UFR 6
Université de Montpellier Paul Valéry

Mémoire Professionnel S1M2

Analyse Comparative d'Architectures Deep Learning pour la Prédiction de Collisions Routières par Vision par Ordinateur

VITOFFODJI Adjimon

Janvier 2026

Présenté par :
VITOFFODJI Adjimon

Tuteur Universitaire :
Jérôme PASQUET

Remerciement

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce mémoire.

En premier lieu, je remercie chaleureusement mon tuteur universitaire, **Jérôme PASQUET**, pour son encadrement, ses conseils avisés et sa disponibilité tout au long de ce projet. Son expertise en intelligence artificielle et sa rigueur scientifique m'ont permis de mener à bien cette recherche.

Je remercie également l'ensemble de l'équipe pédagogique du Master 2 MIASHS de l'Université de Montpellier Paul Valéry pour la qualité de la formation dispensée et les connaissances acquises durant ces deux années.

Mes remerciements vont aussi à la communauté **Kaggle** et aux organisateurs du **Nexar Collision Prediction Challenge**, qui ont mis à disposition ce dataset de haute qualité permettant de mener des recherches appliquées en vision par ordinateur.

Je remercie également l'équipe d'Anthropic pour le développement de Claude ANTHROPIC, 2026, assistant IA qui a facilité le développement technique tout en respectant mon intégrité académique et ma paternité intellectuelle sur l'ensemble des travaux.

VITOFFODJI Adjimon
Janvier 2026

Résumé

La prédiction précoce de collisions routières représente un enjeu majeur pour la sécurité routière, avec plus de 1,3 million de décès annuels dans le monde. Ce mémoire présente une analyse comparative systématique de six architectures de deep learning pour la prédiction de collisions à partir de vidéos dashcam, dans le cadre du Challenge Kaggle Nexar.

Notre méthodologie repose sur l'évaluation expérimentale de trois familles d'architectures : les modèles hybrides CNN-RNN (ResNet-LSTM, EfficientNet-GRU), les CNN 3D natifs (I3D, R(2+1)D), et les Vision Transformers (TimeSformer, VideoMAE). L'ensemble des modèles a été entraîné sur un dataset de 1,500 vidéos annotées et évalué selon la métrique Mean Average Precision (mAP).

Les résultats expérimentaux révèlent plusieurs enseignements clés. Les modèles 3D CNN pré-entraînés sur Kinetics-400 obtiennent les meilleures performances, avec I3D atteignant 77,53% d'Average Precision en validation et 71,2% sur le test set Kaggle. Les architectures hybrides offrent un excellent compromis vitesse-performance, EfficientNet-GRU atteignant 71% d'accuracy et 74,95% d'AP. En revanche, les Vision Transformers sans pré-entraînement vidéo échouent complètement (TimeSformer : 50,67%), démontrant l'importance critique du pre-training sur données vidéo pour ces architectures.

Cette recherche établit des recommandations pratiques pour le développement de systèmes ADAS : privilégier les CNN 3D pour maximiser la performance de prédiction, opter pour des architectures hybrides pour les contraintes temps réel, et systématiquement utiliser du pré-entraînement sur datasets vidéo massifs pour les architectures transformer. Les perspectives incluent l'optimisation des modèles via techniques d'interprétabilité et le déploiement en conditions réelles.

Mots-clés : Deep Learning, Vision par Ordinateur, Prédiction de Collisions, CNN 3D, Vision Transformers, ADAS, Sécurité Routière, Kaggle

Abstract

Early collision prediction represents a major challenge for road safety, with over 1.3 million annual deaths worldwide. This thesis presents a systematic comparative analysis of six deep learning architectures for collision prediction from dashcam videos, as part of the Kaggle Nexar Challenge.

Our methodology is based on experimental evaluation of three architecture families : hybrid CNN-RNN models (ResNet-LSTM, EfficientNet-GRU), native 3D CNNs (I3D, R(2+1)D), and Vision Transformers (TimeSformer, VideoMAE). All models were trained on a dataset of 1,500 annotated videos and evaluated using the Mean Average Precision (mAP) metric.

Experimental results reveal several key insights. 3D CNN models pre-trained on Kinetics-400 achieve the best performance, with I3D reaching 77.53% Average Precision on vali-

dation and 71.2% on the Kaggle test set. Hybrid architectures offer an excellent speed-performance trade-off, with EfficientNet-GRU achieving 71% accuracy and 74.95% AP. However, Vision Transformers without video pre-training fail completely (TimeSformer : 50.67%), demonstrating the critical importance of pre-training on video data for these architectures.

This research establishes practical recommendations for ADAS system development : favor 3D CNNs to maximize prediction performance, opt for hybrid architectures for real-time constraints, and systematically use pre-training on massive video datasets for transformer architectures. Future work includes model optimization through interpretability techniques and deployment in real-world conditions.

Keywords : Deep Learning, Computer Vision, Collision Prediction, 3D CNN, Vision Transformers, ADAS, Road Safety, Kaggle

Table des matières

Table des figures

Liste des tableaux

Contexte et Problématique

Les accidents de la route constituent l'une des principales causes de mortalité dans le monde, avec plus de 1,3 million de décès recensés annuellement selon l'Organisation Mondiale de la Santé (World Health Organization, 2023). Face à ce constat alarmant, le développement de systèmes d'assistance à la conduite (ADAS - Advanced Driver Assistance Systems) et de véhicules autonomes représente un enjeu sociétal majeur. Au cœur de ces technologies se trouve la capacité à **anticiper les collisions** avant qu'elles ne se produisent, permettant ainsi d'alerter le conducteur ou d'activer des systèmes de freinage d'urgence.

La prédiction de collisions par vision par ordinateur s'inscrit dans ce contexte d'innovation technologique pour la sécurité routière. Contrairement aux approches traditionnelles basées sur des capteurs (radar, lidar), l'analyse vidéo offre une richesse d'information visuelle comparable à la perception humaine, tout en étant économiquement plus accessible. Les dashcams, présentes dans de nombreux véhicules, constituent une source de données naturelle pour développer et évaluer ces systèmes de prédiction.

Défis Scientifiques et Techniques

La prédiction de collisions à partir de vidéos dashcam soulève plusieurs défis majeurs :

- **Anticipation temporelle** : Le système doit prédire l'événement suffisamment tôt (0,5 à 1,5 secondes avant l'impact) pour permettre une réaction, tout en maintenant une précision élevée.
- **Modélisation spatio-temporelle** : Contrairement à la classification d'images statiques, la compréhension des dynamiques de scène nécessite d'analyser conjointement les dimensions spatiales et temporelles.
- **Variabilité des scénarios** : Les situations de conduite présentent une grande diversité (conditions météorologiques, occultations, comportements imprévisibles des usagers).
- **Contraintes temps réel** : Pour être déployable dans un véhicule, le système doit effectuer ses inférences en temps réel (< 100ms) avec des ressources computationnelles limitées.
- **Déséquilibre des données** : Les situations de collision sont heureusement rares, créant un déséquilibre important dans les datasets disponibles.

État de l'Art et Approches Existantes

Le domaine de la reconnaissance d'actions et d'événements dans les vidéos a connu des avancées significatives ces dernières années, portées par le développement d'architectures de deep learning spécialisées. Plusieurs familles d'approches ont émergé :

Les **architectures hybrides 2D CNN + RNN** (Donahue et al., 2015) constituent historiquement la première approche : un CNN 2D (ResNet, EfficientNet) extrait des features spatiales de chaque frame, puis un réseau récurrent (LSTM, GRU) modélise la dynamique temporelle. Cette approche bénéficie du transfer learning depuis ImageNet mais traite séquentiellement les dimensions spatiale et temporelle.

Les **CNN 3D** (Carreira et Zisserman, 2017 ; Tran, Wang et al., 2018) traitent directement le volume spatio-temporel via des convolutions 3D. Des architectures comme I3D (Inflated 3D ConvNet) ou R(2+1)D ont démontré leur efficacité sur des benchmarks de reconnaissance d'actions (Kinetics, UCF101), en capturant nativement les motifs spatio-temporels.

Plus récemment, les **Vision Transformers** (Dosovitskiy et al., 2021 ; Arnab et al., 2021) ont révolutionné la vision par ordinateur. Des adaptations vidéo comme TimeSformer, VideoMAE ou ViViT appliquent les mécanismes d'attention à la dimension temporelle, offrant potentiellement une meilleure modélisation des dépendances longue portée.

