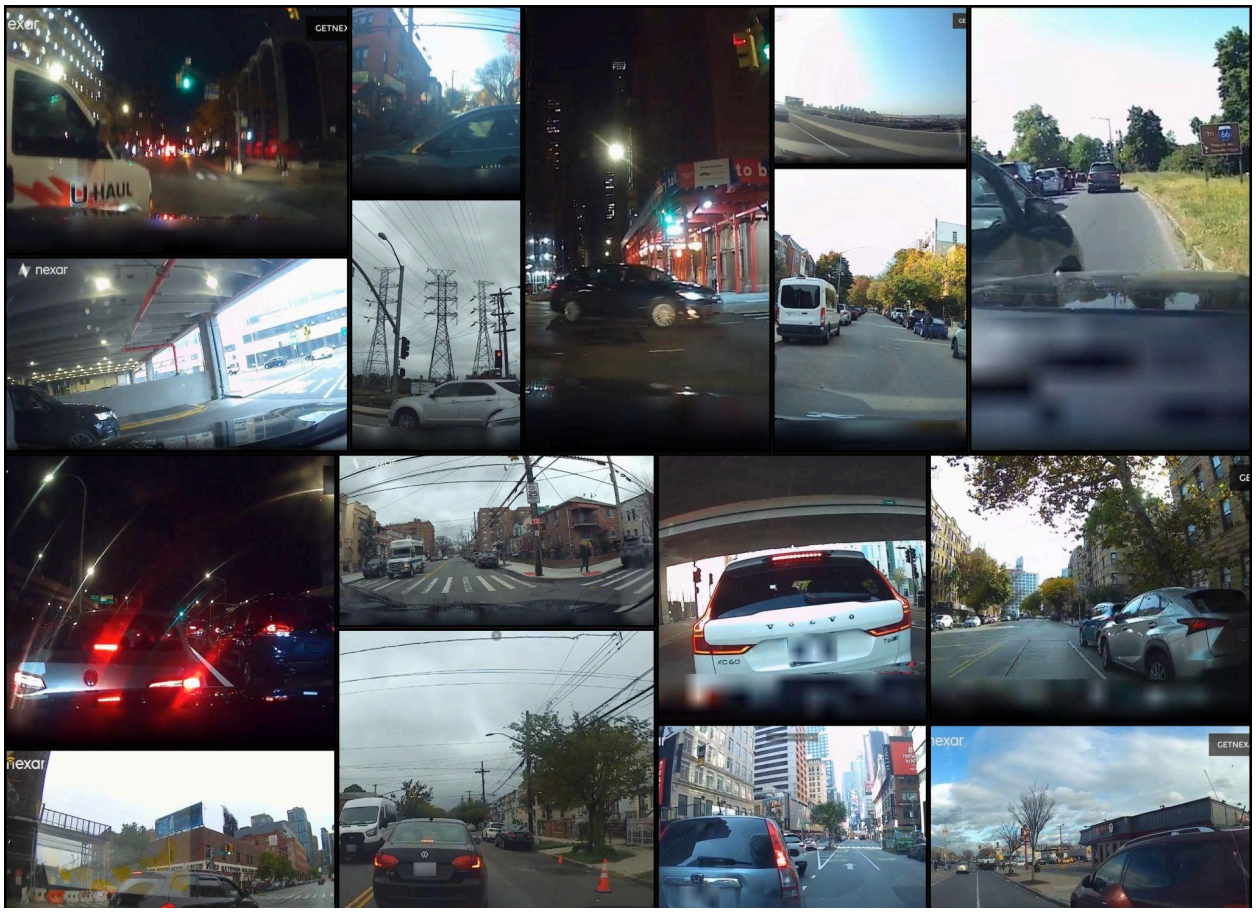
Predicting Imminent Road Collisions Using Dashcam Footage and 3D-CNN

MingYi Wei
mingyi5@illinois.edu
University of Illinois Urbana Champaign
Urbana, Illinois, USA

Sophie Huang
chihyuh3@illinois.edu
University of Illinois Urbana Champaign
Urbana, Illinois, USA

Kai-Po Chang
kpchang2@illinois.edu
University of Illinois Urbana Champaign
Urbana, Illinois, USA



In recent years, autonomous driving technology has become increasingly widespread. Both self-driving systems and advanced driver-assistance systems (ADAS) rely on accurate accident prediction to enhance road safety. The goal of this project is to develop a machine learning model capable of predicting imminent vehicle collisions using footage captured by dashcams. We hope to use this project to aid in car accident prediction. We implement it using one of the datasets available on Kaggle.

