FUNDAMENTALS OF COMPUTER VISION
VIOLENCE CLASSIFICATION

Berk Bubuş
21945939

İrem Zehra Altun
2220356186

Mehmet Akif Özgür
2200356832

Video Link

Drive Link

1 INTRODUCTION

Detecting violence in various forms, such as physical altercations or aggressive behavior, is crucial for maintaining safety and security in society. The aim of violence detection systems is to automatically identify instances of violence in real-time using advanced technologies like computer vision and machine learning. These systems analyze visual and auditory cues from surveillance cameras, social media platforms, or other sources to recognize violent actions or events. The motivation behind developing violence detection technology stems from the pressing need to prevent and respond to violent incidents effectively. By swiftly detecting and alerting authorities to potentially dangerous situations, these systems can help mitigate the impact of violence and en- hance public safety. Furthermore, in contexts like law enforcement, security monitoring, or crowd management, the ability to identify and intervene in violent situations promptly is paramount for maintaining order and preventing harm. Additionally, violence detection systems can aid in the monitoring and enforcement of policies and regulations aimed at curbing violence in various settings, including public spaces, schools, or on- line platforms. By providing valuable insights into patterns of violent behavior and facilitating data-driven decision-making, these systems contribute to the development of proactive strategies for violence prevention and intervention. In summary, the aim of violence detection technology is to automatically identify instances of vi- olence using advanced technologies like computer vision and machine learning. The motivation behind this technology is to prevent and respond to violent incidents effectively, enhance public safety, and enforce policies aimed at curbing violence.

2 METHOD

We utilized the Video Classification method to address the Violence/Non-Violence Classification problem. To apply this method, we gathered labeled data, perform pre-processing operations on the data, select an appropriate model for our method, and train this model. Then we tested the trained model with data that is not used for training and evaluate the succession of our model on video classification tasks using various metrics such as accuracy, precision, recall, and f1 score. Afterwards, we tested again using videos that we found ourselves and that were not present in the dataset, by manually checking these videos. Let's take a closer look at the processes we will undertake for the Video Classification method:

2.1 Data Collection

For our violence classification project, we created a dataset comprising short video clips labeled as either violence or non-violence. The dataset was meticulously gathered from various publicly available sources such as YouTube videos To ensure diversity and representativeness, we collected videos depicting a wide range of violent and non-violent activities across different contexts and scenarios. These activities encompassed physical altercations, aggressive behavior, sports-related confrontations, as well as non-violent interactions and daily activities.

2.2 Data Pre-Processing

Next, we performed some pre-processing on the dataset. First, we read the video files from the dataset and resize (h:224, w:224) the frames of the videos to a fixed width and height, to reduce the computations and normalized the data to range [0-1] by using the normalize function of albumentation module, which makes convergence faster while training the network.

2.3 Model Selection

In this project, we used the pretrained ResNet-101 model, which has proven to be an important model in classification tasks, as the backbone. We passed our data through the ResNet to extract feature maps. Next, we fed the obtained feature maps into an LSTM layer. Finally, we took the output from the LSTM layer and passed it through a fully connected layer to perform the classification. Throughout this process, we set the initial learning rate to 1e-5. We used Adam as the optimizer and Binary Cross-Entropy Loss as the loss function.

