Deep Crash Detection from Vehicular Sensor Data with Multimodal Self-Supervision

Luca Kubin†, Tommaso Bianconcini†, Douglas Coimbra de Andrade†, Matteo Simoncini†, Leonardo Taccari†, Francesco Sambo†‡

***Abstract*—The ability to detect vehicle accidents from on-board sensor data is of the utmost importance to provide prompt assistance to prevent injuries and fatalities. In this article, we present a novel deep learning method capable of analyzing time series recorded from Inertial Measurement Units (IMU) and GPS devices to recognize the presence of an accident along with its severity. We propose a neural architecture capable of exploiting the different sensor streams (i.e., acceleration, gyroscope, and GPS speed), a multimodal constrastive self-supervised training procedure, and an ad-hoc stack of data augmentation techniques, specifically designed to counteract the extreme class imbalance and to improve the generalization capabilities of the whole pipeline. The proposed method has been validated against several state-of-the-art methods on a large and highly imbalanced dataset, composed of more than 200 thousand time series collected from US vehicles, with different vehicle sizes and traveling on different types of road. Our method achieves an average-precision score (AP) of 0.9 in the detection of crashes and 0.76 in the detection of severe crashes, significantly outperforming all the other approaches, and has small footprint and latency, so that it can easily be deployed on embedded devices.**

***Index Terms*—Crash Detection, Neural Networks, Self-Supervised Learning, Time Series Classification.**

I. Introduction

ROAD traffic accidents are a leading cause of death, injury and disability. In 2019 about 40000 fatalities were registered in the US due to traffic accidents. World Health Organization predicts that road traffic injuries will be the fifth cause of death by 2030. The development of a real-time collision detection algorithm able to promptly detect and report traffic accidents is of the utmost importance for intelligent vehicles. In recent years, manufacturers, insurance companies, and fleet intelligence companies have been developing automatic crash detection systems embedded into connected vehicles (recording IMU and GPS data), whose aim is to monitor the vehicle dynamics and detect anomalies from telematics recordings [1]. The reliability of crash detection algorithms and their ability to estimate the severity of detected events assume a critical role in promptly and automatically calling police offices or emergency when severe accidents are detected. In this direction, we present a novel light-weight deep learning approach that analyzes vehicle sensor data (acceleration, angular speed, and GPS speed) recorded inside the vehicle to understand the presence of an accident and its severity. Our method works on non-overlapping chunks of 16 seconds of duration, and it is able to compute a prediction on a standard CPU in around 0.03 seconds. Given the fact that crashes are extremely rare events (the average US driver has one accident every 165000 miles on average), and machine learning approaches find a great challenge when dealing with imbalanced datasets [2], we propose a custom training procedure based on self-supervised learning that greatly improves the end-to-end performance of our method. We also propose a neural architecture, capable of tackling data recorded from heterogeneous sensor streams (multimodal data), with a custom architecture specifically designed to leverage the knowledge that crash severity levels are intrinsically ordered (ordinal classification problem). We compare our architecture with state-of-the-art architectures for time series classification, showing that our design choices outperform the state of the art for the problem we consider. Moreover, we develop a custom pipeline of data augmentations, specifically designed to deal with IMU/GPS data, aiming to reduce overfitting for the minority accident-related classes in our dataset. The main contributions of our work are:

- An effective deep learning architecture capable of extracting vehicle dynamics-related features that are employed to solve the traffic accident detection problem formulated as an ordinal multiclass classification task.
- A stack of data augmentation techniques, specific for this task, able to significantly reduce overfitting for the minority class.
- A customized training procedure, suitable for this kind of imbalanced classification problem, based on a contrastive multimodal self-supervised learning method, followed by fine-tuning with negative sampling.

The paper is organized as follows: in Section II we review existing literature about time series classification and crash detection from vehicular sensor data; in Section III we present an overview of our dataset, we describe the sensors' specifications and we provide definitions for our ground truth labels; in Section IV we introduce the proposed neural architecture, we formulate mathematically the task we want to solve, we showcase all the data augmentations employed to improve the generalization capabilities, and finally we describe the training procedure composed of an unsupervised pre-training step and supervised fine-tuning; in Section V we discuss experimental results followed by concluding remarks in Section VI.

II. Related Work

Time series classification (TSC) is a well-known problem in the machine learning literature that aims at learning a classification function mapping a whole temporal sequence to a single categorical variable. Before the advent of deep learning

†Verizon Connect Research, Florence, Italy
‡Email: francesco.sambo@verizonconnect.com

based techniques, popular approaches for TSC were based on Nearest Neighbor (NN) classifier coupled with a suitable distance function (e.g., Dynamic Time Warping, DTW) [3]. Other classical approaches [4], [5] leverage ensemble methods where random forests or other discriminative classifiers exploit several automatically extracted features from different representations (e.g., shapelet transformations or DTW features). Although these methods achieve great accuracy, in practice they often result unfeasible to datasets with several thousands of samples. Their main limitation is the time complexity, which typically scales as $O(n^2l^4)$ or $O(n^2l^2)$, where n is the number of examples and l is the time series length [6]. More recently, deep neural networks have proved to be effective when processing unstructured data such as images, texts and time series. In the survey [7] the authors perform an empirical comparative study of the most recent deep learning approaches for TSC, showing that deep neural networks can significantly outperform DTW-based NN classifiers and can achieve the same performance of ensemble methods with a significant drop in time complexity. Through an exhaustive empirical study, the authors strongly suggest using out-of-the-box ResNet [8] implementations with 1D convolutional blocks and batch normalization instead of 2D blocks used for images. The same authors of the survey propose an Inception-like convolutional neural network (CNN) [9] capable of achieving better performance than ResNet in several scenarios. Recurrent neural networks (RNN) have also proved extremely effective in solving time series classification-related tasks [10], [11].

One known problem of deep neural networks is weight initialization, especially working with small datasets. In image classification problems, it is customary to *pre-train* CNN-based models on extremely large, high-quality datasets, such as ImageNet: pre-trained models can be fine-tuned much more easily on supervised tasks where only few or noisy labels are available, providing better generalization capabilities. Recently developed machine learning techniques attempt to pre-train deep models on unlabeled data leveraging the self-supervised learning paradigm [12]. Self-supervised learning is a form of unsupervised learning where the data itself provides a mechanism to obtain the error signal needed for training. In particular, contrastive self-supervised learning recently led to state-of-the-art performances in the unsupervised training of deep image models [13]. The general idea behind contrastive self-supervised learning is to train an encoder to maximize the agreement (usually through cosine similarity) between different augmented views of the same data point in the latent space [14]. In essence, an encoder trained using a contrastive loss function learns to map similar data points into similar representations that can be used as features for supervised downstream tasks. Contrastive self-supervised learning techniques have been successfully applied as pre-training tasks in numerous deep learning field, from speech [15], time series [16], [17], structured language models [18] to computer vision [19], [20]. The main drawback of these methods is that they usually require big batch sizes (with more than 512 elements) to produce meaningful encoded representations and to avoid possible collapsed solutions (i.e., solutions in which the encoder outputs a constant vector for every input). This issue has been overcome in [21] where the authors show that stop-gradient operations are enough to learn meaningful representations from the data, avoiding collapsed solutions, without the use of big batches. Let us also highlight that, to effectively use contrastive learning, data augmentation techniques play a fundamental role. In [22], [23] the authors present an exhaustive set of transformations based on spectral distortion, time distortion, dynamical filtering, and time-warping specifically designed for multivariate time series.