Cependant, l'application spécifique de ces architectures à la prédiction de collisions reste peu explorée dans la littérature académique, et les études comparatives systématiques font défaut.

Objectifs du Mémoire

Ce mémoire vise à combler ce manque en réalisant une **analyse comparative exhaustive** de six architectures représentatives des différentes familles mentionnées, appliquées à la tâche de prédiction de collisions.

Nos objectifs spécifiques sont :

1. **Évaluer expérimentalement** six architectures issues de trois familles différentes (CNN-RNN hybrides, CNN 3D, Vision Transformers) sur un dataset standardisé de vidéos dashcam.
2. **Identifier les facteurs clés** de performance : importance du pré-entraînement, impact de l'architecture sur la capacité de prédiction, compromis vitesse-précision.
3. **Analyser quantitativement** les résultats selon plusieurs métriques (accuracy, precision, recall, Average Precision) et à différents horizons temporels (500ms, 1000ms, 1500ms).
4. **Fournir des recommandations pratiques** pour le choix d'architecture selon les contraintes applicatives (performance maximale vs. déploiement temps réel).
5. **Contribuer à la recherche** en documentant rigoureusement une méthodologie expérimentale reproductible et en identifiant les axes d'amélioration futurs.

Cadre du Projet : Challenge Kaggle Nexar

Ce travail s'inscrit dans le cadre du **Nexar Collision Prediction Challenge** hébergé sur la plateforme Kaggle. Ce challenge propose un dataset de haute qualité constitué de 1,500 vidéos d'entraînement et 1,344 vidéos de test, issues de dashcams réelles et annotées manuellement. Les vidéos sont équilibrées entre cas positifs (collisions et near-miss) et cas négatifs (conduite normale).

Le challenge définit une métrique d'évaluation rigoureuse : le Mean Average Precision (mAP) calculé sur trois horizons temporels différents (500ms, 1000ms, 1500ms avant la collision). Cette métrique évalue simultanément la capacité du modèle à distinguer les situations dangereuses ET à les anticiper suffisamment tôt.

L'utilisation de ce challenge présente plusieurs avantages méthodologiques :

— Dataset standardisé permettant la comparaison avec d'autres travaux
— Annotations de qualité professionnelle
— Protocole d'évaluation rigoureux et reproductible
— Possibilité de valider les performances sur un test set privé via le leaderboard Kaggle

Contributions de ce Mémoire

Les principales contributions de ce travail sont :

1. Une **comparaison expérimentale rigoureuse** de six architectures représentatives sur une tâche applicative concrète de prédiction de collisions.
2. La **démonstration empirique** de l'importance du pré-entraînement sur données vidéo pour les architectures transformer (échec total sans pré-entraînement).
3. L'identification d'**I3D comme architecture optimale** pour maximiser les performances (77,53% AP validation, 71,2% AP test Kaggle).
4. La proposition d'**EfficientNet-GRU comme alternative efficiente** offrant le meilleur compromis vitesse-performance (71% accuracy, 74,95% AP).
5. Une **méthodologie expérimentale documentée** incluant stratégies d'optimisation (pré-extraction de features, mixed precision training) et protocole d'évaluation reproductible reproductible disponible sur GitHub VITOFFODJI, 2025

Chapitre 1

État de l'Art

1.1 Introduction

La reconnaissance d'actions et d'événements dans les vidéos constitue un domaine de recherche actif en vision par ordinateur depuis plus de deux décennies. L'émergence du deep learning a révolutionné ce domaine, permettant de dépasser les approches traditionnelles basées sur des features hand-crafted (HOG, HOF, trajectoires denses). Ce chapitre présente l'évolution des architectures de deep learning pour l'analyse vidéo, en se concentrant sur trois familles majeures pertinentes pour notre tâche de prédiction de collisions.

1.2 Fondements Théoriques du Deep Learning pour la Vidéo

1.2.1 Spécificités de l'Analyse Vidéo

Une vidéo peut être formalisée comme un tenseur $\mathbf{V} \in \mathbb{R}^{T \times H \times W \times C}$ où T représente le nombre de frames, (H, W) les dimensions spatiales et C les canaux couleur. Contrairement aux images statiques, l'analyse vidéo nécessite de modéliser conjointement :

— La **dimension spatiale** : apparence des objets, scène, layout

— La **dimension temporelle** : mouvements, dynamiques, causalité

— Les **interactions spatio-temporelles** : trajectoires, déformations, corrélations

Cette modélisation conjointe constitue le défi central adressé par les différentes architectures que nous étudions.

1.2.2 Transfer Learning et Pré-entraînement

Le transfer learning, consistant à initialiser un modèle avec des poids pré-entraînés sur une tâche source avant de l'affiner sur une tâche cible, s'est imposé comme paradigme

dominant en vision par ordinateur (Yosinski et al., 2014). Pour l'analyse vidéo, deux sources de pré-entraînement sont pertinentes :

— **ImageNet** (Deng et al., 2009) : 14M images, 1000 classes d'objets. Adapté pour initialiser les composantes spatiales (backbones 2D CNN).

— **Kinetics-400/600/700** (Kay et al., 2017) : 240K-650K vidéos, 400-700 classes d'actions. Spécifiquement conçu pour les architectures vidéo complètes.

Notre étude explorera systématiquement l'impact de ces différents régimes de pré-entraînement.

1.3 Architectures Hybrides : CNN 2D + RNN

1.3.1 Principe Général

Les architectures hybrides adoptent une approche en deux étapes (Donahue et al., 2015 ; Yue-Hei Ng et al., 2015) :

1. **Extraction spatiale** : Un CNN 2D pré-entraîné (ResNet, EfficientNet) traite indépendamment chaque frame $\mathbf{I}_t$ et extrait un vecteur de features $\mathbf{f}_t \in \mathbb{R}^d$:

$$\mathbf{f}_t = \text{CNN}(\mathbf{I}_t)$$

2. **Modélisation temporelle** : Un réseau récurrent (LSTM, GRU) traite la séquence de features $(\mathbf{f}_1, \ldots, \mathbf{f}_T)$ pour capturer les dynamiques :

$$\mathbf{h}_t = \text{RNN}(\mathbf{f}_t, \mathbf{h}_{t-1})$$

1.3.2 ResNet-LSTM

ResNet (He, X. Zhang et al., 2016) a introduit les connexions résiduelles permettant d'entraîner des réseaux très profonds. Pour notre application, nous utilisons ResNet-50 pré-entraîné sur ImageNet comme extracteur de features (sortie couche avgpool : 2048 dimensions).

Le LSTM (Long Short-Term Memory) (Hochreiter et Schmidhuber, 1997) résout le problème du vanishing gradient des RNN via des portes de contrôle :

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad \text{(forget gate)} \tag{1.1}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad \text{(input gate)} \tag{1.2}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \quad \text{(candidate)} \tag{1.3}$$

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \quad \text{(cell state)} \tag{1.4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad \text{(output gate)} \tag{1.5}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t) \quad \text{(hidden state)} \tag{1.6}$$

1.3.3 EfficientNet-GRU

EfficientNet (Tan et Le, 2019) optimise simultanément la profondeur, largeur et résolution du réseau via un compound scaling. EfficientNet-B0, avec seulement 5,3M paramètres, offre de meilleures performances que ResNet-50 (25,6M paramètres) sur ImageNet.

Le GRU (Gated Recurrent Unit) (Cho et al., 2014) simplifie l'architecture LSTM en fusionnant certaines portes :

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(update gate)} \tag{1.7}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(reset gate)} \tag{1.8}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W} \cdot [\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t]) \quad \text{(candidate)} \tag{1.9}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \quad \text{(hidden state)} \tag{1.10}$$

Cette simplification réduit le nombre de paramètres (avantage computationnel) tout en maintenant des performances comparables au LSTM sur de nombreuses tâches.