Our source code : https://github.com/OscarWei61/CS444-Final-Project.

1. Introduction

In recent years, autonomous driving technology has become increasingly widespread. Both self-driving systems and advanced driver-assistance systems (ADAS) rely on accurate accident prediction to enhance road safety. The ever-changing weather and the diverse, non-uniform nature of road conditions make this project particularly challenging. Unlike more straightforward tasks like handwriting recognition, our project must operate in complex, dynamic environments. Our model needs to identify potential collision risks from dashcam footage under a variety of road conditions. This is the challenge we aim to tackle.

The goal of this project is to develop a machine learning model that can predict imminent car accidents using video captured by dashboard cameras.

We found a relevant dataset on Kaggle — the *Nexar Dashcam Crash Prediction Challenge* (https://www.kaggle.com/competitions/nexar-collision-prediction/overview). The dataset includes three types of driving scenarios: collisions, near-collisions (close calls), and normal driving. The main challenge of this project is to detect potential accidents as early as possible, allowing for timely intervention and accident prevention.

This project aims to address the following key issues:

- Early Prediction: Accurately predict accidents before they happen, providing enough time for proactive responses.
- Complex Driving Scenarios: Handle the challenges posed by varying weather conditions, visual occlusions, and unpredictable traffic patterns.
- Distinguishing Near-Collisions from Actual Collisions: Identify high-risk driving situations that, while not resulting in actual crashes, still pose serious threats.

By developing an accurate and efficient prediction model, this project seeks to improve road safety and reduce the frequency of traffic accidents, offering drivers better and more convenient assistance tools.

2. Details of the approach

Our implementation is divided into two parts: data preprocessing and the model part.

Data Preprocess Part

The Kaggle dataset includes multiple video frames and two CSV files for training and testing, as shown in Table 1. The id represents the video identifier, indicating which dashcam video the data belongs to. The target indicates whether an accident occurred; if an accident did occur, the corresponding time is recorded. time_of_event and time_of_alert record the time of the accident and the alert time, respectively. The actual video frames from the dataset are shown in Figure 1.

id	time_of_event	time_of_alert	target
1924			0
822	19.500	18.633	1
1429			0
208	19.800	19.233	1