A specific example of TSC is crash detection from vehicular sensor data, that is the topic we focus on in this article. Many crash detection systems are based on lightweight algorithms that rely on filters and thresholds specifically designed to run on low-cost hardware such as digital signal processors embedded in the sensor boards, see [24], [25]. These systems are usually tuned to have a low recall and a high precision and can detect only very severe accidents that are usually associated with airbag deployment. In [26] the authors present a similar method: they first apply a smoothing technique (based on Kalman filters or other signal processing tools) on the inertial signals to estimate the instantaneous linear acceleration, and then they use thresholds on the magnitude of the linear acceleration and the vehicle speed. In [27] the authors perform an ablation study regarding the most relevant variables characterizing the severity of vehicle accidents. They first separate accidents in 3 big families (front, side, and rear-end accidents), and, for each of them, they provide a set of thresholds for the vehicle speed and acceleration peaks needed to infer a severity level in a scale of 1 to 3. In [28] the authors present a crash detection strategy for motorcycles, using GPS and inertial measurements collected by telematics e-Boxes. After an initial self-calibration phase that aligns the sensor reference system with a virtual ground reference frame, an on-board monitoring system detects anomalous motion patterns that are further processed to extract the severity of the analyzed event. Finally, in [29] the authors develop a recurrent neural network, jointly analyzing sensors stream and dashcam videos data, to classify crash-related events. To deal with missing values and noise, the authors propose a novel neural network layer, called denoising gated recurrent unit.

To the best of our knowledge, compared to the literature on the subject, ours is the first successful attempt to apply modern constrastive self-supervised techniques to the problem of crash detection purely from inertial sensors and GPS data.

III. Dataset and Definitions

The dataset we use in this work is composed of IMU (acceleration and gyroscope) and GPS speed time series recorded from vehicles travelling in the US. Each sample in our dataset contains acceleration from a triaxial accelerometer (sampled at 100 Hz and with dynamical range between $[-4, +4]$ g), angular speed from a triaxial gyroscope (sampled at 100 Hz and with a dynamical range between $[-360, 360]$ degree/s), and speed from GPS (sampled at 1 Hz but interpolated to 100 Hz to have the same duration of IMU sensors). The time series are not further preprocessed before being fed to the network. Each sample included in our dataset has a duration of 16 seconds and has one of three possible ground truth labels:

TABLE I: Data Distributions

	Train	Validation	Test
Non-crash	162217	37082	32445
Minor crash	707	163	143
Severe crash	207	49	38

- Non-crash: this label denotes samples not associated to any traffic accident event.
- Minor crash: this label denotes samples associated to minor crash events. These kinds of events includes physical contact with other road objects but with minimal damage (e.g., fender benders, small animal strikes, curb strikes).
- Severe crash: This class denotes samples associated to police-reportable crashes, i.e., events in which visible damage are present in the vehicles involved. This class also contains crashes that include airbag deployment and possible injury of drivers.

In general, the crashes in our dataset are very diverse: they include collisions from any direction (front, lateral, rear) and cases such as rollovers or off-roads. The distribution of the events in our dataset is highly imbalanced: it is composed overall of 233051 samples, of which 1013 are minor crashes, and only 294 are severe crashes. We split our dataset into 3 stratified parts in order to obtain a train, validation, and test set. Data distributions for each split are reported in Table I.

IV. Methodology

The deep learning model we present in this article aims at predicting one of the categories presented in the previous section, through the analysis of vehicular sensor data (GPS speed and IMU). Such model is an end-to-end deep learning architecture that we decompose in two distinct components: an encoder backbone f_ψ that maps the sensor streams into a fixed dimensional vector v, and a two-headed prediction block that maps the latent representation v into posterior probabilities. Formally, given IMU and GPS speed time series $[a(t), s(t), g(t)]$ (where $a(t)$ denotes the acceleration, $s(t)$ the GPS speed, and $g(t)$ the angular speed from gyroscope), our encoder with parameters ψ, produces a latent representation v:

$$v = f_\psi(a(t), s(t), g(t)) \tag{1}$$

This latent representation v is mapped with two distinct prediction heads (parametrized with ϕ_1 and ϕ_2) to posterior probabilities:

$$\text{P(crash)} = f_{\phi_1}(v) \tag{2}$$

$$\text{P(severe crash} \mid \text{crash)} = f_{\phi_2}(v) \tag{3}$$

Equations 2 and 3, respectively, represent a soft score about the presence of a crash (Minor + Severe) and of a Severe crash given that a crash occurred. The backbone f_ψ is initially pre-trained with a contrastive self-supervised method. Afterwards, the whole architecture including parameters [ψ, ϕ_1, ϕ_2] is fine-tuned to solve the supervised downstream task. In the next subsections, we firstly introduce the architecture of the backbone f_ψ (Section IV-A), then we show the whole architecture (f_ψ, f_{ϕ_1} and f_{ϕ_2}) used to solve the downstream task (Section IV-B), explaining in details the structure of the prediction heads and the supervised training loss; after that, we show the data augmentation procedure we employ in our pipeline, for both supervised and self-supervised tasks (Section IV-C); finally we describe in depth our training procedure, namely, our self-supervised pre-training method (Section IV-D) and the supervised fine-tuning (Section IV-E).

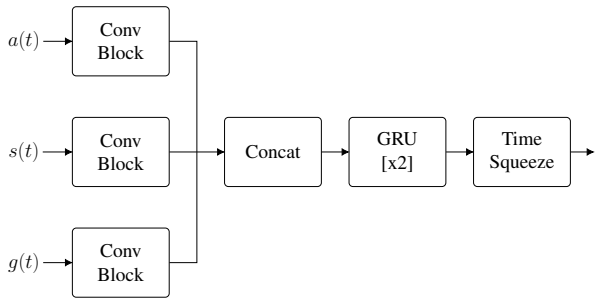

Fig. 1: Backbone Architecture: Mono-dimensional convolutional blocks process each input sensor data without mixing the information of different sensor streams. A stack of bidirectional GRU merges the features extracted from different sensors producing a time series that is finally time-squeezed to produce a fixed dimension feature vector.