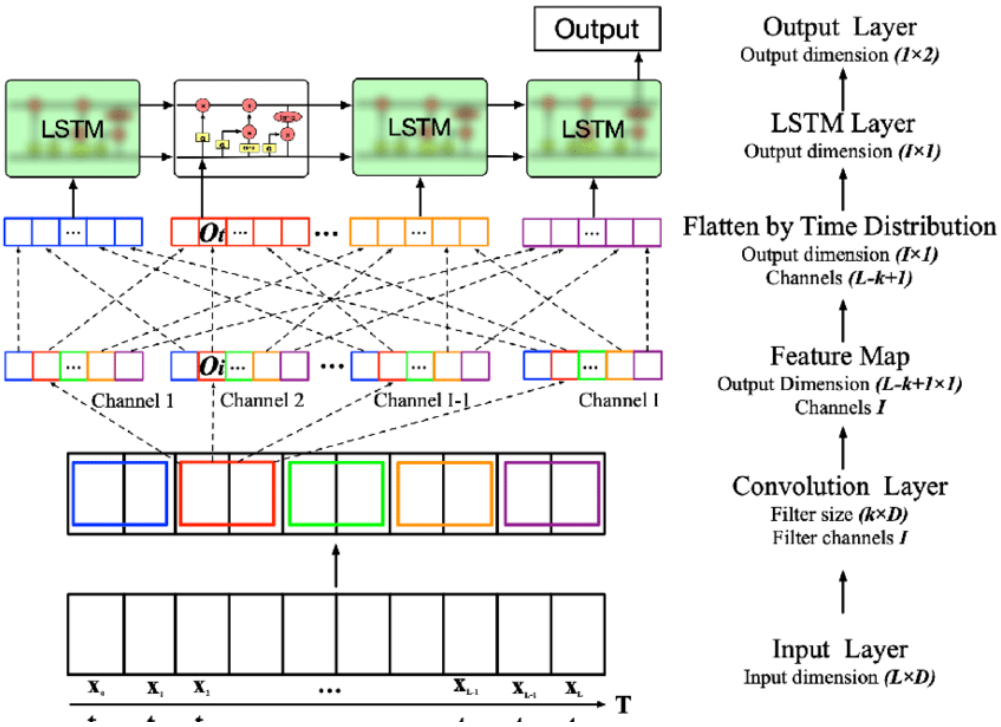

Figure 1: Model Architecture

2.4 Model Performance

We attempted to predict the classes of the data that are not used during the training phase using the trained model. We will compare these predictions with the correct labels and calculate how accurately our model classifies the videos. To measure the performance of the model, we utilized

performance evaluation metrics such as accuracy, precision, recall, and F1 score. The results we get from these metrics are in the result section.

3 Experimental Settings

3.1 Dataset

For our project, we have utilized the Real Life Violence Situations Dataset dataset, available on Kaggle. This dataset contains 1000 violence and 1000 non-violence videos collected from YouTube videos. The videos in the violence class contain many real street fight situations in several environments and conditions. Also, the videos in the non-violence class were collected from many different human actions like sports, eating, walking, etc.

3.2 Environments

For our experiments, we leveraged the powerful computing resources provided by Google Colab, utilizing the NVIDIA A100 GPU. This environment allowed us to efficiently train and evaluate our deep learning models for violence classification based on videos.

Our deep learning models were implemented using the Python programming language. For our violence video classification experiment, we utilized a range of powerful tools and libraries to ensure a robust and efficient environment. We used NumPy for numerical computations and array operations, while time helped us track the experiment's duration. Seaborn and Matplotlib were employed for data visualization, enabling us to create insightful plots and graphs. OpenCV (cv2) facilitated video processing tasks such as frame extraction, and Pandas allowed us to handle and analyze structured data efficiently.

We leveraged PyTorch and its extension torchvision for building and training our deep learning models, and incorporated pre-trained models from timm for transfer learning. Data augmentation was performed using Albumentations and ToTensorV2 to enhance model robustness. The TQDM library provided progress bars to monitor our training and validation loops effectively. MoviePy was essential for video editing and preprocessing tasks. To ensure reproducibility, we used random to set seeds for our experiments. Finally, we evaluated our model's performance using Scikit-learn to generate classification reports and confusion matrices, providing detailed insights into our model's accuracy and precision.

3.3 Evaluation

To evaluate the performance of our violence/non-violence classification method, we have employed the following evaluation metrics:

Accuracy: We calculated the overall accuracy of our model in correctly classifying violence situations. It measures the proportion of correctly classified videos over the total number of videos in the test set.

Precision and Recall: We computed the precision and recall values for two classes. Precision measures the percentage of correctly classified instances for a specific class, while recall indicates the proportion of correctly classified instances out of all instances belonging to a particular class.

F1 Score: The F1 score combines precision and recall into a single metric, providing a balanced measure of performance. It is calculated as the harmonic mean of precision and recall.

Confusion Matrix: We generated a confusion matrix to gain insights into the performance of our model across two classes. The confusion matrix displays the number of instances classified correctly and incorrectly for each class, enabling us to analyze any patterns of misclassification.