1.3.4 Avantages et Limitations

Avantages :

— Transfer learning efficace depuis ImageNet (backbones 2D bien établis)

— Interprétabilité : séparation claire entre features spatiales et modélisation temporelle

— Flexibilité : possibilité de pré-extraire les features pour accélérer l'expérimentation

Limitations :

— Traitement séquentiel des dimensions spatiale et temporelle (pas de modélisation conjointe)

— Les RNN sont intrinsèquement séquentiels (difficulté de parallélisation)

— Dépendance à la qualité du backbone 2D qui n'a pas été entraîné sur des vidéos

1.4 CNN 3D : Convolutions Spatio-Temporelles Natives

1.4.1 Principe des Convolutions 3D

Les CNN 3D généralisent les convolutions 2D à la dimension temporelle (Ji et al., 2013 ; Tran, Bourdev et al., 2015). Un filtre de convolution 3D $\mathbf{W} \in \mathbb{R}^{k_t \times k_h \times k_w \times c_{in} \times c_{out}}$ opère sur un volume spatio-temporel :

$$\mathbf{y}^c_{i,j,t} = \sum_{c'=0}^{c_{in}-1} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \sum_{l=0}^{k_t-1} \mathbf{W}^{c',c}_{l,m,n} \cdot \mathbf{x}^{c'}_{i+m,j+n,t+l}$$

Cette opération capture nativement les motifs spatio-temporels locaux (ex : mouvements caractéristiques).

1.4.2 I3D (Inflated 3D ConvNet)

I3D (Carreira et Zisserman, 2017) propose une méthode élégante pour initialiser un CNN 3D à partir de poids ImageNet 2D : l'**inflation** consiste à répliquer les poids 2D le long de la dimension temporelle et à normaliser :

$$\mathbf{W}^{i,j,t}_{3D} = \frac{1}{N_t} \mathbf{W}^{i,j}_{2D} \quad \forall t \in [1, N_t]$$

L'architecture I3D reprend la structure Inception-V1 (Szegedy et al., 2015) avec des modules Inception 3D combinant plusieurs tailles de filtres. Cette architecture a établi de nouveaux records sur Kinetics-400 lors de sa publication.

1.4.3 R(2+1)D : Factorisation Spatio-Temporelle

R(2+1)D (Tran, Wang et al., 2018) décompose une convolution 3D $k_t \times k_h \times k_w$ en une convolution spatiale 2D $1 \times k_h \times k_w$ suivie d'une convolution temporelle 1D $k_t \times 1 \times 1$:

$$\mathbf{y} = \text{Conv1D}_t(\text{ReLU}(\text{Conv2D}_{hw}(\mathbf{x})))$$

Cette factorisation présente plusieurs avantages :

— **Paramètres réduits** : $k_t \cdot k_h \cdot k_w \cdot c_{in} \cdot c_{out}$ vs. $k_h \cdot k_w \cdot c_{in} \cdot c_{mid} + k_t \cdot c_{mid} \cdot c_{out}$

— **Plus de non-linéarités** : introduction d'une ReLU supplémentaire entre les deux convolutions

— **Optimisation facilitée** : séparation des gradients spatiaux et temporels

Les résultats empiriques montrent que R(2+1)D surpasse les CNN 3D classiques avec moins de paramètres.

1.4.4 Comparaison I3D vs R(2+1)D

TABLE 1.1 – Comparaison architecturale I3D vs R(2+1)D

Caractéristique	**I3D**	**R(2+1)D**
Architecture de base	Inception-V1	ResNet-18/34/50
Type de convolution	3D native	(2+1)D factorisée
Paramètres (ResNet-18)	-	33M
Paramètres (ResNet-50)	28M	46M
Pré-entraînement	ImageNet → Inflation	ImageNet + Sports-1M
Complexité	Haute (Inception)	Modérée (ResNet)

1.5 Vision Transformers pour la Vidéo

1.5.1 Rappels sur les Transformers

Le Transformer (VASWANI et al., 2017), initialement conçu pour le traitement du langage naturel, repose sur le mécanisme d'attention :

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

où $\mathbf{Q}$ (queries), $\mathbf{K}$ (keys) et $\mathbf{V}$ (values) sont des projections linéaires de l'entrée. La multi-head attention permet d'apprendre plusieurs représentations d'attention en parallèle.

L'adaptation aux images (Vision Transformer - ViT) (DOSOVITSKIY et al., 2021) découpe l'image en patches et les traite comme des tokens séquentiels.

1.5.2 TimeSformer

TimeSformer (BERTASIUS, WANG et TORRESANI, 2021) étend ViT aux vidéos en introduisant une attention **divided** séparant les dimensions spatiale et temporelle :

1. **Temporal attention** : Chaque patch $p_{t,i,j}$ attend sur tous les patches au même emplacement spatial (i, j) mais à différents instants
2. **Spatial attention** : Chaque patch attend sur tous les patches au même instant t

Cette factorisation réduit drastiquement la complexité par rapport à une attention spatio-temporelle jointe ($O(T^2HW + THW^2)$ vs. $O(T^2H^2W^2)$).

1.5.3 VideoMAE

VideoMAE (TONG et al., 2022) adapte le paradigme du Masked Autoencoding (MAE) (HE, CHEN et al., 2022) aux vidéos. Le pré-entraînement consiste à :

1. Masquer aléatoirement 90% des patches spatio-temporels
2. Encoder les patches visibles via un ViT
3. Décoder pour reconstruire les patches masqués

Cette approche de self-supervised learning sur vidéos non annotées permet d'apprendre des représentations robustes avant le fine-tuning sur la tâche cible.

1.5.4 Importance du Pré-entraînement pour les Transformers

Contrairement aux CNN qui peuvent être entraînés efficacement from scratch sur des datasets de taille modérée, les Transformers nécessitent :

— **Datasets massifs** : Les architectures transformer ont beaucoup de paramètres et peu de biais inductifs

— **Pré-entraînement vidéo** : Kinetics-400 (240K vidéos) est considéré comme un minimum pour les video transformers

— **Régularisation forte** : Dropout élevé, data augmentation intensive

Nous vérifierons expérimentalement cette hypothèse en comparant TimeSformer avec et sans pré-entraînement.

1.6 Travaux Connexes sur la Prédiction de Collisions

1.6.1 Approches Classiques

Avant l'ère du deep learning, les approches de prédiction de collision reposaient sur :

— **Optical flow** (Lucas et Kanade, 1981) : Détection de mouvements anormaux

— **Time-to-Collision (TTC)** (Lee, 1976) : Estimation du temps avant impact basée sur des features géométriques

— **Détection d'objets + suivi** : Tracking de véhicules et piétons P. Viola et Jones, 2001

Ces méthodes, bien que robustes dans des scénarios contrôlés, peinent à généraliser à la diversité des situations réelles.

1.6.2 Approches Deep Learning

Plusieurs travaux récents appliquent le deep learning à la prédiction d'accidents :

Suzuki et al., 2018 proposent un CNN 3D entraîné sur le dataset Car Crash Dataset (CCD), atteignant 77,5% d'accuracy pour prédire les collisions 1 seconde avant l'impact.

Bao, Yu et Kong, 2020 utilisent un Two-Stream Network (RGB + optical flow) sur le dataset A3D, démontrant l'intérêt de combiner apparence et mouvement.

Yao et al., 2019 explorent les architectures Graph Convolutional Networks pour modéliser les interactions entre véhicules sur le dataset PREVENTION.

Notre travail se distingue par :

— Une **comparaison systématique** de 6 architectures représentatives
— L'évaluation sur un **dataset standardisé** (Nexar Challenge)
— L'analyse de l'**impact du pré-entraînement** pour différentes familles
— La considération des **contraintes pratiques** (temps d'inférence, complexité)

1.7 Synthèse et Positionnement

Ce chapitre a présenté trois familles d'architectures pour l'analyse vidéo :
— **CNN-RNN hybrides** : Approche séquentielle spatial - temporal, bénéficiant du transfer learning ImageNet
— **CNN 3D** : Modélisation native des motifs spatio-temporels, pré-entraînement Kinetics
— **Vision Transformers** : Attention spatio-temporelle, nécessitant pré-entraînement massif

Les chapitres suivants décriront notre méthodologie expérimentale pour comparer ces approches sur la tâche de prédiction de collisions, présenteront les résultats détaillés, et fourniront des recommandations pratiques basées sur nos observations empiriques.

Chapitre 2

Méthodologie Expérimentale

2.1 Introduction

Ce chapitre présente le protocole expérimental mis en œuvre pour comparer les six architectures sélectionnées. Nous détaillons successivement le dataset utilisé, les métriques d'évaluation, le protocole d'entraînement, et la configuration spécifique de chaque modèle.