A. Backbone Architecture

Inspired by the most recent advancements in multimodal machine learning for time series classification [30], the proposed architecture is composed of three distinct parts, as shown in Figure 1:

- Three mono-dimensional convolutional blocks extract independent features from each sensor stream. These blocks reduce the time duration of their input data, through pooling operations, and enlarge the number of feature channels.
- A stack of bidirectional gated recurrent layers [31] (GRU), working on the concatenation of convolutional block features, extracts joint representations merging the independent sensor streams.
- A time-squeeze operator reduces the multivariate time series, at the output of the recurrent network, to a multidimensional feature vector that is used to solve the downstream classification tasks.

Each convolutional block is a series of [Convolution 1D $\rightarrow$ Batch Norm 1D $\rightarrow$ Rectified Linear Unit (ReLu) $\rightarrow$ Max Pooling], where at each stage the time duration is halved by the max-pooling operator. In our experiments, we observed that using the max-pooling operator to downsample the signal gives better results compared to strided convolutions or average poolings. We conjecture that this happens because crash related information is contained in burst spikes of inertial signals that are passed forward, layer by layer, by the max-pooling operator.

The time squeeze operation produces the latent representation v by concatenating the last timestamp of the forward GRU output together with the first timestamps of the backward one, a standard practice in classification tasks with RNNs.

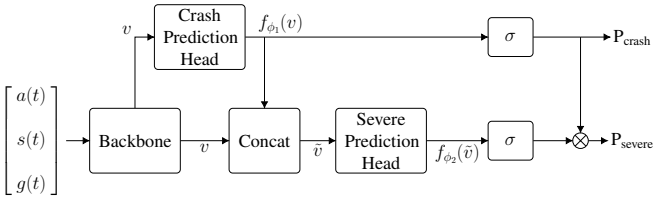

Fig. 2: Downstream Architecture: The multidimensional vector at the output of the backbone is fed to two distinct MLP heads that are used to produce posterior probabilities of crashes and severe crashes.

B. Downstream Task

The problem we aim to solve in this article is a multiclass classification problem where the target variable can assume three different ordered values (Non-crash, Minor crash and Severe crash). Classification problems where target categories are ordered are known as ordinal classification problems. In principle, we should shape the error function, needed to optimize model parameters, such that misclassifications from non-crashes to severe crashes should provide more error signal, compared to misclassifications from severe crashes to minor crashes and from non-crashes to minor crashes. A suitable approach to solve this kind of problems is to treat them as standard regression tasks with an appropriate loss function (e.g., l_2 distance function). However, distances between categories are not well defined and finding an optimal mapping from the categorical space to $\mathbb{R}$ is not a trivial task. For this reason, following a similar approach as the one used in [32], we reformulate the ordinal multiclass classification problem as two distinct binary problems. In particular, we stack on top of the backbone introduced in the previous paragraph two distinct prediction MLP heads. A first head predicts the probability of a generic crash (Minor + Severe), given the input time series:

$$\mathrm{P}_{\text{crash}} = \mathrm{P}(\text{crash} \mid a(t),\, s(t),\, g(t)). \tag{4}$$

A second head predicts the probability of a severe crash, given the input time series and the fact that a crash occurred, i.e.,

$$\mathrm{P}_{\text{severe|crash}} = \mathrm{P}(\text{severe} \mid a(t),\, s(t),\, g(t),\, \text{crash}). \tag{5}$$

The posterior probability of a severe crash is then computed through Bayes' rule as:

$$\mathrm{P}_{\text{severe}} = \mathrm{P}(\text{severe} \mid a(t),\, s(t),\, g(t)) = \mathrm{P}_{\text{crash}}\mathrm{P}_{\text{severe|crash}}. \tag{6}$$

With this formalization we can treat both $\mathrm{P}_{\text{crash}}$ and $\mathrm{P}_{\text{severe}}$ as output of binary classifiers, and we can optimize the model parameters using binary cross-entropy loss functions. Figure 2 shows the whole architecture used to implement this policy. Referring to the naming convention used in the introduction of this section, the features obtained at the output of the backbone $v = f_\psi(a(t), s(t), g(t))$ are used in two different streams. In the upper branch they are mapped through a multilayer perceptron (MLP), Crash Prediction Head, to crash posterior probabilities $\mathrm{P}_{\text{crash}} = \sigma(f_{\phi_1}(v))$ (where σ denotes the sigmoid logistic function). Crash logits are then concatenated to the output features of the backbone, composing $\tilde{v} = \text{Concat}(v, f_{\phi_1}(v))$. Vector $\tilde{v}$ is subsequently mapped, in the lower branch of the block scheme, through another MLP head, Severe Prediction Head, to conditional severe crash posterior probabilities $\mathrm{P}_{\text{severe|crash}} = \sigma(f_{\phi_2}(\tilde{v}))$. The output of the lower branch $\mathrm{P}_{\text{severe}}$ is then obtained by multiplication with $\mathrm{P}_{\text{crash}}$ as shown in Eq. (6). Given the two possible binarizations of ground truth labels y_{crash} (positive in case of Minor and Severe crashes) and y_{severe} (positive only in case of Severe crashes), the loss function we optimize is:

$$L = \text{BCE}(\mathrm{P}_{\text{crash}}, y_{\text{crash}}) + \text{BCE}(\mathrm{P}_{\text{severe}}, y_{\text{severe}}) \tag{7}$$

where BCE denotes the binary cross-entropy loss function. During inference, we select two distinct thresholds θ_{crash} and θ_{severe} (with $\theta_{\text{crash}} \leq \theta_{\text{severe}}$) that are used to map the two binary posterior probabilities back to the original three classes.

$$\hat{y} = \begin{cases} \text{Non-crash} & \mathrm{P}_{\text{crash}} < \theta_{\text{crash}} \\ \text{Minor crash} & \mathrm{P}_{\text{crash}} \geq \theta_{\text{crash}},\ \mathrm{P}_{\text{severe}} < \theta_{\text{severe}} \\ \text{Severe crash} & \mathrm{P}_{\text{severe}} \geq \theta_{\text{severe}} \end{cases}$$

The adoption of the multiplicative scheme presented in this section, together with the constraint on the thresholds ($\theta_{\text{crash}} \leq \theta_{\text{severe}}$), will completely avoid ill-posed predictions, i.e., the one in which $\mathrm{P}_{\text{crash}} < \theta_{\text{crash}}$ and $\mathrm{P}_{\text{severe}} \geq \theta_{\text{severe}}$.

C. Data Augmentation

Data augmentation techniques allow generating new artificial data points by applying random perturbations on real data samples belonging to a dataset. The applied perturbations must have the property of not disrupting the semantic content hidden in the samples (i.e., they should not jeopardize the association between data points and ground truth information). In case of vehicle-related inertial measurements, semantic-preserving transformations are not obvious because they should match feasibility constraints of motion. For this reason, we developed a data-driven approach to select the family of augmentations suitable for our problem, considering several different possible data augmentation techniques and keeping only the ones that gave a performance boost in the underlying classification task. All the augmentations we describe give a positive contribution, with no single transformation significantly outperforming the rest.