By utilizing these evaluation metrics, we aim to assess the accuracy, precision, recall, and overall performance of our violence/non-violence classification model. These metrics helped us to understand the strengths and weaknesses of our approach and guide us in further refining and optimizing our models and algorithms.

4 Experimental Results

The following experimental results show how the results are obtained when Lr is 1e-4 and when Lr is 1e-5. In our machine learning model, we're processing the test dataset at a rate of 6.37 iterations per second. This indicates how quickly we can evaluate the model's performance on new, unseen data.

Learning Rate is 1e-4

```
epoch: 1/20: 100%|██████████| 100/100 [06:50<00:00,  4.10s/it]
epoch: 1, train_accuracy:  0.74, loss:  0.570, val_accuracy:  0.86, val_loss: 0.347
Saved successfully best weights to: /content/drive/MyDrive/416Project/Kaggle/best_weights.pt
epoch: 2/20: 100%|██████████| 100/100 [06:46<00:00,  4.06s/it]
epoch: 2, train_accuracy:  0.90, loss:  0.257, val_accuracy:  0.86, val_loss: 0.311
Saved successfully best weights to: /content/drive/MyDrive/416Project/Kaggle/best_weights.pt
epoch: 3/20: 100%|██████████| 100/100 [06:41<00:00,  4.01s/it]
epoch: 3, train_accuracy:  0.94, loss:  0.156, val_accuracy:  0.94, val_loss: 0.207
Saved successfully best weights to: /content/drive/MyDrive/416Project/Kaggle/best_weights.pt
epoch: 4/20: 100%|██████████| 100/100 [06:48<00:00,  4.08s/it]
epoch: 4, train_accuracy:  0.96, loss:  0.105, val_accuracy:  0.92, val_loss: 0.204
Saved successfully best weights to: /content/drive/MyDrive/416Project/Kaggle/best_weights.pt
epoch: 5/20: 100%|██████████| 100/100 [06:45<00:00,  4.06s/it]
epoch: 5, train_accuracy:  0.98, loss:  0.076, val_accuracy:  0.95, val_loss: 0.158
Saved successfully best weights to: /content/drive/MyDrive/416Project/Kaggle/best_weights.pt
epoch: 6/20: 100%|██████████| 100/100 [06:43<00:00,  4.04s/it]
epoch: 6, train_accuracy:  0.98, loss:  0.062, val_accuracy:  0.95, val_loss: 0.169
epoch: 7/20: 100%|██████████| 100/100 [06:44<00:00,  4.04s/it]
epoch: 7, train_accuracy:  0.98, loss:  0.068, val_accuracy:  0.93, val_loss: 0.239
epoch: 8/20: 100%|██████████| 100/100 [06:45<00:00,  4.06s/it]
epoch: 8, train_accuracy:  0.99, loss:  0.038, val_accuracy:  0.92, val_loss: 0.280
Early Stopped!
Saved successfully last weights to: /content/drive/MyDrive/416Project/Kaggle/last_weights.pt
```