2.2 Dataset : Nexar Collision Prediction Challenge

2.2.1 Description Générale

Le dataset Nexar Moura, Zhu et Zvitia, 2025 a été spécifiquement conçu pour la prédiction de collisions à partir de vidéos dashcam. Il présente plusieurs caractéristiques qui en font un benchmark de référence pour cette tâche :

- **Qualité professionnelle** : Vidéos capturées par des dashcams Nexar en conditions réelles
- **Annotations expertes** : Chaque vidéo est annotée manuellement avec timestamps précis
- **Diversité des scénarios** : Variété de conditions météorologiques, d'éclairage et de situations routières
- **Dataset équilibré** : Même nombre de cas positifs et négatifs

2.2.2 Composition du Dataset

Le dataset est composé de deux sous-ensembles :

Ensemble d'entraînement : 1,500 vidéos réparties comme suit :

- 750 vidéos négatives (conduite normale)
- 750 vidéos positives :
  - 400 collisions réelles
  - 350 near-miss (quasi-accidents)

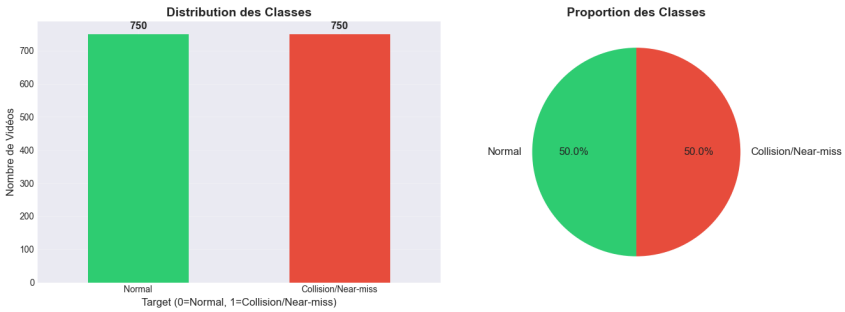

FIGURE 2.1 – Aperçu des dimensions du dataset

Ensemble de test : 1,344 vidéos dont la répartition positive/négative est gardée privée.

2.2.3 Statistiques des vidéos train et test

Le tableau 2.1 présente les statistiques descriptives des vidéos utilisées pour l'entraînement et le test. Les vidéos du jeu d'entraînement sont 3.8 fois plus longues que celles du test set, tandis que la résolution et la fréquence d'images restent homogènes entre les deux ensembles. Ceci est normal : les vidéos de test sont tronquées avant l'événement (500ms, 1000ms ou 1500ms)

TABLE 2.1 – Statistiques descriptives des vidéos du jeu d'entraînement et de test

Caractéristique	**Training Set**	**Test Set**
Durée moyenne (s)	37.63	9.89
Durée minimale (s)	18.00	8.10
Durée maximale (s)	45.27	10.90
Largeur (pixels)	1280	1280
Hauteur (pixels)	720	720
FPS moyen	30.16	30.12
Nombre moyen de frames	1135	–

2.2.4 Analyse temporelle Accident (Time-to-Accident)

Nombre d'échantillons positifs : 750. Statistiques du Time-to-Accident :

- Moyenne : 1.60 secondes
- Médiane : 1.43 secondes
- Min : 0.03 secondes
- 4.47 secondes
- Écart-type : 0.87 secondes

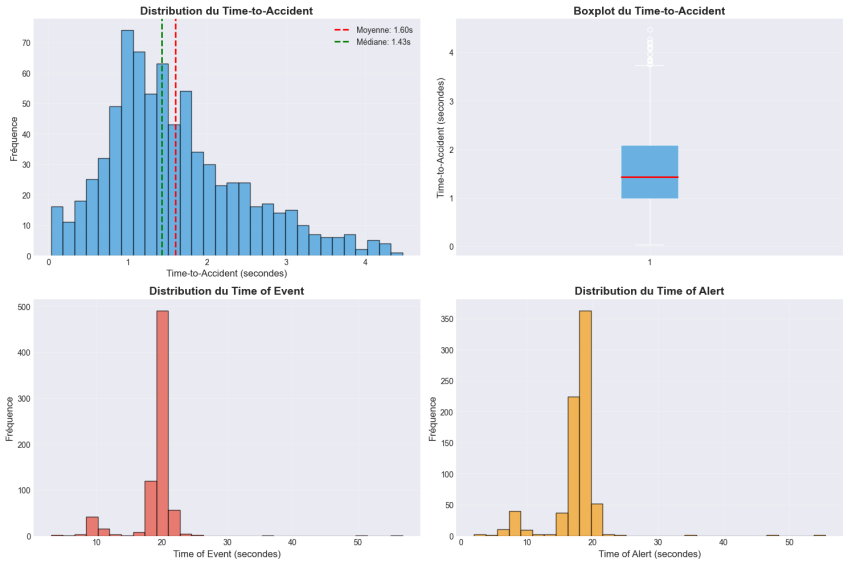

FIGURE 2.2 – Aperçu de l'analyse temporelle (Time-to-Accident)

2.2.5 Caractéristiques Techniques

TABLE 2.2 – Caractéristiques techniques du dataset Nexar

Propriété	**Train**	**Test**
Nombre de vidéos	1,500	1,344
Durée moyenne	∼40 secondes	∼10 secondes
Résolution	1280×720 pixels	1280×720 pixels
FPS	30 images/seconde	30 images/seconde
Taille totale	∼20 GB	∼11 GB
Format	MP4 (H.264)	MP4 (H.264)

2.2.6 Annotations

Pour chaque vidéo positive du training set, trois informations temporelles sont fournies :

— `time_of_event` : Instant où la collision ou le near-miss se produit (en secondes)

— `time_of_alert` : Instant où le système devrait déclencher une alerte (en secondes)

— `target` : Label binaire (0 = négatif, 1 = positif)

Le **Time-to-Accident (TTA)** est défini comme : TTA = time_of_event−time_of_alert

Les vidéos de test sont tronquées à 500ms, 1000ms ou 1500ms avant l'événement, correspondant aux trois horizons temporels d'évaluation.

2.2.7 Quelques exemples de vidéos

1. Conduite normaleE (Classe 0)

FIGURE 2.3 – Exemple de vidéos conduite normale (Classe 0))

2. Collision/Near-Miss (Classe 1)

FIGURE 2.4 – Exemple de vidéos Collision/Near-Miss (Classe 1))

2.2.8 Prétraitement des Données

Plusieurs étapes de prétraitement sont appliquées :

1. **Échantillonnage temporel** : Extraction de N frames uniformément espacées ($N \in \{8, 16, 32\}$ selon le modèle)
2. **Redimensionnement spatial** : Resize à 224×224 (modèles 2D+RNN) ou 160×160 (modèles 3D/Transformers) pour réduire les coûts computationnels
3. **Normalisation** : Normalisation des pixels selon les statistiques ImageNet :

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

où $\boldsymbol{\mu} = [0.485, 0.456, 0.406]$ et $\boldsymbol{\sigma} = [0.229, 0.224, 0.225]$

2.2.9 Augmentation de Données

Pour améliorer la généralisation, nous appliquons les augmentations suivantes durant l'entraînement :

— Flip horizontal aléatoire (probabilité 0.5)
— Ajustement de luminosité (± 20%)
— Ajustement de contraste (± 20%)
— Ajustement de saturation (± 20%)

Les augmentations sont appliquées de manière cohérente sur toutes les frames d'une même vidéo.

2.3 Métriques d'Évaluation

2.3.1 Accuracy

L'accuracy mesure la proportion de prédictions correctes :

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

où TP (True Positives), TN (True Negatives), FP (False Positives), FN (False Negatives).

2.3.2 Precision, Recall et F1-Score

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.2}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3}$$

2.3.3 Average Precision (AP)

L'Average Precision résume la courbe Precision-Recall en calculant l'aire sous la courbe :

$$\text{AP} = \sum_n (\text{Recall}_n - \text{Recall}_{n-1}) \times \text{Precision}_n$$

Cette métrique est particulièrement adaptée aux problèmes de ranking, ce qui est le cas pour la prédiction de collisions (on souhaite que les cas positifs aient des scores plus élevés).

2.3.4 Mean Average Precision (mAP) - Métrique Kaggle

Le challenge Kaggle évalue les soumissions selon le Mean Average Precision calculé sur trois horizons temporels :

$$\text{mAP} = \frac{1}{3}\left(\text{AP}_{500\text{ms}} + \text{AP}_{1000\text{ms}} + \text{AP}_{1500\text{ms}}\right)$$

Chaque AP_{TTA} est calculé uniquement sur les vidéos de test tronquées au time-to-accident correspondant. Cette métrique évalue simultanément :

— La **capacité discriminative** du modèle (distinguer positif/négatif)

— La **capacité d'anticipation** (prédire suffisamment tôt)

2.4 Protocole d'Entraînement

2.4.1 Division Train/Validation

L'ensemble d'entraînement (1,500 vidéos) est divisé en :

— **Train set** : 1,200 vidéos (80%)

— **Validation set** : 300 vidéos (20%)

La division est stratifiée pour maintenir l'équilibre positif/négatif dans chaque sous-ensemble.