The data augmentation pipeline is a crucial ingredient of our method for a twofold reason:

- it allows us to leverage recently developed contrastive self-supervised pre-training methods, founded on the ability of generating many possible plausible training samples from a single real sample contained in our dataset;
- it helps in preventing overfitting (acting as a regularizer) for the minority class in an imbalanced classification problem, thus improving the generalization.

In mathematical terms, we seek a family T of transformations from which we can extract a deterministic function $t \sim T$ such that given a sample (X, y) contained in our dataset, $(t(X), y)$ is another suitable sample that can be used during the training of the model, i.e., y is still a correct label for $t(X)$. In the next subsections, we formally introduce all the data augmentation techniques used in our work.

1) Random Rotation: Through the application of a random rotation matrix we mimic a different mounting position of the inertial sensor, without any risk of compromising the semantic content associated to ground truth labels. In practice, a random rotation matrix is created sampling 3 random Euler angles (α, β, γ). The overall rotation is found by composing the three Euler rotation matrix around canonical axis through matrix multiplication (i.e., $R = R_x(\alpha)R_y(\beta)R_z(\gamma)$).

2) Colored Gaussian Noise: The superposition of low-passed white Gaussian noise to the acceleration and gyroscope signals mimics different vehicle background vibrations due to different road environments and road conditions. For each training sample, we extract standard deviation values uniformly distributed in a plausible interval and then we extract zero-mean Gaussian noise samples, independently for each sensor and sensor axis, using these random standard deviation values. Before adding the noise to the signal components, we color it through the application of a low-pass filter with a random normalized cut-off frequency in a range $[\nu_1, \nu_2]$.

3) Random Cropping: Similar to image cropping, used in computer vision applications, we extract a shorter random contiguous sub-sequence from the original time series.

4) Random Permutation: Random permutation works independently on each sensor and each sensor axis: it extracts a random number of short fixed-length time windows in which values are randomly permuted with a given probability.

5) Magnitude Warping: This data augmentation technique mimics possible random changes in the amplitude of a time series. It is applied on gyroscope and acceleration signals, independently for each sensor axis. The idea is to modulate each signal with a slow-varying envelope, extracted from a Gaussian distribution with unitary mean and standard deviation σ. To generate the modulation envelope a set of few values v_i (from 6 to 10) are extracted independently from a normal distribution $v_i \sim N(1, \sigma)$. Through cubic spline interpolation, these values are interpolated in the number of samples N composing the time series $\tilde{v} = \text{CubicSpline}(v, N)$, giving smoothness to the random envelope. Each sensor axis x is finally multiplied by this slow-varying envelope to obtain the resulting output signal $y = \tilde{v}x$.

6) Time Warping: Time warping mimics a distortion of the sample time of a digital signal. It is applied coherently to both gyroscope and acceleration, in all their sensor axis. Assume that the signals are composed of N time points and that their discrete time axis is the vector $t = [0, 1, \ldots, N-1]$. This augmentation is applied through the extraction of a new time axis from a white normal distribution centered in t, i.e., $\tilde{t} \sim N(t, \sigma I)$. The elements of $\tilde{t}$ are then clipped between 0 and $N-1$ in order to avoid extrapolation. Once $\tilde{t}$ is computed we interpolate the original signals in this new time axis.

7) Window Warping: This augmentation aims to randomly increase or decrease the duration of a random time slice of a signal $x(t)$, without altering too much the samples outside this window. Given a random signal slice defined through a random temporal interval, i.e., $x_W(t) = \{x(t) : t_l \leq t \leq t_h\}$, we can decompose the original signal $x(t) = \text{Concat}(x_B(t), x_W(t), x_A(t))$, where Concat denotes the temporal concatenation, $x_B(t) = \{x(t) : t < t_l\}$ is the signal portion before the random signal slice, and $x_A(t) = \{x(t) : t > t_h\}$ the one after. The random signal slice is then upsampled or downsampled with equal probability by a factor 2, i.e., we compute $x_W(\alpha t)$ with $\alpha \in \{0.5, 2\}$. Finally, the signal $y(t) = \text{Concat}(x_B(t), x_W(\alpha t), x_A(t))$ is interpolated back to the original duration through cubic spline interpolation.

8) Window Slicing: We apply random cropping followed by a temporal upsampling. A large random time window is cropped from the original time series, stretched to the original length, and interpolated back to the original time indices through cubic interpolation.

9) Speed Dropout: The whole speed signal is set to zero with a given probability. We measured a drastic performance improvement by introducing this data augmentation technique during the self-supervised pre-training. We hypothesize that this happens because most other data augmentation techniques (except for random cropping) do not have any significant effect on the slow-varying speed signal. This would make the self-supervised task easy to solve just by computing an embedding of the speed signal and ignoring the other two sensors.

D. Self-Supervised Pre-Training

Given the extremely imbalanced problem due to the small number of crashes in our dataset, we use a form of contrastive self-supervised learning as a way to obtain a pre-trained backbone to help the supervised training on our downstream task. In a generic contrastive self-supervised framework, two augmented views of a dataset point x are generated sampling two data augmentation operators from the same family of augmentations T, i.e., $t_1 \sim T$, $t_2 \sim T$, $x^{(1)} = t_1(x)$, $x^{(2)} = t_2(x)$. The network is then trained to match the output representations produced from $x^{(1)}$ and $x^{(2)}$ through the maximization of a similarity score (e.g., cosine similarity). This process results in models that learned to extract meaningful features and can be easily fine-tuned on the task at hand.

In more details, in our work, we perform the self-supervised pre-training step of our backbone model by adapting the SimSiam framework [21] to our multimodal setting. Given a sample $x = [a(t), s(t), g(t)]$, a shared backbone b followed by a projection MLP head f process two views ($x^{(1)}$ and $x^{(2)}$) of x. Let then $v^{(1)} = b(x^{(1)})$, $v^{(2)} = b(x^{(2)})$, $k^{(1)} = f(v^{(1)})$, $k^{(2)} = f(v^{(2)})$. A predictor MLP head h transforms the projector output of one view to match the one of the other view: $p^{(1)} = h(k^{(1)})$, $p^{(2)} = h(k^{(2)})$. The SimSiam framework performs the matching in terms of maximization of cosine similarity (i.e., minimization of negative cosine similarity, see Eq. 8, where $\| \cdot \|$ denotes the l_2 norm, and $\langle \cdot | \cdot \rangle$ the dot product).

$$D(x, y) = -\frac{\langle x | y \rangle}{\|x\| \|y\|} \tag{8}$$

Such equation allows for a degenerate solution in which every sample is mapped into a fixed constant output vector. Surprisingly, the authors of [21] show, through an exhaustive set of empirical results, that a stop-gradient operation (SG) is

enough to prevent this issue. Equation 9 shows the complete loss function used in [21] for a single dataset sample:

$$L = \frac{1}{2}D(p^{(1)}, \texttt{SG}(k^{(2)})) + \frac{1}{2}D(p^{(2)}, \texttt{SG}(k^{(1)})) \quad (9)$$

An out-of-the-box application of the SimSiam framework to our method is to consider only the final output of our backbone (see Fig. 1) as input for the SimSiam projector head, and use the loss L in Eq. (9). We propose a change in the SimSiam loss for multimodal inputs that significantly improves the performance on the downstream task, as we will show in our experiments. More specifically, our idea is to add additional loss terms depending on the intermediate backbone outputs, similar to what is done with *deep supervision* [33].