Figure 2: Model Train Result When Lr is 1e-4

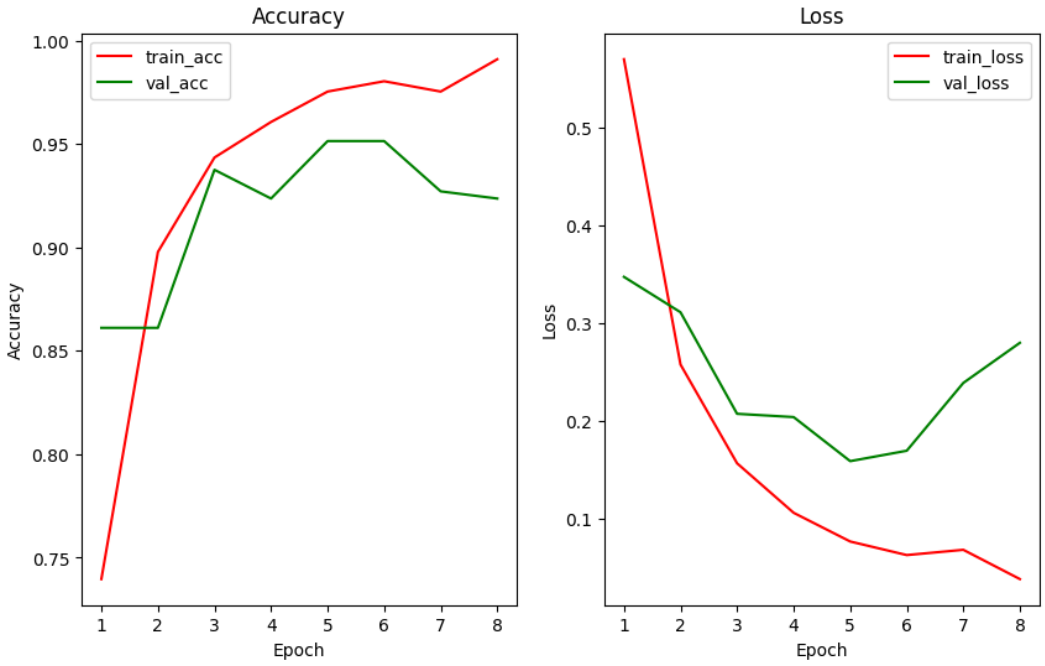

Figure 3: Learning Curves When Lr is 1e-4

Learning Rate is 1e-5

```
epoch: 1/20: 100%|| 100/100 [08:28<00:00,  5.08s/it]
epoch: 1, train_accuracy:  0.64, loss:  0.680, val_accuracy:  0.73, val_loss: 0.663
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 2/20: 100%|| 100/100 [08:23<00:00,  5.03s/it]
epoch: 2, train_accuracy:  0.77, loss:  0.645, val_accuracy:  0.81, val_loss: 0.619
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 3/20: 100%|| 100/100 [08:17<00:00,  4.97s/it]
epoch: 3, train_accuracy:  0.80, loss:  0.591, val_accuracy:  0.81, val_loss: 0.552
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 4/20: 100%|| 100/100 [08:37<00:00,  5.18s/it]
epoch: 4, train_accuracy:  0.83, loss:  0.509, val_accuracy:  0.85, val_loss: 0.464
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 5/20: 100%|| 100/100 [08:32<00:00,  5.12s/it]
epoch: 5, train_accuracy:  0.86, loss:  0.423, val_accuracy:  0.87, val_loss: 0.385
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 6/20: 100%|| 100/100 [08:23<00:00,  5.04s/it]
epoch: 6, train_accuracy:  0.88, loss:  0.350, val_accuracy:  0.89, val_loss: 0.335
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 7/20: 100%|| 100/100 [08:23<00:00,  5.03s/it]
epoch: 7, train_accuracy:  0.90, loss:  0.292, val_accuracy:  0.91, val_loss: 0.285
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 8/20: 100%|| 100/100 [08:29<00:00,  5.10s/it]
epoch: 8, train_accuracy:  0.92, loss:  0.240, val_accuracy:  0.92, val_loss: 0.255
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 9/20: 100%|| 100/100 [08:21<00:00,  5.01s/it]
epoch: 9, train_accuracy:  0.93, loss:  0.212, val_accuracy:  0.91, val_loss: 0.242
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 10/20: 100%|| 100/100 [08:33<00:00,  5.14s/it]
epoch: 10, train_accuracy:  0.94, loss:  0.189, val_accuracy:  0.92, val_loss: 0.246
epoch: 11/20: 100%|| 100/100 [08:31<00:00,  5.11s/it]
epoch: 11, train_accuracy:  0.95, loss:  0.168, val_accuracy:  0.92, val_loss: 0.204
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 12/20: 100%|| 100/100 [08:20<00:00,  5.00s/it]
epoch: 12, train_accuracy:  0.95, loss:  0.149, val_accuracy:  0.93, val_loss: 0.203
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 13/20: 100%|| 100/100 [08:31<00:00,  5.11s/it]
epoch: 13, train_accuracy:  0.95, loss:  0.144, val_accuracy:  0.93, val_loss: 0.192
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 14/20: 100%|| 100/100 [08:31<00:00,  5.12s/it]
epoch: 14, train_accuracy:  0.97, loss:  0.119, val_accuracy:  0.94, val_loss: 0.173
Saved successfully best weights to: /content/drive/MyDrive/416Project/binaryclass_classification/best_weights.pt
epoch: 15/20: 100%|| 100/100 [08:20<00:00,  5.00s/it]
epoch: 15, train_accuracy:  0.97, loss:  0.120, val_accuracy:  0.94, val_loss: 0.173
epoch: 16/20: 100%|| 100/100 [08:32<00:00,  5.12s/it]
epoch: 16, train_accuracy:  0.97, loss:  0.103, val_accuracy:  0.93, val_loss: 0.193
epoch: 17/20: 100%|| 100/100 [08:25<00:00,  5.05s/it]
epoch: 17, train_accuracy:  0.97, loss:  0.097, val_accuracy:  0.93, val_loss: 0.189
Early Stopped!
Saved successfully last weights to: /content/drive/MyDrive/416Project/binaryclass_classification/last_weights.pt
```

