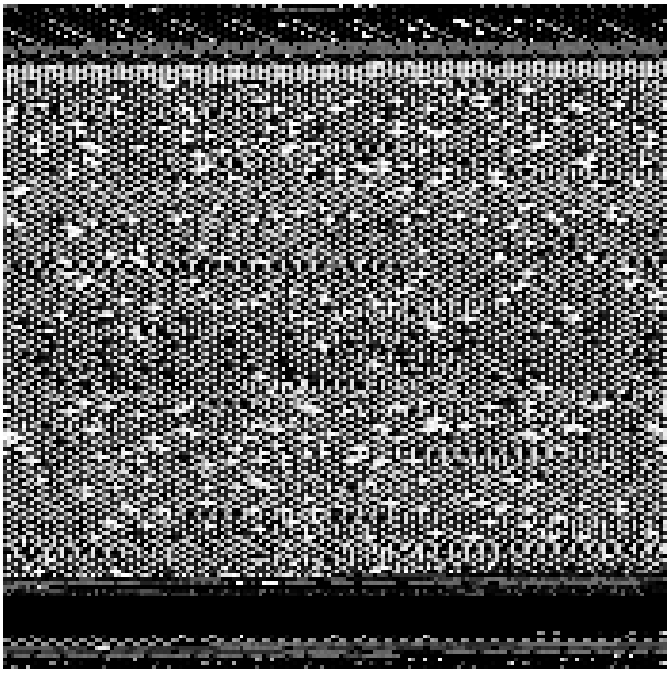
Result (raw malware & processed malware)