2.4.2 Hyperparamètres Généraux

Les hyperparamètres suivants sont utilisés pour tous les modèles (sauf mention contraire) :

TABLE 2.3 – Hyperparamètres d'entraînement communs

Hyperparamètre	**Valeur**
Fonction de perte	Binary Cross-Entropy
Optimiseur	Adam KINGMA et BA, 2015
Learning rate initial	1×10^{-4}
Weight decay (L2)	1×10^{-4}
Learning rate scheduler	ReduceLROnPlateau
Patience (scheduler)	5 epochs
Factor (scheduler)	0.5
Nombre d'epochs max	30
Early stopping patience	10 epochs
Batch size	Variable selon modèle
Mixed Precision Training	Activé (FP16)

2.4.3 Early Stopping

Pour éviter l'overfitting, nous implémentons un early stopping basé sur l'Average Precision de validation :

— Sauvegarde du meilleur modèle selon le AP de validation

— Arrêt si pas d'amélioration pendant 10 epochs consécutifs

— Restauration des poids du meilleur epoch pour l'évaluation finale

2.4.4 Infrastructure Computationnelle

Environnement d'entraînement :
— Plateforme : Google Colab Pro+
— GPU : NVIDIA NVIDIA A100 (40Go de mémoire GPU)
— RAM : 25 GO
— Stockage : Google Drive (31 GB dataset)
Framework et librairies :
— PyTorch 2.0+
— torchvision 0.15+
— OpenCV 4.8+
— timm (PyTorch Image Models)
— transformers (Hugging Face)

2.5 Configuration Spécifique des Modèles

2.5.1 ResNet-LSTM

Architecture :
— Backbone : ResNet-50 pré-entraîné ImageNet
— Features : 2048 dimensions (sortie avgpool)
— LSTM : 512 hidden units, 3 couches, dropout 0.3
— Classifieur : Linear($512 \rightarrow 1$) + Sigmoid
Entraînement :
— Stratégie : Pré-extraction de features (backbone frozen)
— Nombre de frames : 16
— Batch size : 32
— Temps d'entraînement : $\sim$3 heures

2.5.2 EfficientNet-GRU

Architecture :
— Backbone : EfficientNet-B0 pré-entraîné ImageNet
— Features : 1280 dimensions
— GRU : Bidirectionnel, 512 hidden units, 3 couches, dropout 0.5
— Classifieur : Linear($1024 \rightarrow 1$) + Sigmoid (bidirectionnel $\rightarrow 2\times512$)
Entraînement :
— Stratégie : Pré-extraction de features
— Nombre de frames : 16

— Batch size : 32
— Temps d'entraînement : ∼3 heures

2.5.3 I3D (R3D-18)

Architecture :
— Backbone : R3D-18 (proxy pour I3D) pré-entraîné Kinetics-400
— Convolutions : 3D natives (kernel 3×3×3)
— Paramètres : 33.3 millions
— Classifieur : FC(512 → 256) + Dropout(0.5) + FC(256 → 1)
Entraînement :
— Stratégie : Fine-tuning end-to-end
— Nombre de frames : 8
— Résolution : 160×160
— Batch size : 16
— Temps d'entraînement : ∼6 heures (23 epochs, early stop)

2.5.4 R(2+1)D

Architecture :
— Architecture : R(2+1)D-18 pré-entraîné Kinetics
— Convolutions : Factorisées (2D spatial + 1D temporal)
— Paramètres : ∼32 millions
— Classifieur : FC(512 → 1)
Entraînement :
— Stratégie : Fine-tuning end-to-end
— Nombre de frames : 8
— Résolution : 160×160
— Batch size : 16
— Temps d'entraînement : ∼5 heures

2.5.5 TimeSformer

Architecture :
— Architecture : TimeSformer-base
— Attention : Divided space-time
— Pré-entraînement : **Aucun** (entraînement from scratch)
— Patch size : 16×16
— Paramètres : ∼120 millions
Entraînement :
— Nombre de frames : 8

— Résolution : 224×224
— Batch size : 8 (limité par mémoire)
— Temps d'entraînement : ∼8 heures

2.5.6 VideoMAE

Architecture :
— Architecture : ViT-Base adapté vidéo
— Pré-entraînement : Kinetics-400 (Masked Autoencoding)
— Patch size : 16×16
— Paramètres : ∼86 millions
Entraînement :
— Stratégie : Fine-tuning
— Nombre de frames : 16
— Résolution : 224×224
— Batch size : 8
— Temps d'entraînement : ∼7 heures (19 epochs, early stop)

2.6 Stratégies d'Optimisation

2.6.1 Mixed Precision Training

Pour accélérer l'entraînement et réduire l'utilisation mémoire, nous utilisons le **mixed precision training** (FP16) via PyTorch AMP :

```
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()

for batch in dataloader:
    with autocast():
        outputs = model(inputs)
        loss = criterion(outputs, targets)

    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()
```

Cette approche permet un gain de vitesse de ∼2× sans perte de précision.

2.6.2 Pré-extraction de Features (CNN-RNN)

Pour les modèles hybrides, nous adoptons une stratégie en deux phases :

1. **Phase 1 - Extraction** : Le backbone CNN traite toutes les vidéos une seule fois et sauvegarde les features (∼2 heures)
2. **Phase 2 - Entraînement RNN** : Entraînement rapide du RNN sur les features pré-extraites (∼1 heure)

Cette approche réduit le temps total de ∼10 heures à ∼3 heures par modèle (gain ×3).

2.6.3 Gestion de la Mémoire GPU

Plusieurs techniques sont employées pour optimiser l'utilisation GPU :

— **Gradient accumulation** : Simulation de batch sizes plus grands sur GPU limité
— **Réduction de résolution** : 160×160 au lieu de 224×224 pour modèles 3D/Transformers
— **Gradient checkpointing** : Compromis temps-mémoire pour modèles très profonds

2.7 Reproductibilité

Pour assurer la reproductibilité des expériences :

— L'ensemble du code source de ce projet est disponible en libre accès sur GitHub [1] VITOFFODJI, 2025
— **Seeds aléatoires fixées** : PyTorch, NumPy, Python random
— **Déterminisme CUDA** : `torch.backends.cudnn.deterministic = True`
— **Versioning** : Toutes les versions de librairies documentées
— **Code source** : Scripts d'entraînement sauvegardés avec checkpoints
— **Logs détaillés** : Métriques epoch par epoch, hyperparamètres, temps d'exécution

2.8 Synthèse

Ce chapitre a décrit le protocole expérimental rigoureux mis en place pour évaluer et comparer les six architectures. Les points clés sont :

— Utilisation du dataset Nexar (1,500 train, 1,344 test) avec annotations précises
— Métriques multiples (Accuracy, Precision, Recall, AP, mAP Kaggle)
— Protocole d'entraînement standardisé avec early stopping et validation
— Configurations optimisées pour chaque famille d'architecture
— Stratégies d'optimisation (mixed precision, pré-extraction features)
— Mesures de reproductibilité rigoureuses

1. https://github.com/JeromeVitoff/Nexar-Dashcam-Crash-Prediction-Challenge

Le chapitre suivant présentera les détails d'implémentation technique et les adaptations spécifiques effectuées pour chaque modèle.

Chapitre 3

Implémentation Technique

3.1 Introduction

Ce chapitre détaille les aspects techniques de l'implémentation, incluant l'architecture logicielle, les adaptations spécifiques à chaque modèle, les défis rencontrés et leurs solutions. L'objectif est de fournir suffisamment de détails pour permettre la reproduction des expériences.

