Bahareh Arghavani Midterm report

This section focuses on the data-related aspects of your project based on the initial steps described in your notebook. If you have implemented further data augmentation, feature engineering, or any other preprocessing techniques beyond those mentioned, please share additional details for a more comprehensive overview.

Preprocessing and Data Normalization

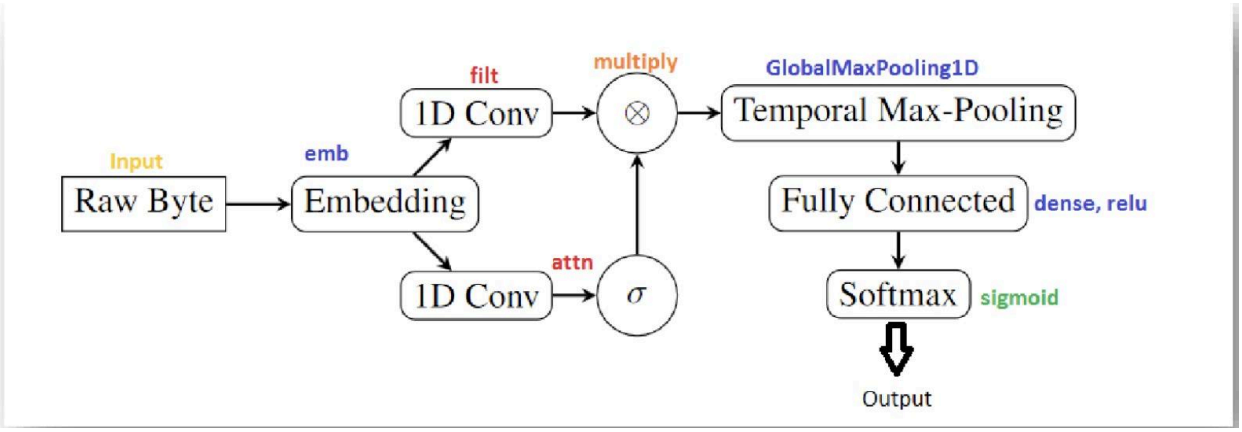
The preprocessing phase of the project plays a critical role in preparing the EMBER-2017 v2 dataset for the deep learning model, ensuring that the data is in a suitable format that maximizes the model's ability to learn and make accurate predictions. One of the key steps in this phase is the application of MinMaxScaler, a normalization technique that adjusts the features' values to a common scale without distorting differences in the ranges of values. This is particularly important for machine learning models, as it helps in speeding up the convergence during training by ensuring that all features contribute equally to the model's learning process. By scaling the dataset's features to a fixed range between 0 and 1, the MinMaxScaler effectively mitigates the issue of certain features dominating the learning process due to their larger magnitude, leading to a more balanced and fair learning environment for all features.

The decision to exclude reshaping steps that would create additional channels indicates a thoughtful consideration of the model architecture and the nature of the input data. Originally, such reshaping might be considered to adapt data for models that expect input in a certain dimensionality or channel configuration, such as Convolutional Neural Networks (CNNs) typically used in image processing tasks. However, for this project, focusing on the malware classification task with the MalConv architecture, the preprocessing strategy has been optimized to maintain the data in a format that directly complements the model's input requirements. This approach simplifies the data preparation process and ensures that the model receives the input in a manner that is most conducive to learning from the features extracted from PE files. The elimination of unnecessary

reshaping steps demonstrates a nuanced understanding of both the dataset and the chosen model architecture, streamlining the preprocessing pipeline for efficiency and effectiveness.

Model Architecture and Training

The core of the project is centered around the MalConv model, a specialized neural network designed for the classification of Portable Executable (PE) files as either malware or benign. The model architecture is implemented using PyTorch and is constructed to handle the unique characteristics of malware data efficiently. At its foundation, the model employs an embedding layer designed to transform the raw numeric representations of the PE files into a dense vector space, facilitating the model's ability to learn complex patterns within the data. This is followed by two convolutional layers with a novel gating mechanism in between, which serves to modulate the feature maps produced by the convolutional layers, enhancing the model's capacity to focus on the most relevant features for malware detection. The use of a global max pooling layer further aids in distilling the most salient information from the feature maps into a form that is conducive to classification. Finally, the model comprises fully connected layers that culminate in a sigmoid activation function, outputting a probability score indicative of whether the input PE file is malware or benign.