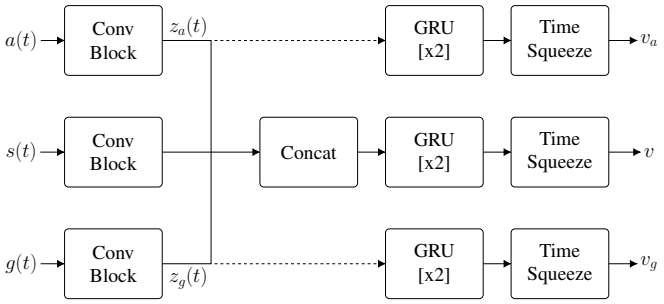

Fig. 3: Multimodal SimSiam - Backbone Modifications: Two additional branches are attached to the backbone to produce acceleration and gyroscope specific feature vectors (v_a and v_g respectively).

Figure 3 shows the modifications we apply to our backbone to adapt the SimSiam framework to our multimodal setting. We attach two auxiliary branches[1] (composed of bidirectional GRU and Time Squeeze operations as presented in Section IV-A), one working on the convolutional embedding extracted from the acceleration signal z_a, and one working on the gyroscope embedding z_g, outputting two additional feature vectors v_a and v_g, respectively.

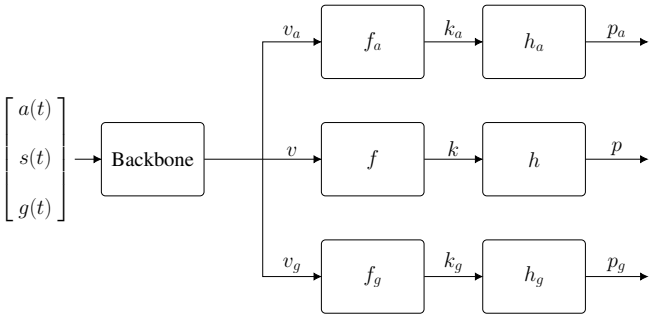

Fig. 4: Multimodal SimSiam - Overall Architecture: The backbone outputs (v, v_a, v_g) are fed to 3 different projection + prediction heads to compute three separate SimSiam loss terms.

Figure 4 shows the overall block scheme of the multimodal version of SimSiam. To each backbone output feature vectors (v, v_a, v_g), we attach independent projector and predictor heads: [f, h] for v, [f_a, h_a] for v_a, and [f_g, h_g] for v_g. From the two additional branches, we compute the following loss terms:

$$L_a = \frac{1}{2}D(p_a^{(1)}, \texttt{SG}(k_a^{(2)})) + \frac{1}{2}D(p_a^{(2)}, \texttt{SG}(k_a^{(1)})) \quad (10)$$

$$L_g = \frac{1}{2}D(p_g^{(1)}, \texttt{SG}(k_g^{(2)})) + \frac{1}{2}D(p_g^{(2)}, \texttt{SG}(k_g^{(1)})) \quad (11)$$

that are combined with the loss L from the central branch (as in Eq. (9)) to obtain the following multimodal SimSiam loss:

$$L_{\text{MM}} = \frac{1}{3}(L + L_a + L_g) \quad (12)$$

The additional loss terms L_a and L_g, depending only on gyroscope and acceleration embedding, force the backbone to focus more on these sensor sources, rather than the speed, that are more discriminative for the underlying crash detection downstream task. In section V, we empirically show that this modification, that keeps into account the multiple modalities of our input sensor streams, gives a considerable gain in terms of generalization capabilities on the downstream task.

E. Supervised Fine-Tuning with Negative Sampling

Once the backbone has been pre-trained using the self-supervised technique introduced in the previous paragraph, we attach the prediction heads needed to solve the ordinal downstream classification task following the formulation presented in Section IV-B. To compensate for the evident class imbalance present in our dataset, we adopt a training strategy where, at each training epoch, we only select a random subset of negative (non-crash) examples. Supposing that the total number of accidents (Minor or Severe) in our training set is K and the batch size used in the training procedure is B, at the beginning of each training epoch, we sample $K \cdot (B-1)$ negative samples. We then populate the training data for that specific training epoch with all the K accidents present in our dataset plus $K \cdot (B-1)$ randomly selected non-accident samples, for a total of $K \cdot B$ training samples. The idea behind this training technique is that, on average, each training batch will contain a crash sample and $B-1$ non-crash samples.

V. Results

We present our experimental results using the dataset introduced in Section III. For all the experiments in this section, we employ early-stopping on the validation set for selecting the best model obtained during the training procedure, and we show the results on the test set obtained using the best validation model. We measure the performance using the area under the precision-recall (PR) curve, also known as average-precision score (AP), which is particularly suitable for imbalanced classification problems. This score is obtained by integrating the PR curve that provides precision and recall points for different values of binary decision threshold: it assumes a value of 1 for a perfect classifier, and it decreases for each false positive misclassification performed by the model. Because with our model formulation (presented in Section IV-B) we are explicitly solving two distinct binary problems, we report performance metrics on the generic crash

[1]These two auxiliary branches are only used for self-supervised pre-training, and are afterwards discarded during the supervised fine-tuning.

TABLE II: Architecture details (backbone)

a(t), 3	**g**(t), 3	**s**(t), 1
11 **Conv**, 16	11 **Conv**, 16	11 **Conv**, 16
7 **Conv**, 24	7 **Conv**, 24	7 **Conv**, 24
5 **Conv**, 48	5 **Conv**, 48	5 **Conv**, 48
5 **Conv**, 64	5 **Conv**, 64	5 **Conv**, 64
Concat		
GRU, 512		
Dropout 0.5		
GRU, 512		
Time Squeeze		
Output Features, 512		

class (Minor + Severe) and the Severe one. We organize this section as follows:

- We show the implementation details of our approach: model structure, training, and augmentation strategies.
- We report a set of extensive experiments to compare the performance of our pipeline with several baselines, and discuss the impact of each contribution.
- We analyze computational requirements for training and inference.