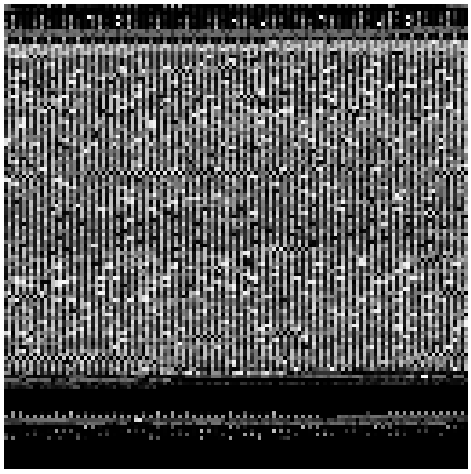
Before averaging



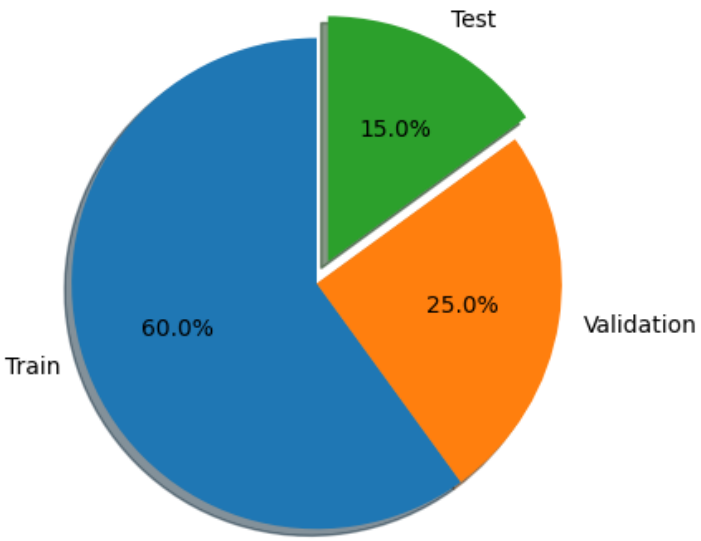
After averaging







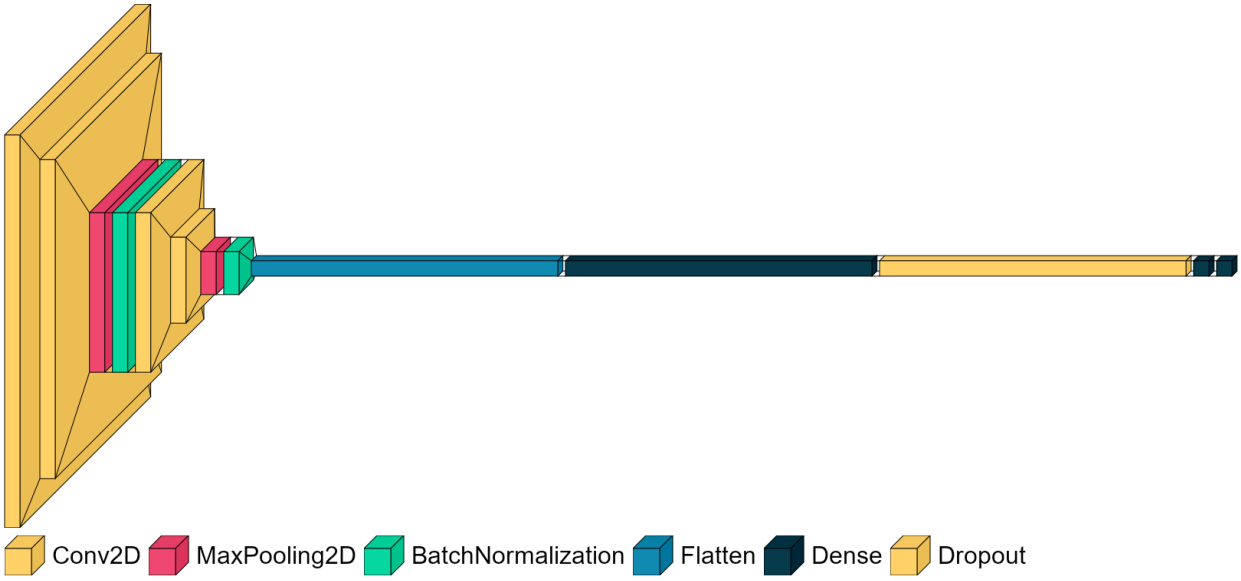
We can see that images have similar features.



After the successful transformation of binaries to images, the dataset is split into three sections as seen in the chart above. Training set is used to train the model while the validation set is used to check the performance of the specific training at one epoch. Lastly, the model is tested on unseen Test data. The performance on test data plays the most important role in defining the model. If the model shows good performance with training and validation sets but shows bad performance with testing data, it is said that the model is overfitting. Overfitting is one of the most common issues with training machine learning models. In order to reduce this negative effect, many techniques can be used and tested, such as removing some neurons in the network to make it more difficult for the model to adapt to training dataset.