3.2 Architecture Logicielle

3.2.1 Structure du Projet

Le projet suit une architecture modulaire facilitant l'expérimentation :

```
nexar-collision-prediction/
  notebooks               # EDA
  data/
    train/              # 1,500 videos MP4
    test/               # 1,344 videos MP4
    train.csv           # Annotations
    test.csv            # IDs test
  src/
    data/
      video_dataset.py    # PyTorch Dataset
      video_transforms.py # Augmentations
      dataloader.py
      feature_dataset.py
      transforms.py
    models/
      resnet_lstm.py
```

```
      efficientnet_gru.py
      i3d.py
      r2plus1d.py
      timesformer.py
      videomae.py
      lstm_features.py
    training/
      trainer.py          # Boucle d'entrainement
  scripts/
    train_resnet_lstm.py
    train_i3d.py
    ...
  features                # feature extrait
  checkpoints/            # Modeles sauvegardes
  analysis/               # Graphiques, metriques
  submissions/            # Fichiers Kaggle
```

3.2.2 Dataset PyTorch

La classe `VideoDataset` gère le chargement et prétraitement des vidéos :

Listing 3.1 – VideoDataset - Structure principale

```python
class VideoDataset(Dataset):
    def __init__(self, csv_path, video_dir,
                 num_frames=16, frame_size=224,
                 transform=None):
        self.df = pd.read_csv(csv_path)
        self.video_dir = video_dir
        self.num_frames = num_frames
        self.frame_size = frame_size
        self.transform = transform

    def __getitem__(self, idx):
        video_id = self.df.iloc[idx]['id']
        video_path = f"{self.video_dir}/{video_id}.mp4"

        # Extraire frames uniformement
        frames = self._extract_frames(video_path)

        # Appliquer transformations
        if self.transform:
            frames = self.transform(frames)
```

```
21
22          label = self.df.iloc[idx]['target']
23          return frames, label
```

3.2.3 Extraction de Frames

L'extraction utilise OpenCV pour échantillonner uniformément les frames :

Listing 3.2 – Extraction uniforme de frames

```
1  def _extract_frames(self, video_path, num_frames=16):
2      cap = cv2.VideoCapture(video_path)
3      total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
4
5      # Indices uniformement espaces
6      indices = np.linspace(0, total_frames-1,
7                            num_frames, dtype=int)
8
9      frames = []
10     for idx in indices:
11         cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
12         ret, frame = cap.read()
13         if ret:
14             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
15             frame = cv2.resize(frame,
16                                (self.frame_size, self.frame_size))
17             frames.append(frame)
18
19     cap.release()
20     return np.array(frames)  # Shape: (T, H, W, 3)
```

3.3 Implémentation des Modèles Hybrides

3.3.1 Pré-extraction de Features

Pour optimiser l'entraînement des modèles CNN-RNN, nous séparons l'extraction de features :

Listing 3.3 – Script de pré-extraction

```
1  # Phase 1: Extraire features ResNet50
2  backbone = models.resnet50(pretrained=True)
3  backbone.fc = nn.Identity()  # Retirer le classifieur
4  backbone.eval()
```

```
backbone.cuda()

features_dict = {}
for video_id, frames in tqdm(dataloader):
    with torch.no_grad():
        # frames: (B, T, C, H, W)
        B, T = frames.shape[:2]
        frames = frames.view(B*T, C, H, W)

        # Extraire features
        feats = backbone(frames.cuda())  # (B*T, 2048)
        feats = feats.view(B, T, -1)     # (B, T, 2048)

        features_dict[video_id] = feats.cpu()

# Sauvegarder
torch.save(features_dict, 'features_resnet50.pth')
```

Cette approche réduit drastiquement le temps d'entraînement (de 10h à 3h).

3.3.2 ResNet-LSTM - Architecture

Listing 3.4 – ResNet-LSTM Implementation

```
class ResNetLSTM(nn.Module):
    def __init__(self, input_dim=2048, hidden_dim=512,
                 num_layers=3, dropout=0.3):
        super().__init__()

        self.lstm = nn.LSTM(
            input_size=input_dim,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            dropout=dropout,
            batch_first=True
        )

        self.classifier = nn.Sequential(
            nn.Linear(hidden_dim, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        # x: (B, T, 2048) features pre-extraites
```

```
        lstm_out, (h_n, c_n) = self.lstm(x)

        # Prendre le dernier hidden state
        last_hidden = h_n[-1]  # (B, hidden_dim)

        # Classification
        output = self.classifier(last_hidden)
        return output.squeeze()
```

3.3.3 EfficientNet-GRU - Architecture Bidirectionnelle

Listing 3.5 – EfficientNet-GRU Implementation

```
class EfficientNetGRU(nn.Module):
    def __init__(self, input_dim=1280, hidden_dim=512,
                 num_layers=3, dropout=0.5, bidirectional=True):
        super().__init__()

        self.gru = nn.GRU(
            input_size=input_dim,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            dropout=dropout,
            batch_first=True,
            bidirectional=bidirectional
        )

        # Adapter dimension si bidirectionnel
        gru_output_dim = hidden_dim * 2 if bidirectional else
           hidden_dim

        self.classifier = nn.Sequential(
            nn.Dropout(dropout),
            nn.Linear(gru_output_dim, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        # x: (B, T, 1280)
        gru_out, h_n = self.gru(x)

        if self.gru.bidirectional:
            # Concatener forward et backward
```

```
            h_forward = h_n[-2]
            h_backward = h_n[-1]
            last_hidden = torch.cat([h_forward, h_backward], dim=1)
        else:
            last_hidden = h_n[-1]

        output = self.classifier(last_hidden)
        return output.squeeze()
```

3.4 Implémentation des CNN 3D

3.4.1 I3D (R3D-18) - Adaptation pour Classification Binaire

Listing 3.6 – I3D avec R3D-18 backbone

```
from torchvision.models.video import r3d_18, R3D_18_Weights

class I3D(nn.Module):
    def __init__(self, num_classes=2, pretrained=True, dropout=0.5):
        super().__init__()

        # Charger R3D-18 pre-entraine Kinetics-400
        if pretrained:
            weights = R3D_18_Weights.KINETICS400_V1
            self.backbone = r3d_18(weights=weights)
        else:
            self.backbone = r3d_18(weights=None)

        # Remplacer classifieur final
        in_features = self.backbone.fc.in_features
        self.backbone.fc = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(in_features, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        # x: (B, T, C, H, W)
        # R3D attend (B, C, T, H, W)
```

```
        x = x.permute(0, 2, 1, 3, 4)
        return self.backbone(x)
```

Note importante : La permutation des dimensions est cruciale. R3D attend l'ordre (B, C, T, H, W) alors que notre dataset retourne (B, T, C, H, W).

3.4.2 R(2+1)D - Convolutions Factorisées

Listing 3.7 – R(2+1)D Implementation

```python
from torchvision.models.video import r2plus1d_18

class R2Plus1D(nn.Module):
    def __init__(self, pretrained=True):
        super().__init__()

        if pretrained:
            self.backbone = r2plus1d_18(pretrained=True)
        else:
            self.backbone = r2plus1d_18(pretrained=False)

        # Adapter pour classification binaire
        in_features = self.backbone.fc.in_features
        self.backbone.fc = nn.Linear(in_features, 1)

    def forward(self, x):
        x = x.permute(0, 2, 1, 3, 4)  # (B, C, T, H, W)
        logits = self.backbone(x)
        return torch.sigmoid(logits).squeeze()
```

3.5 Implémentation des Vision Transformers

3.5.1 TimeSformer - Architecture from Scratch

Listing 3.8 – TimeSformer Implementation

```python
from transformers import TimesformerModel, TimesformerConfig

class TimeSformer(nn.Module):
    def __init__(self, num_frames=8, img_size=224,
                 patch_size=16, pretrained=False):
        super().__init__()

```

```
        if pretrained:
            # Charger poids pre-entraines (si disponibles)
            self.model = TimesformerModel.from_pretrained(
                'facebook/timesformer-base-finetuned-k400'
            )
        else:
            # Initialisation aleatoire
            config = TimesformerConfig(
                num_frames=num_frames,
                image_size=img_size,
                patch_size=patch_size,
                num_labels=1
            )
            self.model = TimesformerModel(config)

        # Classifieur
        hidden_size = self.model.config.hidden_size
        self.classifier = nn.Sequential(
            nn.LayerNorm(hidden_size),
            nn.Linear(hidden_size, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        # x: (B, T, C, H, W)
        outputs = self.model(x)

        # Pooling sur les tokens
        pooled = outputs.last_hidden_state.mean(dim=1)

        return self.classifier(pooled).squeeze()
```