A. Implementation Details

Table II shows the details of each layer used in our backbone: Conv layers are a stack of [Convolution 1D $\rightarrow$ Batch Norm 1D $\rightarrow$ ReLu $\rightarrow$ Max Pooling], and for each block we indicate the filter length and the number of output channels, respectively. The stack of bidirectional gated recurrent units is composed of two layers of size 512. We found dropout (with a dropout probability of 0.5) beneficial to improve the generalization capabilities of our model if applied after the output of the first GRU layer. After the Time Squeeze block, the output of the backbone is a 512-dimensional feature vector. The MLPs used for the prediction heads are composed of a single hidden layer with 256 units and ReLu non-linearity.

We optimize the self-supervised loss function using Stochastic Gradient Descent (SGD) optimizer (momentum = 0.9, batch size = 256), for 200 epochs with 10 epochs of linear warm-up (with maximum learning rate of 0.05) followed by 190 epochs of cosine annealing. During the downstream supervised fine-tuning we train for 20 epochs with Adam optimizer (learning rate = 10^{-4}, for both the backbone and the prediction heads, batch size = 64). We tried to use a lower learning rate for the backbone, but we obtained a negligible improvement, thus we decided to keep the simpler configuration.

Table III shows the parameters employed in our data augmentation pipeline for both self-supervised pre-training and supervised fine-tuning. During the self-supervised pre-training, we employ an aggressive data augmentation policy: we apply all the data augmentation techniques (including Speed Dropout) with only a small probability (0.1) of skipping an augmentation step. For the supervised training, instead, we use a different data augmentation strategy: with probability 0.5 for each step, we apply a Random Rotation, we add Colored Gaussian Noise and then a single augmentation extracted randomly in [Random Permutation, Magnitude Warping, Time Warping, Window Warping, Window Slicing]. In all cases, Random Cropping is always applied at the end.

B. Model Benchmarks

In this section, we show the performance gain introduced by our model, alongside our custom training procedure (self-supervised pre-training + supervised fine-tuning), compared to state-of-the-art deep neural networks for time series classification and thresholds based methods.

Following the suggestions provided in [7] and [9], we compare with a one-dimensional ResNet[2] and InceptionTime as strong NN models for TSC.

Then, we build a threshold-based baseline inspired by [24], [25] ("Baseline" in the following figures and tables). We only consider the acceleration on the longitudinal and transversal vehicle axis (x and y). The axis orthogonal to the ground plane is discarded because it often shows large amplitude peaks in correspondence to speed bumps and potholes. We compress both x and y components into a single value, obtaining the norm $a_{xy}(t) = \sqrt{a_x(t)^2 a_y(t)^2}$. We apply a low-pass filter with a cut-off frequency of 0.25 Hz to reduce false positives due to noise. Then, we take the maximum of this smoothed acceleration norm, i.e., $a^* = \max_t a_{xy}(t)$, as the value we use for the final classification via a threshold.

In Table IV we report the AP score obtained on our test set for the different methods. Figure 5 shows the full Precision-Recall (PR) curves. From the reported results, we

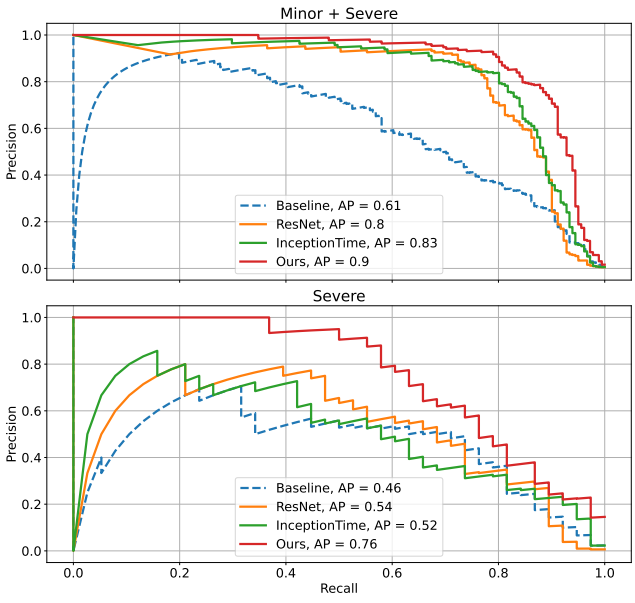

Fig. 5: PR curves for generic crash detection (Minor + Severe crashes vs Non-crashes) on top, and for Severe only (Severe vs rest) on bottom. Dashed blue lines refer to the threshold-based baseline, orange lines refer to ResNet, green lines to InceptionTime, and the red lines to our model. The first point is set to Precision=1, Recall=0 for all methods by convention.

can appreciate a huge performance gain of our proposed method ($+0.24$, more than 40% improvement) in the detection

[2]As in [7], we use a standard ResNet model for image classification where bi-dimensional blocks are replaced by their mono-dimensional counterparts.

TABLE III: Data augmentation parameters during self-supervised and supervised training. For each transformation, we report its parameter range and the sensors on which it is applied. With σ_D we denote the sensor-specific standard deviation computed over the training set.

	Self-Supervised	**Supervised**	**Sensors**
Random Rotation - Angle Range	[-45°, +45°]	[-10°, +10°]	Acceleration, Gyroscope
Colored Noise - σ Range	$[\sigma_D/100, \sigma_D/10]$	$[\sigma_D/100, \sigma_D/10]$	Acceleration, Gyroscope
Colored Noise - $[\nu_1, \nu_2]$	[0.1, 1.0]	[0.1, 1.0]	Acceleration, Gyroscope
Random Cropping - Crop Length	14 s	14 s	Acceleration, Gyroscope, Speed
Random Permutation - Window Length	20 ms	20 ms	Acceleration, Gyroscope
Random Permutation - Permutation Probability	0.5	0.5	Acceleration, Gyroscope
Magnitude Warping - σ	$\sigma_D/3$	$\sigma_D/3$	Acceleration, Gyroscope
Time Warping - σ	0.2	0.2	Acceleration, Gyroscope
Window Warping - Window Duration % Range	[5%, 15%]	[5%, 15%]	Acceleration, Gyroscope
Window Slicing - Window Duration % Range	[90%, 100%]	[90%, 100%]	Acceleration, Gyroscope
Speed Dropout - Probability	0.7	0	Speed

TABLE IV: Test performance (best model selected on validation)

Method	**Crash (Minor + Severe) AP**	**Severe Crash AP**
Baseline	0.61	0.46
ResNet [7]	0.8	0.54
InceptionTime [9]	0.83	0.52
Ours	**0.90**	**0.76**

of Severe accidents compared to the second best. All other methods have much worse precision on the highest-confidence predictions. Also on the Minor+Severe task, our method significantly outperforms the other approaches (+0.07, around 9% improvement). We believe that this performance gain is due to several contributions: the backbone architecture, specifically designed for multimodal inputs; the custom prediction head and loss, that fully exploit the intrinsic order of the classes; and our custom training procedure that, through self-supervised pre-training and domain-specific data augmentations, prevents overfitting and provides better generalization capabilities – crucial aspects on such extremely imbalanced problems. In the following two subsections we provide a deeper discussion of each of these components.