Table1. Train data example

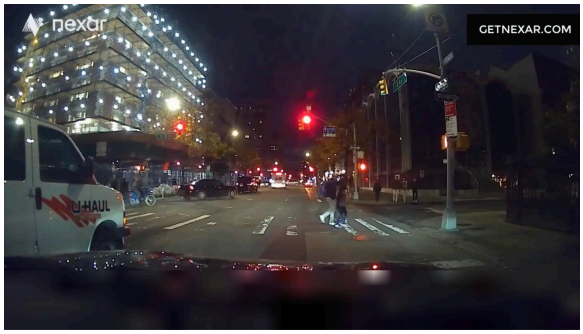


Figure1. dashcam image example

Since our data is stored in video format, we need to extract each video into individual frames before training. These frames are then converted into tensors to facilitate subsequent model training, as shown in figure2.

```python
# Define function to extract randomly sampled frames from a video
def extract_random_frames(video_path, num_frames=40, resize=(224, 224)):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error opening video file: {video_path}")
        return None

    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    if total_frames <= 0:
        cap.release()
        return None

    # Randomly sample 'num_frames' unique indices and sort them to preserve temporal order
    frame_indices = sorted(np.random.choice(total_frames, num_frames, replace=False))

    frames = []
    current_frame = 0
    next_idx = 0
    ret = True
    while ret and next_idx < len(frame_indices):
        ret, frame = cap.read()
        if not ret:
            break
        if current_frame == frame_indices[next_idx]:
            frame = cv2.resize(frame, resize)
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame = frame.astype(np.float32) / 255.0  # Normalize to [0,1]
            frames.append(frame)
            next_idx += 1
        current_frame += 1
    cap.release()

    if len(frames) < num_frames:
        while len(frames) < num_frames:
            frames.append(frames[-1])

    # Convert frames to tensor with shape (B, C, T, H, W)
    frames = np.stack(frames, axis=0)          # (T, H, W, C)
    frames = np.transpose(frames, (3, 0, 1, 2))  # (C, T, H, W)
    frames_tensor = torch.from_numpy(frames).unsqueeze(0)  # Add batch dimension
    return frames_tensor
```

Figure2. extract frame from video

Model Part

We design a CrashDetectNeuralNetwork model which is a custom-designed 3D Convolutional Neural Network (3D CNN) developed for binary video classification tasks, detecting whether a video contains a crash or not. The model processes spatiotemporal video input using 3D convolutional layers, allowing it to capture both spatial features (from individual frames) and temporal dynamics (across frames).

The model input is a tensor of shape (Batch, Channels=3, Time, Height, Width), representing RGB video clips sampled over time. The architecture consists of three main convolutional blocks followed by a fully connected classifier.

Block 1 performs initial feature extraction. It contains two 3D convolutional layers with 16 output channels, each followed by Batch Normalization and ReLU activation. A spatial-only MaxPool3d layer is used to reduce the height and width dimensions by half while preserving the temporal dimension.

Block 2 deepens the feature representation by increasing the number of channels to 32. Like Block 1, it uses two 3D convolutional layers with BatchNorm and ReLU, followed by another spatial pooling layer. After this stage, the spatial dimensions are reduced to one-quarter of the original size.

Block 3 increases the channel depth to 64 and uses two more convolutional layers. At the end of this block, the model applies AdaptiveAvgPool3d(1), which performs global average pooling across the entire spatiotemporal space (T, H, W), effectively compressing the data into a compact representation of shape (B, 64, 1, 1, 1).

The output is then flattened and passed through a Dropout layer (p=0.5) to prevent overfitting. A final fully connected layer projects the 64-dimensional vector down to a single value, and a sigmoid activation function maps it to a probability between 0 and 1.

In summary, our CrashDetectNeuralNetwork is a hierarchical 3D CNN designed to extract rich spatial and temporal features from video clips. It compresses this information using global pooling and outputs a single probability score for binary classification. Its design is efficient for real-time inference and suitable for tasks