Training Procedure and Dataset Handling

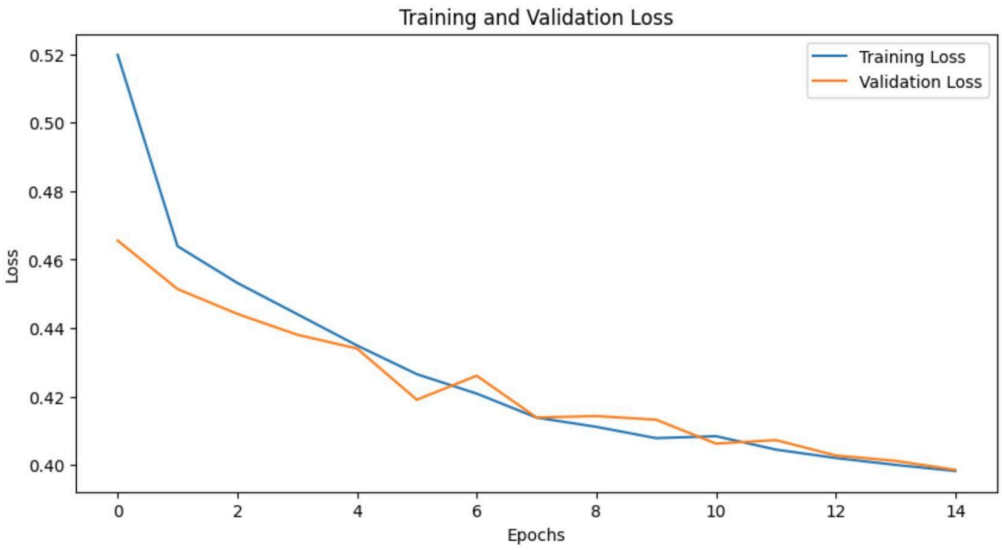
For training the MalConv model, the preprocessed dataset was first converted into PyTorch tensors, ensuring compatibility with the PyTorch framework. The data was then split into training and validation sets, adhering to the conventional 80/20 partition. This split not only facilitates the model's learning process but also enables the evaluation of its generalization capabilities on unseen data. The training was conducted over multiple epochs, utilizing the Adam optimizer for adjusting the model's weights and the Binary Cross-Entropy (BCE) loss function as the criterion for measuring the model's performance. This setup aims to minimize the discrepancy between the predicted and actual labels, thereby enhancing the model's accuracy in malware detection. Throughout the training process, both training and validation losses were monitored, providing valuable insights into the model's learning progress and allowing for the early detection of issues such as overfitting.

Evaluation, Optimization, and Architecture Visualization

The training routine incorporated a systematic approach to model evaluation and optimization, with checkpoints saved at regular intervals. This practice not only safeguards the training progress but also enables the exploration of the model's performance across different stages of training. The plotted training and validation loss curves serve as a crucial diagnostic tool, helping to visually assess the model's convergence and the effectiveness of the training strategy. Such insights are instrumental in making informed decisions about potential adjustments to the model architecture or training procedure, with the ultimate goal of achieving optimal performance in malware classification.

For a visual representation of the MalConv architecture, let's generate an architecture diagram that captures the essence of its design, including the embedding layer, convolutional layers with gating mechanism, global max pooling, and fully connected layers, culminating in the output layer with a sigmoid function.

```
Epoch 14, Training Loss: 0.40004609303271516, Validation Loss: 0.40121592960115204
Epoch 15, Training Loss: 0.3982520307632203, Validation Loss: 0.39857483162718305
Model checkpoint saved to ./model_checkpoints/model_epoch_15.pt
```



Model Performance Evaluation

Upon concluding the model's training phase, an extensive evaluation was conducted using a held-out test dataset to assess the MalConv model's capability in accurately classifying PE files as either benign or malware. This evaluation phase is critical, as it provides insights into the model's generalization ability on unseen data, a key indicator of its practical utility in real-world scenarios. The test data was methodically prepared by converting it into PyTorch tensors and grouped into batches using a DataLoader, ensuring an efficient evaluation process. The model was then switched to evaluation mode, a necessary step to disable training-specific operations like dropout, thus ensuring the outputs reflect the model's true predictive power under operational conditions.

The core of the evaluation rested on computing three fundamental metrics: accuracy, precision, and recall. These metrics were selected for their relevance in the context of malware detection, where the cost of false positives (benign files misclassified as malware) and false negatives (malware files not detected) can be significant. **Accuracy** provides a high-level view of the model's overall performance, indicating the proportion of total predictions (both positive and negative) that were correctly identified. **Precision** focuses on the model's reliability in identifying malware files, highlighting the ratio of true positive predictions in all positive predictions made. **Recall**, on the other hand, measures the model's ability to detect all actual malware instances, crucial for minimizing the risk of undetected threats. Together, these metrics offer a comprehensive evaluation of the model's effectiveness in malware classification, guiding further refinement and optimization efforts to enhance its performance.

```
Test Accuracy: 0.7373
Precision: 0.6965
Recall: 0.8413
```

Task 2: Deploying the Model as a Cloud API using Amazon SageMaker

Initial Setup and Environment Preparation

The notebook initiates with the setup and verification of the required libraries in the SageMaker environment. This step ensures that all necessary Python packages, such as Torch, are installed and accessible within SageMaker's Python environment. Furthermore, it includes the installation of the Ember project directly from GitHub, which is crucial for processing PE files in preparation for inference with the MalConv model.

Model Definition and Training Check

Following the environment setup, the notebook defines the MalConv model architecture consistent with the model trained earlier. This architecture is pivotal for ensuring that the deployed model in SageMaker mirrors the one used during training. Additionally, the notebook includes steps for loading a trained model checkpoint, signifying that the model's weights, trained on your local setup or another environment, can be successfully loaded and utilized within SageMaker.

Preprocessing and Model Inference Functions

The notebook proceeds to define essential functions for preprocessing PE files and performing inference with the loaded model. These functions are crucial for converting raw PE files into a format compatible with the MalConv model, allowing for the classification of files as either benign or malware based on their features. This setup demonstrates a clear pathway from raw input files to actionable malware classification within the cloud environment