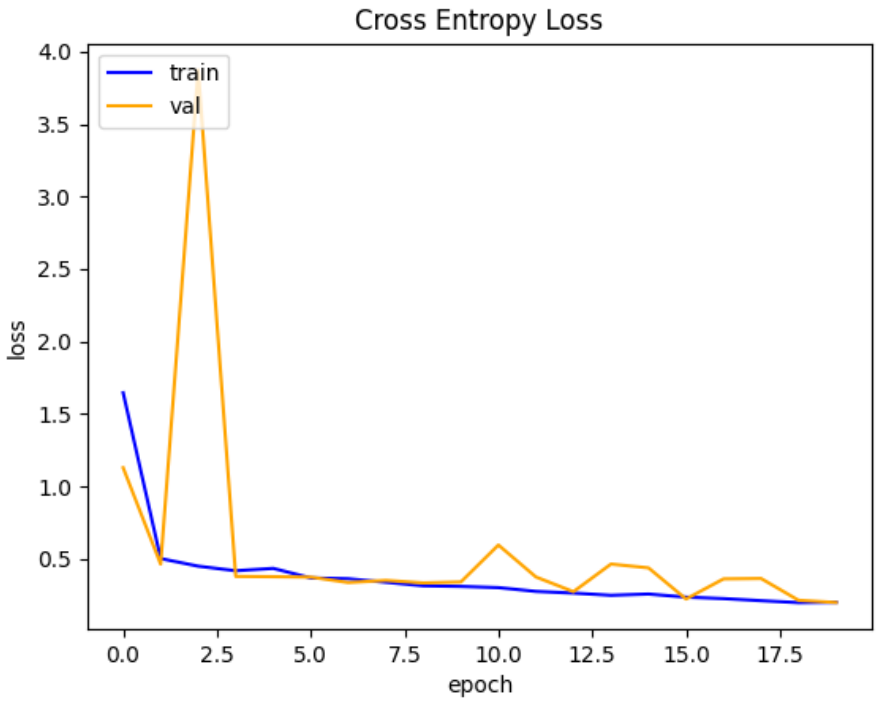
Training the model

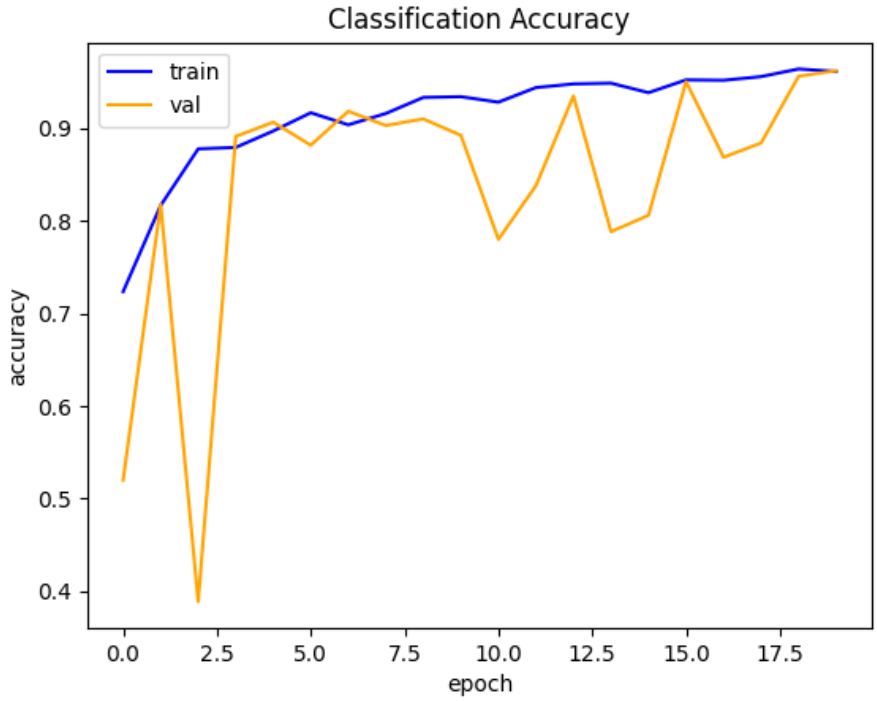
In this demonstration, the CNN model has 2 hidden layer, each of which consists of 2 convolutional layers with 32 filters, 16x16 kernel size, Relu activation function. Convolutional layers extract features from the images. Max Pooling layer is used to downsize the sample for consecutive layers. BatchNormalization layer re-scales the values improving the training rate and stability while Dropout blocks random neurons improving the model's performance on unseen dataset.



The model is trained on training dataset, validated at each epoch with validation dataset, and finally tested with test dataset.

Result

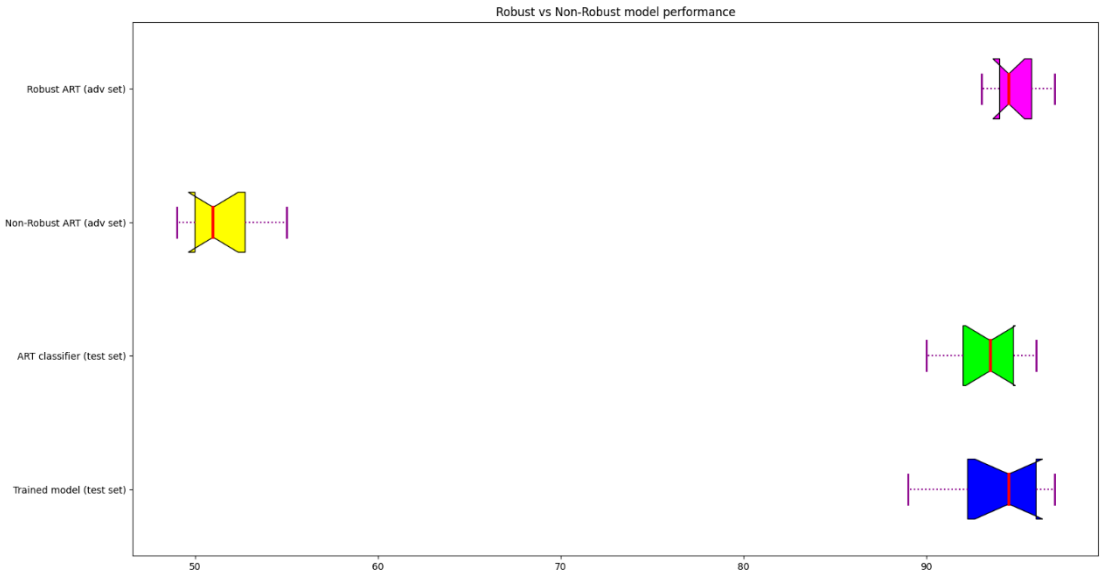




From the plots above, it can be seen that validation curve gradually follows the training curve. The fluctuations of validation curve may possible be due to not large enough validation dataset.

Evaluation of Robust and Non-Robust models

Finally, the trained model can be trained against robustness with ART tooblox. The graph below shows four performance versions of the same model.



It can be seen the model has not less than 95° accuracy in all cases except that the Non-Robust model is practically ineffective against adversarial images. Therefore, it is quite necessary to train malware detectors to be robust against such complex threats.

Credits

Similar work:

Classifying Malware Images with Convolutional Neural Network Models

Useful materials:

CNN based malware detection (python and TensorFlow

How to Develop a CNN for MNIST Handwritten Digit Classification

Special thanks:

Ács Gergely, Crysys Lab, BME