involving short video segments or safety event detection.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv3d-1     [-1, 16, 16, 224, 224]           1,312
       BatchNorm3d-2     [-1, 16, 16, 224, 224]              32
              ReLU-3     [-1, 16, 16, 224, 224]               0
            Conv3d-4     [-1, 16, 16, 224, 224]           6,928
       BatchNorm3d-5     [-1, 16, 16, 224, 224]              32
              ReLU-6     [-1, 16, 16, 224, 224]               0
         MaxPool3d-7     [-1, 16, 16, 112, 112]               0
            Conv3d-8     [-1, 32, 16, 112, 112]          13,856
       BatchNorm3d-9     [-1, 32, 16, 112, 112]              64
             ReLU-10     [-1, 32, 16, 112, 112]               0
           Conv3d-11     [-1, 32, 16, 112, 112]          27,680
      BatchNorm3d-12     [-1, 32, 16, 112, 112]              64
             ReLU-13     [-1, 32, 16, 112, 112]               0
        MaxPool3d-14       [-1, 32, 16, 56, 56]               0
           Conv3d-15       [-1, 64, 16, 56, 56]          55,360
      BatchNorm3d-16       [-1, 64, 16, 56, 56]             128
             ReLU-17       [-1, 64, 16, 56, 56]               0
           Conv3d-18       [-1, 64, 16, 56, 56]         110,656
      BatchNorm3d-19       [-1, 64, 16, 56, 56]             128
             ReLU-20       [-1, 64, 16, 56, 56]               0
AdaptiveAvgPool3d-21          [-1, 64, 1, 1, 1]               0
          Dropout-22                [-1, 64]                  0
...
Forward/backward pass size (MB): 1065.75
Params size (MB): 0.83
Estimated Total Size (MB): 1075.76
```

Figure3. Model Structure

3. Results

3.1 Effect of Accident/Non-Accident Class Balance on Model Performance

To understand how the proportion of positive (accident) and negative (non-accident) clips influences generalisation, we trained six different variants. In every case we maintained the same preprocessing pipeline, learning rate = 0.0001, epoch = 30, batch size = 8, and optimiser Adam. The dataset still uses the *Nexar Dashcam Crash Prediction Challenge* on Kaggle. Also, performance was evaluated with Kaggle's leaderboard.

According to *table 2*, a balanced 200-clip subset (ID 1) provides a competitive baseline (Score = 0.605). Expanding that subset to 300 clips while preserving the 1 : 1 ratio (ID 2) raises score by 0.021, indicating that additional visual diversity outweighs the risk of over-fitting at this scale. Shifting to a slight accident surplus (ID 3, 3 : 2) hurts performance, whereas a mirror surplus of non-accident footage (ID 4, 2 : 3) achieves the best score.

ID	Sampling strategy (accident : non-accident)	# Clips	Epochs	Score on Kaggle
1	100 + 100 (balanced)	200	30	0.605
2	150 + 150 (balanced)	300	30	0.626
3	120 + 80 (3 : 2)	200	30	0.620