C. Self-Supervised Pre-Training - Discussion

We show the performance gain introduced by the self-supervised pre-training by estimating the probability distribution of the AP score (in terms of mean and standard deviation) obtained by fine-tuning the pre-trained model multiple times[3] and comparing it with the AP scores obtained with models trained from scratch with randomly initialized weights. Moreover, following a common practice used in self-supervised learning [14], we estimate the AP score obtained when only a subset of training samples is available (25% and 50%), to show that self-supervised pre-trained models generalize better when trained on fewer annotated samples, compared to their randomly initialized counterpart.

Figure 6 shows AP distributions (Minor + Severe on top and Severe on bottom) described in terms of mean (diamond marker) and standard deviation (vertical line) for different fractions of training samples used during supervised learning and different backbone weights initialization methods. We consider three possible initialization strategies:

[3]Even starting from a self-supervised pre-trained backbone, training is not deterministic (prediction heads are randomly initialized).

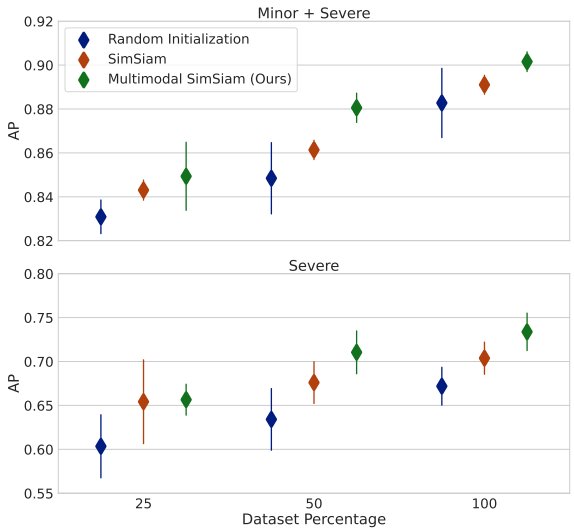

Fig. 6: Self-supervised pre-training - AP comparison: The x-axis indicates the percentage of training samples used during supervised learning. Blue distributions refer to the models trained from scratch with random initialization, red ones to models pre-trained using vanilla SimSiam, and green ones to models pre-trained with our multimodal SimSiam. The top figure refers to the generic crash detection problem (Minor + Severe), the bottom to the Severe one.

1) Random Initialization: we randomly initialize the weights of the network (both backbone and prediction heads), and we train it from scratch.
2) SimSiam: we pre-train the backbone accordingly to the SimSiam framework without the multimodal modifications (see Section IV-D). Once the backbone is pre-trained, we attach the randomly initialized prediction heads and fine-tune the whole architecture.
3) Multimodal SimSiam: we perform the self-supervised pre-training step using the multimodal version of the SimSiam framework (see Section IV-D).

The distributions of the AP scores shows a clear performance boost when the backbone is pre-trained using a self-supervised method even when very few training samples are available. Our modification of the SimSiam framework, accounting for multiple modalities, introduces a noticeable improvement compared both to the standard version of SimSiam and to a standard random initialization. Note in particular, that for the detection of Severe crashes, our multimodal self-supervised

pre-training allows us to reach a superior performance (in terms of AP) with only *half* of the training samples compared to a supervised training using the *entire* dataset. This is of the utter importance in such a problem, where very few (and extremely rare) positive samples are available.

D. Data Augmentation - Discussion

To prove the effectiveness of our custom data augmentation pipeline during supervised training[4], we turn it on and off, and for both cases, we estimate the distribution of the AP scores over 10 runs.

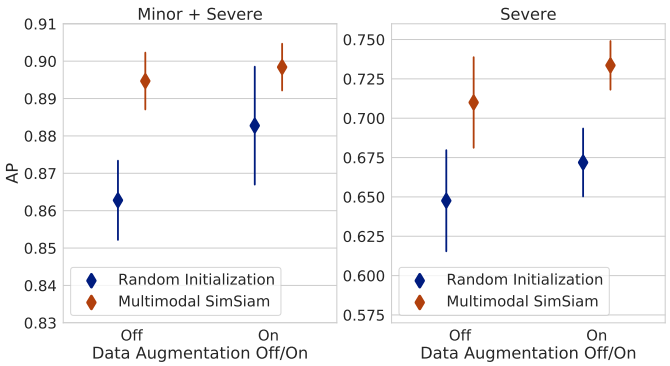

Fig. 7: Data Augmentation Off/On - AP comparison: Blue distributions refer to the models randomly initialized and trained from scratch, red ones to models fine-tuned starting from weights obtained with Multimodal SimSiam. The figure on the left refers to the generic crash class (Minor + Severe), the one on the right to the Severe class.

Figure 7 shows AP scores obtained when we activate or deactivate the data augmentation pipeline. When we train the model starting from randomly initialized weights, our data augmentation pipeline plays a very big role in counteracting overfitting, yielding an average AP gain of 0.02 points for Severe and 0.03 points for Severe+Minor. In the case where we fine-tune starting from initial weights obtained with Multimodal SimSiam (in red in Fig. 7), only a small number of training epochs are needed to reach convergence, and the performance gain introduced by data augmentation is smaller. Still, it is noticeable especially for Severe crashes, where we obtain a 0.02 improvement.

E. Ordinal Classification - Discussion

To motivate our choice of formulating the problem as an ordinal multiclass problem, we perform a study where we compare our proposed approach, that has two prediction heads and a custom loss (see Section IV), with a standard categorical cross-entropy loss. We use the same backbone, but using a single 3-dimensional prediction head with a softmax activation. Compared with our proposed custom architecture and loss, the AP (average on 10 runs) drops from 0.883 to 0.862 and from 0.672 to 0.614 on generic and severe crashes respectively, when training randomly initialized models.

[4]This ablation study does not refer to data augmentation during the contrastive pre-training procedure: without strong data augmentation, it collapses reaching no meaningful result.

F. Computational requirements - Discussion

Our proposed model is compact, with only 2.5M parameters, resulting in a small memory footprint (around 11MB). For comparison, ResNet and InceptionTime have respectively more than 4.4M and 4.9M parameters. To run inference on a 16-second sample, our model takes 0.032 seconds on a standard Intel CPU (i7-8665U @1.90GHz) and 0.095 seconds on a Cortex-A72 CPU @1.5GHz, a common ARM CPU used in phones and embedded devices. Regarding the training time, it requires around 4 hours for self-supervised pre-training on a NVidia V100 GPU, and around 45 minutes for fine-tuning.