Figure 4: Model Train Result When Lr is 1e-5

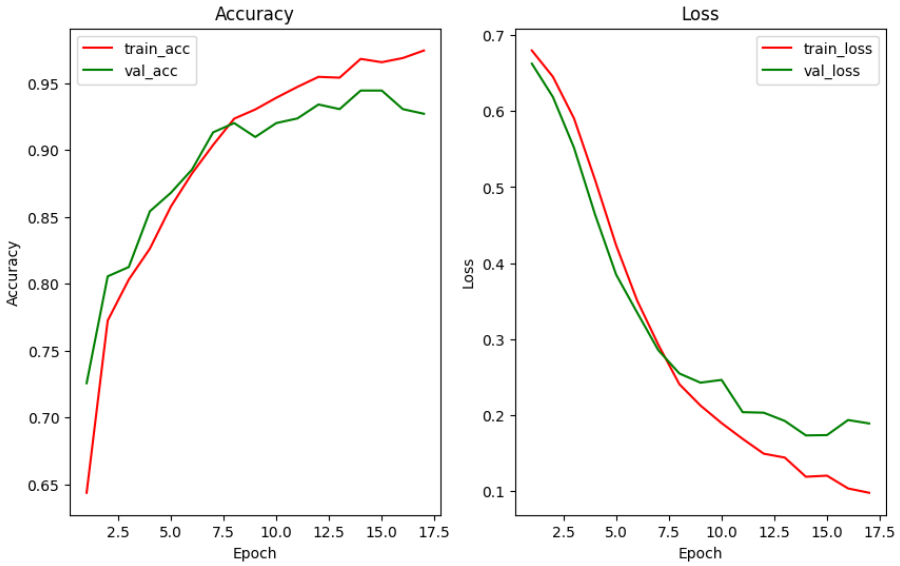

Figure 5: Learning Curves When Lr is 1e-5

	precision	recall	f1-score	support
Non-Violence	0.92	0.91	0.91	190
Violence	0.92	0.92	0.92	199
accuracy			0.92	389
macro avg	0.92	0.92	0.92	389
weighted avg	0.92	0.92	0.92	389

Figure 6: Evaluation Metrics When Lr is 1e-5

Figure 7: Confusion Matrix When Lr is 1e-5

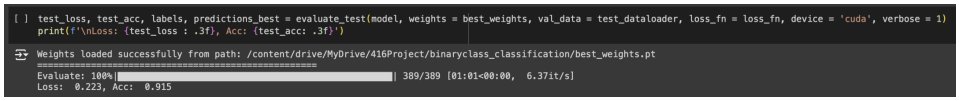

Figure 8: Test Results When Lr is 1e-5

Hyperparameter Discussion Learning Rate:

In this study, the performance of a fine-tuned transfer learning model was evaluated using different learning rates. Our observations are as follows:

1e-4 Learning Rate: The model trained with this learning rate exhibited acceptable performance on some tasks. However, the model's overall accuracy and other important metrics remained lower compared to the 1e-5 learning rate. This might be attributed to the 1e-4 learning rate being too fast for updating the model's parameters, hindering it from reaching an optimal solution.

1e-5 Learning Rate: The model trained with this learning rate significantly outperformed the 1e-4 learning rate model across all tasks. The model's overall accuracy and other important metrics considerably improved with the 1e-5 learning rate. This suggests that the 1e-5 learning rate allowed for slower and more controlled updates of the model's parameters, facilitating its convergence to an optimal solution.