4	80 + 120 (2 : 3)	200	30	0.640

Table 2. Performance of different class balance

Figure 4 shows the curve of both training loss and validation loss. In this figure, both curves decrease obviously. *Figure 5* indicates the exposure to a moderately larger pool of negative scenes, sharpens the network's discrimination boundary without diluting its sensitivity to rare collision cues.

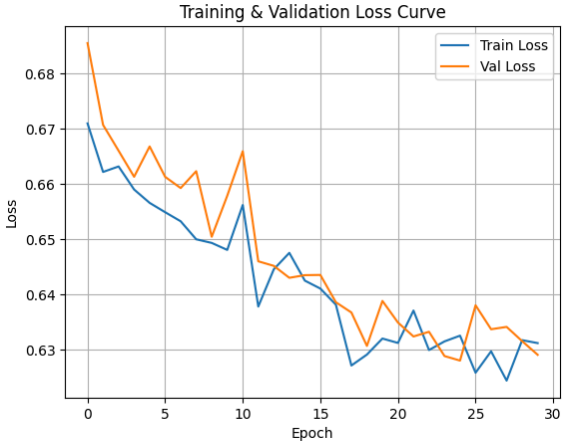


Figure 4. Training and Validation Loss of ID 3

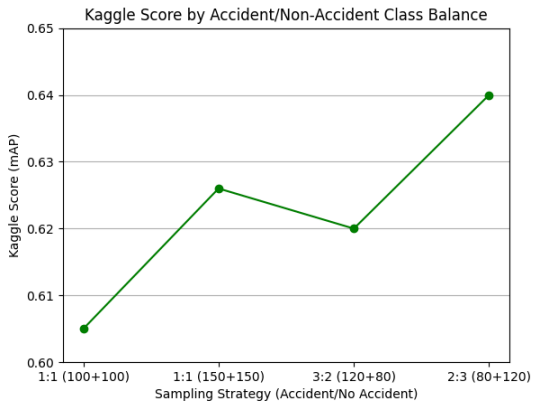


Figure 5. Kaggle Score by Accident/Non-Accident Class Balance

3.2 Effect of Learning Rate/Weight Decay On Model Performance

To analyze the influence of learning rate and weight decay on model effectiveness, we trained four configurations while keeping all other variables constant (batch size = 8, epoch = 30, optimizer = Adam). All experiments were conducted on the same dataset and evaluated using the Nexar Dashcam Crash Prediction Challenge leaderboard.

ID	Learning Rate	Weight Decay	Epochs	Score on Kaggle
5	0.001	0.00001	30	0.570
6	0.0005	0.00001	30	0.588
7	0.0001	0.000001	30	0.612
8	0.00005	0.000001	30	0.622

Table 3. Performance of Different Learning Rates

As shown in *Table 3*, decreasing the learning rate from 0.001 to 0.00005 results in a steady performance improvement, with the best Kaggle score of 0.622 achieved at the smallest learning rate tested (ID 8). This trend suggests that lower learning rates help the model converge more effectively and avoid overshooting minima, especially when combined with minimal regularization through weight decay.

Figure 6 displays the loss curve for ID 5 (learning rate = 0.001), showing unstable training with oscillations in both training and validation loss. This instability likely contributes to its lower performance. In contrast, *Figure 7* illustrates a clear upward trend in model accuracy as learning rate decreases, confirming that more conservative updates help stabilize learning and improve the model's generalization ability.

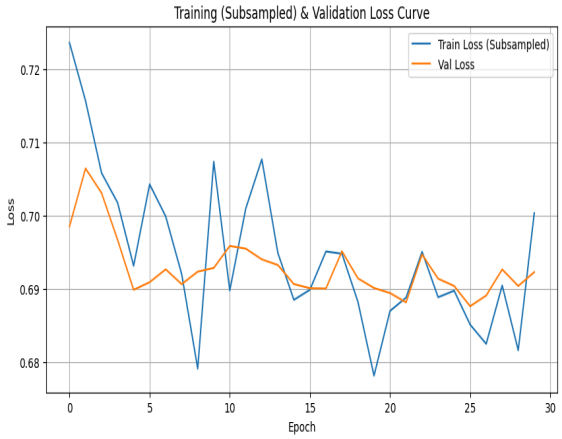


Figure 6. Training and Validation Loss of ID 5

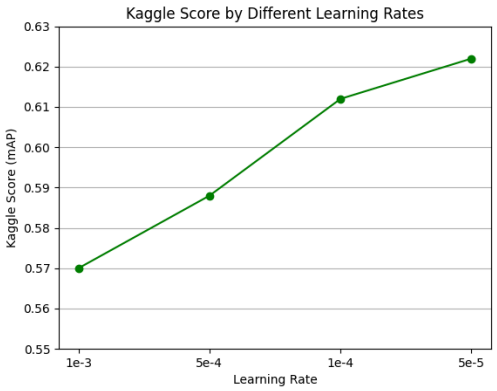


Figure 7. Kaggle Score by Different Learning Rates

3.3 Effect of Optimizer On Model Performance

To investigate how different optimizers influence generalization, we trained three model variants, each using a distinct optimizer with the same loss function, BCEWithLogitsLoss. For all experiments, we maintained a consistent training pipeline: preprocessing procedures were kept identical, with a fixed learning rate of 0.0001, batch size of 8, and 30 training epochs. Model performance was evaluated using Kaggle's official leaderboard to ensure comparability and relevance to real-world competition metrics.