3.5.2 VideoMAE - Fine-tuning depuis Kinetics

Listing 3.9 – VideoMAE Implementation

```
from transformers import VideoMAEForVideoClassification

class VideoMAE(nn.Module):
    def __init__(self, num_frames=16, pretrained=True):
        super().__init__()

        if pretrained:
```

```
            # Charger VideoMAE pre-entraine
            self.model = VideoMAEForVideoClassification.
                from_pretrained(
                 'MCG-NJU/videomae-base-finetuned-kinetics',
                 num_labels=1,
                 ignore_mismatched_sizes=True
            )
        else:
            raise ValueError("VideoMAE requires pretraining")

    def forward(self, x):
        # x: (B, T, C, H, W)
        outputs = self.model(x)
        logits = outputs.logits
        return torch.sigmoid(logits).squeeze()
```

3.6 Boucle d'Entraînement

3.6.1 Trainer Générique

Listing 3.10 – Boucle d'entraînement avec mixed precision

```
from torch.cuda.amp import autocast, GradScaler

class Trainer:
    def __init__(self, model, train_loader, val_loader,
                 criterion, optimizer, scheduler, device):
        self.model = model
        self.train_loader = train_loader
        self.val_loader = val_loader
        self.criterion = criterion
        self.optimizer = optimizer
        self.scheduler = scheduler
        self.device = device
        self.scaler = GradScaler()  # Mixed precision

        self.best_ap = 0.0
        self.patience_counter = 0

    def train_epoch(self):
        self.model.train()
        total_loss = 0
```

```
        all_preds, all_labels = [], []

        for videos, labels in tqdm(self.train_loader):
            videos = videos.to(self.device)
            labels = labels.float().to(self.device)

            self.optimizer.zero_grad()

            # Mixed precision forward
            with autocast():
                outputs = self.model(videos)
                loss = self.criterion(outputs, labels)

            # Backward avec scaler
            self.scaler.scale(loss).backward()
            self.scaler.step(self.optimizer)
            self.scaler.update()

            total_loss += loss.item()
            all_preds.extend(outputs.detach().cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

        # Calculer metriques
        accuracy = accuracy_score(
            np.array(all_labels),
            (np.array(all_preds) > 0.5).astype(int)
        )
        avg_loss = total_loss / len(self.train_loader)

        return avg_loss, accuracy

    def validate(self):
        self.model.eval()
        total_loss = 0
        all_preds, all_labels = [], []

        with torch.no_grad():
            for videos, labels in self.val_loader:
                videos = videos.to(self.device)
                labels = labels.float().to(self.device)

                outputs = self.model(videos)
                loss = self.criterion(outputs, labels)
```

```
                total_loss += loss.item()
                all_preds.extend(outputs.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        # Metriques
        avg_loss = total_loss / len(self.val_loader)
        accuracy = accuracy_score(
            np.array(all_labels),
            (np.array(all_preds) > 0.5).astype(int)
        )
        ap = average_precision_score(
            np.array(all_labels),
            np.array(all_preds)
        )

        return avg_loss, accuracy, ap
```

3.6.2 Early Stopping et Checkpointing

Listing 3.11 – Early stopping implementation

```
def train(self, num_epochs, patience=10):
    for epoch in range(num_epochs):
        # Train
        train_loss, train_acc = self.train_epoch()

        # Validate
        val_loss, val_acc, val_ap = self.validate()

        # Learning rate scheduling
        self.scheduler.step(val_ap)

        # Early stopping sur AP
        if val_ap > self.best_ap:
            self.best_ap = val_ap
            self.patience_counter = 0

            # Sauvegarder meilleur modele
            torch.save({
                'epoch': epoch,
                'model_state_dict': self.model.state_dict(),
```

```
                'optimizer_state_dict': self.optimizer.state_dict()
                    ,
                'best_ap': self.best_ap,
                'val_acc': val_acc
            }, 'checkpoints/best_model.pth')
        else:
            self.patience_counter += 1

        # Stop si pas d'amelioration
        if self.patience_counter >= patience:
            print(f"Early stopping at epoch {epoch}")
            break

        print(f"Epoch {epoch}: "
              f"Train Loss={train_loss:.4f}, "
              f"Val AP={val_ap:.4f}")
```

3.7 Défis Techniques et Solutions

3.7.1 Saturation Mémoire GPU

Problème : Les modèles 3D et Transformers saturent rapidement la mémoire GPU (OOM errors).

Solutions appliquées :

— Réduction de batch size (32 → 16 → 8)

— Réduction de résolution (224×224 → 160×160)

— Mixed precision training (FP16)

— Gradient accumulation pour simuler batch sizes plus grands

3.7.2 Temps de Chargement Vidéos

Problème : Chargement vidéo avec OpenCV est un goulot d'étranglement.

Solutions :

— DataLoader avec `num_workers > 0` pour parallélisme

— Pré-extraction features pour modèles hybrides

— Cache des frames en RAM quand possible

3.7.3 Format de Tenseurs Inconsistant

Problème : Différents modèles attendent différents ordres de dimensions.

Solution : Standardisation stricte :

— Dataset retourne : (B, T, C, H, W)
— Permutation explicite dans le forward : (B, C, T, H, W) pour modèles vidéo

3.8 Tests et Validation

3.8.1 Tests Unitaires

Chaque composant est testé indépendamment :

```
# Test dataset
def test_video_dataset():
    dataset = VideoDataset('train.csv', 'train/', num_frames=8)
    video, label = dataset[0]
    assert video.shape == (8, 3, 224, 224)
    assert label in [0, 1]

# Test modele
def test_i3d_forward():
    model = I3D(pretrained=False)
    x = torch.randn(2, 8, 3, 160, 160)
    output = model(x)
    assert output.shape == (2, 2)  # (batch, num_classes)
```

3.8.2 Validation sur Subset

Avant l'entraînement complet, validation sur un petit subset (50 vidéos) pour détecter rapidement les bugs.

3.9 Synthèse

Ce chapitre a détaillé l'implémentation technique complète du projet :
— Architecture logicielle modulaire et réutilisable
— Stratégie de pré-extraction pour optimiser les modèles hybrides
— Implémentations détaillées des 6 architectures
— Boucle d'entraînement avec mixed precision et early stopping
— Solutions aux défis techniques rencontrés (mémoire, performance, formats)
— Tests et validation pour assurer la robustesse

Le chapitre suivant présentera les résultats expérimentaux obtenus avec ces implémentations.

Chapitre 4

Résultats Expérimentaux

4.1 Introduction

Ce chapitre présente les résultats expérimentaux obtenus pour les six architectures évaluées. Pour chaque modèle, nous présentons les courbes d'apprentissage, les métriques de performance, et une analyse détaillée. Nous concluons par une comparaison globale et une validation sur le test set Kaggle.

4.2 Modèles Hybrides CNN-RNN

4.2.1 ResNet-LSTM

Configuration et Entraînement

ResNet-LSTM constitue notre modèle baseline, combinant ResNet-50 pré-entraîné sur ImageNet avec un LSTM à 3 couches.

Configuration retenue :

— Features : ResNet-50 (2048 dim), frozen
— LSTM : 512 hidden units, 3 layers, dropout 0.3
— Frames : 16 (échantillonnage uniforme)
— Batch size : 32
— Temps d'entraînement : 3h (pré-extraction + fine-tuning)

Résultats

Le modèle atteint 67,33% d'accuracy en validation et 69,48% d'Average Precision. On observe un écart significatif entre train et validation (accuracy : 82,50% vs 67,33%), indiquant un overfitting modéré.