As a result, careful tuning of the learning rate in a model fine-tuned on transfer learning is critical for model performance. In this study, we observed that a learning rate of 1e-5 provides better performance compared to a learning rate of 1e-4. However, since the optimal learning rate may vary depending on the dataset used, model architecture and other hyperparameters, it is important to evaluate and optimize it for each model individually.

Loss Function:

Binary Cross-Entropy Loss is used for tasks with two classes, usually labeled 0 and 1. It evaluates how well a model predicts probabilities between 0 and 1 for these classes. This loss treats the model's output as a probability, making it useful when you want to know the likelihood of belonging to the positive class.

This loss function applies a logarithmic penalty to wrong predictions, especially if the model is very confident but wrong. This encourages the model to be more accurate and cautious in its predictions. Additionally, Binary Cross-Entropy Loss provides a smooth gradient, which helps optimization algorithms like Adam to efficiently update the model's weights during training.

Optimizer:

The Adam optimizer is widely used in binary classification tasks due to its numerous benefits. It dynamically adjusts learning rates, which speeds up the convergence process and ensures efficient learning. Adam's ability to adapt the learning rate for each parameter individually enhances the precision of the model. It is also computationally efficient, making it suitable for large datasets, and it requires less memory, which is crucial for training large models. Additionally, Adam combines the best features of momentum and RMSProp, merging momentum's acceleration with RMSProp's adaptive learning to provide a stable and effective optimization process.

When combined with Binary Cross-Entropy (BCE) Loss, Adam offers significant advantages. The adaptive learning rates of Adam complement the logarithmic error calculations of BCE Loss, resulting in quicker and more accurate convergence. Adam is effective in managing both small and large gradients, which aligns well with the gradients provided by BCE Loss, enhancing their utility. This combination also improves overall model performance, especially in deep neural networks, due to the dynamic adjustments in Adam and the error signals from BCE Loss. The adaptive nature of Adam ensures stable and reliable training, which boosts the model's ability to generalize when used with BCE Loss. Moreover, Adam simplifies hyperparameter tuning and often performs well with default settings, reducing the need for manual intervention. Overall, using the Adam optimizer with BCE Loss in binary classification tasks accelerates training, improves accuracy, and enhances the stability and performance of the model.

5 CONCLUSION

The project to develop a video classification model for identifying violent content appears to be a success. Here's a breakdown of the strengths, weaknesses, and areas for improvement:

5.1 Strengths

High Accuracy: Based on the provided information, the model achieved high accuracy on both training and validation datasets, with accuracy reaching 92% and F1 score reaching 91% in the classification task. This suggests the model can effectively distinguish between violent and non-violent videos.

Generalizability: The gap between training and validation accuracy is relatively small, indicating the model generalizes well to unseen data and avoids overfitting. This is further supported by the decreasing validation loss curve.

Early Stopping: Stopping training at the optimal point based on validation accuracy demonstrates good practice to prevent overfitting.

Transfer Learning: Leveraging pre-trained models from the timm library enabled us to build on existing knowledge, which improved convergence speed and final performance.

5.2 Weaknesses

Misclassification Rate: Despite the high accuracy, the model still has an 8.5% misclassification rate for both classes. This indicates that there is room for improvement in reducing false positives and false negatives.

Computational Resources: Training the model, especially with a lower learning rate, required substantial computational resources and time. This might not be feasible in all settings.

5.3 Improvement Opportunities

Hyperparameter Tuning: Further fine-tuning of hyperparameters such as learning rate, batch size, and network architecture could potentially improve performance.

Data Quality and Quantity: Increasing the dataset size and ensuring high-quality annotations can provide more training data for the model to learn from, thus improving accuracy.

Model Selection If object-detection is used during classification, the performance of the model can be increase. Since, the model can classify tress swaying in the wind as violence. If the object-detection is active, trees could not be classify as violence because they could not be violent.

5.4 Challenges and Resolutions

Learning Rate Adjustment: Initially, the model performed suboptimally with a learning rate of 1e-4. By reducing the learning rate to 1e-5, we observed significant improvements in stability and accuracy. This adjustment required iterative experimentation and monitoring.

Computational Constraints: The extensive computational requirements posed a challenge. We addressed this by using cloud-based GPU resources and optimizing our training pipeline to minimize unnecessary computations.