```python
criterion = nn.BCEWithLogitsLoss()

# Method : Adam
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

# Method : SGD
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4, momentum=0.9)

# Method : RMSprop
optimizer = torch.optim.RMSprop(model.parameters(), lr=1e-4)
```

Figure 8.

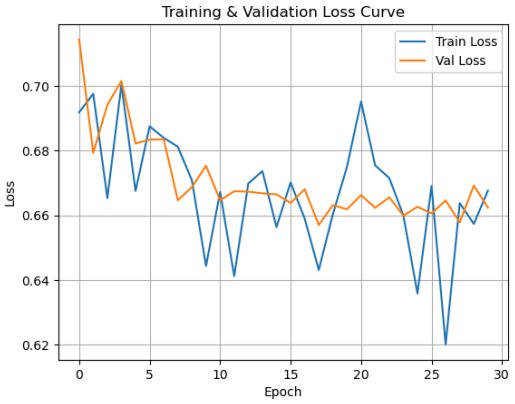


Figure 9. Training and Validation Loss of ID 9

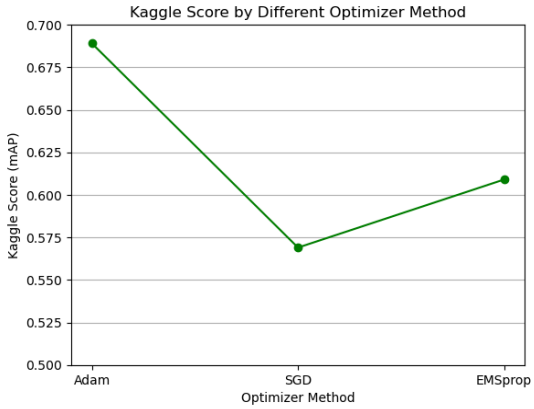


Figure 10 Kaggle Score by Different Optimizer Method

Based on the trend of the training and validation loss curves, we find out that the model using the Adam optimizer demonstrates a relatively stable training process. Although the training loss shows some fluctuations, the validation loss remains comparatively steady and gradually aligns with the training loss, indicating good generalization performance.

ID	Optimizer	Epochs	Score on Kaggle
9	Adam	30	0.686
10	SGD	30	0.569
11	RMSprop	30	0.609

Table 4. Performance of different optimizers

4. Discussion and conclusions

4.1 Discussion

The investigation into the effect of accident/non-accident class balance (Section 3.1) provided several crucial insights into the model's learning behavior and generalization capabilities on the Nexar dashcam dataset. The initial balanced subset of 200 clips (ID 1) established a reasonable baseline Kaggle score of 0.605. Expanding this dataset to 300 clips, while maintaining the 1:1 balance (ID 2), resulted in an improved score of 0.626. This improvement suggests that, at this scale, providing the model with more diverse visual examples is beneficial and helps in learning more generalizable features, outweighing the potential risk of overfitting, which can often be a concern with larger datasets if not managed properly. This aligns with the general understanding that more data often leads to better performance, provided the data quality is maintained and it introduces useful variance. However, creating a slight surplus of accident

clips (ID 3, 3:2 ratio) led to a decrease in performance (score 0.620) compared to the larger balanced set, and even slightly worse than the smaller balanced set, despite *Figure 4* indicating that both training and validation losses were decreasing, which typically suggests the model is learning. Conversely, the configuration with a surplus of non-accident footage (ID 4, 2:3 ratio) achieved the highest score of 0.640 among the tested variations.

The analysis of learning rate and weight decay effects reveals that smaller learning rates consistently lead to better model performance. As shown in the experiments (*Table 3*), the Kaggle score improves from 0.570 at a learning rate of 0.001 to 0.622 at 0.00005, indicating a strong inverse correlation between learning rate and model accuracy. High learning rates (e.g., 0.001) result in unstable training, as evidenced by the fluctuating loss curves in *Figure 6*. These fluctuations suggest that the model struggles to converge and may overshoot optimal weight updates. In contrast, lower learning rates stabilize training, allowing the model to make finer, more controlled updates that lead to better generalization on the validation set.Weight decay was held constant or minimally adjusted across the experiments, and its relatively small values had a limited but supportive regularizing effect. This suggests that, for this task, the learning rate is the dominant factor influencing performance, while weight decay plays a secondary role in preventing overfitting.

As shown in *Table 4*, the model trained with the **Adam** optimizer achieved the highest Kaggle score (0.686). This result aligns with our observations from the training and validation loss curves, where Adam showed the most stable and consistent performance throughout training. SGD, despite being a classical optimizer, appears to struggle in this context due to its sensitivity to learning rate and lack of adaptive behavior, which likely led to underfitting and a lower score. RMSprop performed slightly better but still exhibited considerable fluctuations in loss, suggesting that it might not be as robust in handling the complexity and variability of dashcam video data. In contrast, Adam's adaptive learning rate and momentum components allowed the model to converge more smoothly and maintain better generalization, especially important for real-world accident prediction where data is highly diverse and noisy. Therefore, from both a quantitative and qualitative perspective, **Adam emerges as the most suitable optimizer** for this task based on the experiment results..