VI. Conclusion

In this paper, we present a novel deep learning pipeline for crash detection from vehicular sensor data. The proposed pipeline is composed of a custom neural architecture, a self-supervised pre-training technique, and a set of data augmentation functions. We formulate the problem of detecting accidents alongside their severity from multimodal sensor streams as an ordinal multiclass classification problem. We validate our approach on a big dataset of IMU/GPS time series, containing 233051 samples of which only 0.56% are crashes. For the first time on a similar task, we employ a self-supervised pre-training strategy together with a custom pipeline of data augmentations to improve the generalization capabilities in this extremely imbalanced setting. We carefully perform ablation studies to show the impact of particular design choices, and we compare our approach with state-of-the-art models for time series classification. Our approach reaches an AP score of 0.90 in the detection of generic crashes, and a score of 0.76 in the detection of the extremely rare *severe* crashes, which correspond to an improvement of 9% and 40%, respectively, when compared to the state-of-the-art models InceptionTime and ResNet. Our experiments show that domain-specific augmentation techniques and self-supervised pre-training are instrumental in significantly improving the performance of our model.

Future research directions might consider video data to enrich multimodal input sensor streams, providing additional semantic understandings of the crash scene: consider, for instance, the problem of identifying and describing the cause and dynamics of the crash. Another interesting development would be to study if the proposed architecture could be made even smaller and faster, with neural architecture search or other network optimization techniques. Finally, we also believe that the proposed approach and data augmentations could be successfully applied to other problems with similar input data.

References

[1] L. Taccari, F. Sambo, L. Bravi, S. Salti, L. Sarti, M. Simoncini, and A. Lori, "Classification of crash and near-crash events from dashcam videos and telematics," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2460–2465.

[2] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[3] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 565–592, 2015.

[4] A. Bostrom and A. Bagnall, "Binary shapelet transform for multiclass time series classification," in *International conference on big data analytics and knowledge discovery*. Springer, 2015, pp. 257–269.

[5] J. Lines, S. Taylor, and A. Bagnall, "Time series classification with hivecote: The hierarchical vote collective of transformation-based ensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 5, 2018.

[6] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data mining and knowledge discovery*, vol. 28, no. 4, pp. 851–881, 2014.

[7] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] H. I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "InceptionTime: Finding AlexNet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, 2020.

[10] M. Hüsken and P. Stagge, "Recurrent neural networks for time series classification," *Neurocomputing*, vol. 50, pp. 223–235, 2003.

[11] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[12] J.-C. Su, S. Maji, and B. Hariharan, "When does self-supervision improve few-shot learning?" in *European Conference on Computer Vision*. Springer, 2020, pp. 645–666.

[13] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *arXiv preprint arXiv:2004.11362*, 2020.

[14] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[15] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[16] A. Saeed, T. Ozcelebi, and J. Lukkien, "Multi-task self-supervised learning for human activity detection," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 2, pp. 1–30, 2019.

[17] J.-Y. Franceschi, A. Dieuleveut, and M. Jaggi, "Unsupervised scalable representation learning for multivariate time series," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[19] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[20] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar *et al.*, "Bootstrap your own latent: A new approach to self-supervised learning," *arXiv preprint arXiv:2006.07733*, 2020.

[21] X. Chen and K. He, "Exploring simple siamese representation learning," *arXiv preprint arXiv:2011.10566*, 2020.

[22] B. K. Iwana and S. Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *arXiv preprint arXiv:2007.15951*, 2020.

[23] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," in *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016.

[24] M. Hannan, A. Hussain, and S. Samad, "Sensing systems and algorithms for airbag deployment decision," *IEEE Sensors Journal*, vol. 11, no. 4, pp. 888–890, 2010.

[25] S. M. Mahmud and A. I. Alrabady, "A new decision making algorithm for airbag control," *IEEE transactions on vehicular technology*, vol. 44, no. 3, pp. 690–697, 1995.

[26] S. Amin, M. B. I. Reaz, and S. S. Nasir, "Integrated vehicle accident detection and location system," *Telkomnika*, vol. 12, no. 1, p. 73, 2014.

[27] M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni, "A system for automatic notification and severity estimation of automotive accidents," *IEEE Transactions on mobile computing*, vol. 13, no. 5, pp. 948–963, 2013.

[28] S. Gelmini, S. Strada, M. Tanelli, S. Savaresi, and C. De Tommasi, "A novel crash detection algorithm for two-wheeled vehicles," *IEEE Transactions on Intelligent Vehicles*, 2020.

[29] S. Park, Y. Seonwoo, J. Kim, J. Kim, and A. Oh, "Denoising recurrent neural networks for classifying crash-related events," *IEEE transactions on intelligent transportation systems*, vol. 21, no. 7, pp. 2906–2917, 2019.

[30] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *ICML*, 2011.

[31] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[32] J. Cheng, Z. Wang, and G. Pollastri, "A neural network approach to ordinal regression," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 2008, pp. 1279–1284.

[33] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Artificial intelligence and statistics*. PMLR, 2015, pp. 562–570.

Luca Kubin received the MSc degree in communication engineering from the University of Parma in 2015. His research interests include deep learning, computer vision, and digital signal processing.

Tommaso Bianconcini received the MSc degree in Mathematics and the PhD in Operations Research, both from the University of Florence, Italy, in 2011 and 2015 respectively. He is currently Research Engineer at Verizon Connect. His research interests include deep learning and machine learning in general, computer vision and vehicle routing.

Douglas Coimbra de Andrade graduated mechanical aeronautical engineer in 2005 and received his D. Sc. in the field of Aerospace Systems and Mechatronics in 2017, both from Instituto Tecnologico de Aeronautica, Brazil. His research interests include AI applied to computer vision, video analytics and HPC.

Matteo Simoncini received the MSc degree in computer engineering from the University of Florence, Italy, in 2016. He is currently working toward an industrial Ph.D. degree at Verizon Connect Research, Florence, Italy and the University of Florence, Italy. His research interests include intelligent transportation systems, machine learning, computer vision and their applications.

Leonardo Taccaril received the Ph.D. in Operations Research from Politecnico di Milano in 2015. He is currently lead scientist at Verizon Connect. He is author or coauthor of more than 20 scientific articles and 10 patents. His research interests include machine learning and mathematical optimization applied to transportation, logistics, and energy.

Francesco Sambo holds a PhD in bioinformatics and artificial intelligence from the university of Padova. He is currently Chief Data Scientist at Verizon Connect. His research interests include road scene understanding and predictive maintenance. He is author or coauthor of more than 40 scientific papers and 10 patents.