4.2 Conclusion

In this work, we implemented a simple 3-D CNN (DeepCrashNN) and conducted an extensive grid of experiments on the Nexar Dashcam dataset to isolate the effects of class balance, learning rate, and optimizer choice. Our best model, which uses Adam optimizer, reached a Kaggle score of 0.686, demonstrating that careful tuning of the training regimen can generate substantial gains even without major changes. However, there is still room for improvement in our method's Kaggle performance.

4.3 Future Work

The current exploration with the DeepCrashNN model, while extensive, yielded suboptimal results, primarily attributed to the extreme class imbalance (1s and 0s) in the dataset. This imbalance likely hindered the 3D CNN's ability to effectively learn discriminative features for crash prediction directly from video sequences. Future efforts will focus on addressing this data challenge and exploring architectural enhancements.

4.3.1 Two-Stage Feature Extraction and Temporal Modeling

First, we could implement a pre-processing stage using an object detection model like YOLOv9 with StrongSORT/DeepSORT to identify and track road users and traffic elements, generating bounding boxes, class labels, and tracking IDs. Second, extract feature maps from an intermediate layer of YOLOv9 (or a similar 2D CNN) for detected objects or entire frames, providing a richer representation than raw pixels. This sequence of 2D feature maps can then be fed into the DeepCrashNN or a modified version, allowing the 3D CNN to focus on temporal relationships between higher-level features.

4.3.2 Model Architecture Enhancements

We could explore state-of-the-art video Transformer models like VideoMAE or TimeSformer, which are good at understanding temporal dynamics and can be pre-trained on large video datasets then fine-tuned. Extracted feature maps from YOLOv9 could also serve as input tokens. If using the YOLOv9 feature extraction, per-frame feature map sequences can be processed by LSTMs or GRUs before a final classification layer to explicitly model the temporal sequence. Finally, integrate attention mechanisms (self-attention or cross-attention between object tracks) into the 3D CNN or RNN/Transformer models to allow the network to focus on salient spatio-temporal regions or object interactions.

5. Statement of individual contribution

MingYi Wei: Focuses on data management, experimental setup. He is responsible for preprocessing the Kaggle Nexar dataset, managing the data pipelines, and carrying out hyperparameter tuning and performance analysis.

Kai-Po Chang: Leads the development and optimization of the custom 3D CNN architecture. He handles model implementation, integration of deep learning frameworks, and works on improving computational efficiency and evaluation of prediction performance.

Sophie Huang: Oversees project coordination and system integration. She ensures collaboration using version control tools, manages documentation, and supports both model development and overall project strategy through regular team communications.

6. Reference

[1] Hébert, A., Guédon, T., Glatard, T., & Jaumard, B. (2019). High-resolution road vehicle collision prediction for the City of Montreal. *In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data)* (pp. 1804–1813). IEEE. https://doi.org/10.1109/BigData47090.2019.9006009

[2] Deublein, M., Schubert, M., Adey, B. T., Köhler, J., & Faber, M. H. (2013). Prediction of road accidents: A Bayesian hierarchical approach. *Accident Analysis & Prevention, 51*, 274–291. https://doi.org/10.1016/j.aap.2012.11.019

[3] Owjimehr, O. (2022). Road collision analysis and prediction using machine learning approaches (Master's thesis, Department of Geomatics Engineering, University of Calgary). University of Calgary. https://prism.ucalgary.ca

[4] Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T., & Meng, H. (2023). StrongSORT: Make DeepSORT great again. *arXiv.* https://doi.org/10.48550/arXiv.2202.13514

[5] Wang, C.-Y., Yeh, I.-H., & Liao, H.-Y. M. (2024). YOLOv9: Learning what you want to learn using programmable gradient information. *arXiv*. https://doi.org/10.48550/arXiv.2402.13616