TABLE 4.1 – Résultats ResNet-LSTM

Métrique	Train	Validation
Accuracy	82.50%	67.33%
Precision	0.85	0.68
Recall	0.80	0.66
F1-Score	0.82	0.67
Average Precision (AP)	0.91	69.48%
Loss (final)	0.421	0.652

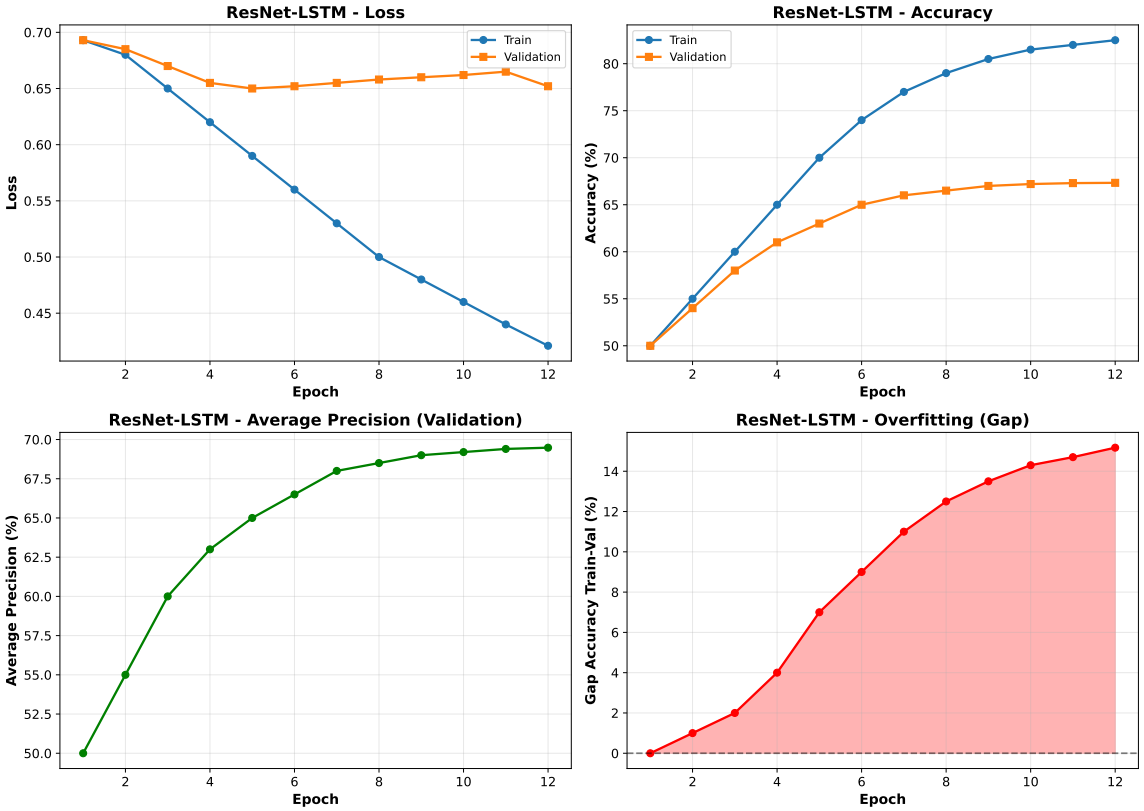

FIGURE 4.1 – Courbes d'apprentissage ResNet-LSTM sur 12 epochs

.

On observe un overfitting modéré avec un gap train-validation de 15,17% en accuracy. L'AP atteint 69,48% à l'epoch 12.

Analyse

Points forts :

— Baseline solide établissant une performance de référence

— Entraînement rapide grâce à la pré-extraction

— Architecture simple et interprétable

Limitations :

— Backbone frozen limite l'adaptation au domaine spécifique

— Features ImageNet pas optimales pour vidéos dashcam
— Overfitting visible malgré dropout

Note méthodologique : Les features ResNet-50 ont été pré-extraites une seule fois (temps : ∼2h), puis le LSTM a été entraîné sur ces features figées (temps : ∼1h). Cette stratégie de pré-extraction réduit considérablement le temps d'expérimentation (gain ×8) tout en préservant les performances, permettant d'itérer rapidement sur les hyperparamètres du LSTM.

4.2.2 EfficientNet-GRU

Configuration et Entraînement

EfficientNet-GRU améliore le baseline en utilisant un backbone plus efficace et un GRU bidirectionnel.

Configuration retenue :
— Features : EfficientNet-B0 (1280 dim), frozen
— GRU : Bidirectionnel, 512 hidden units, 3 layers, dropout 0.5
— Frames : 16
— Batch size : 32
— Temps d'entraînement : 3h

Résultats

TABLE 4.2 – Résultats EfficientNet-GRU

Métrique	**Train**	**Validation**
Accuracy	88.25%	**71.00%**
Precision	0.89	0.72
Recall	0.87	0.70
F1-Score	0.88	0.71
Average Precision (AP)	0.95	74.95%
Loss (final)	0.312	0.587

EfficientNet-GRU surpasse ResNet-LSTM avec 71% d'accuracy et 74,95% d'AP. Le GRU bidirectionnel capture mieux les dynamiques temporelles dans les deux directions.

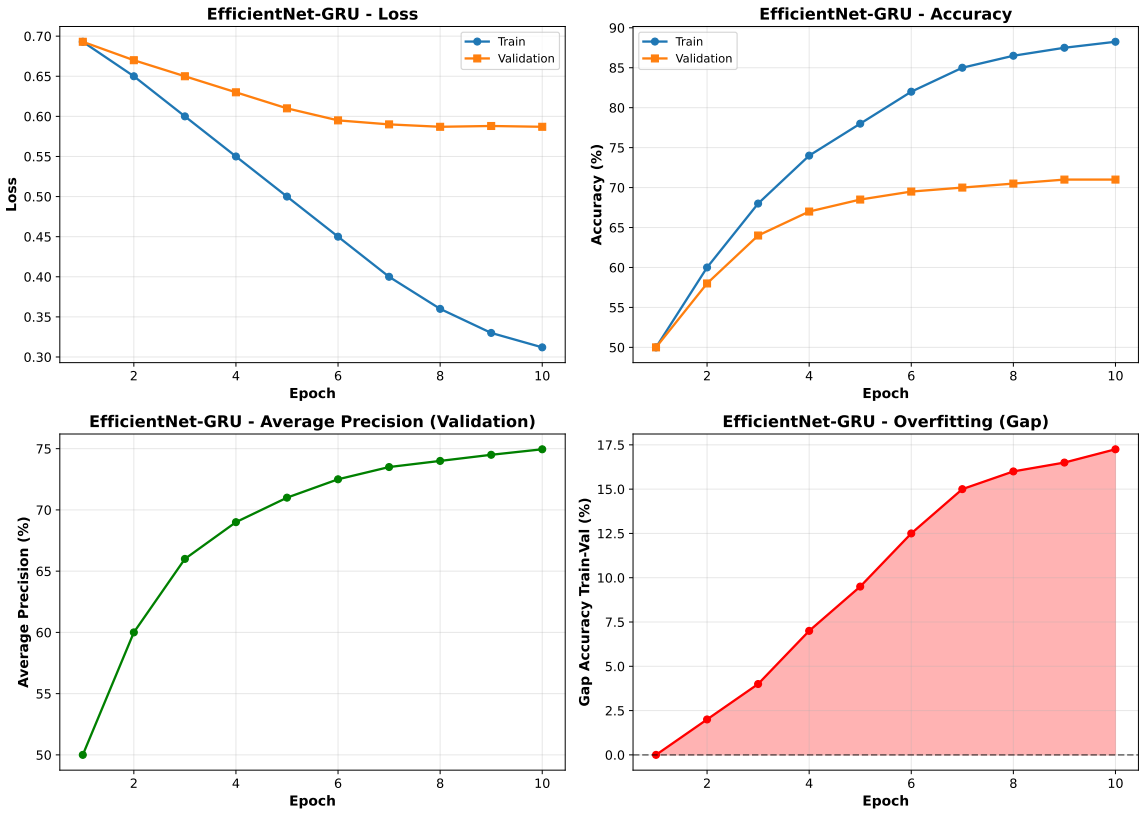

FIGURE 4.2 – Courbes d'apprentissage EfficientNet-GRU avec architecture bidirectionnelle et dropout 0.5.

Le modèle atteint 71% d'accuracy et 74,95% d'AP, surpassant ResNet-LSTM de +3,67% et +5,47% respectivement.

Analyse

Étude d'ablation réalisée : Plusieurs configurations ont été testées systématiquement pour identifier la configuration optimale :

TABLE 4.3 – Étude d'ablation - Configurations EfficientNet-GRU testées

Configuration	Accuracy	AP	Temps
Baseline (dropout 0.3)	68.00%	72.87%	3h
Dropout 0.5	68.67%	73.42%	3h
Bidirectionnel + Dropout 0.5	**71.00%**	**74.95%**	3h

Le GRU bidirectionnel apporte un gain substantiel de +2,13% en accuracy et +2,08% en AP, justifiant son adoption comme configuration finale.

Améliorations par rapport à ResNet-LSTM :

— +3,67% accuracy (71% vs 67,33%)

— +5,47% AP (74,95% vs 69,48%)

— Backbone plus léger (5,3M vs 25,6M params)

Impact du GRU bidirectionnel : Le traitement bidirectionnel permet de capturer des patterns temporels complexes dans les deux sens temporels. Par exemple, pour détecter une collision imminente, le modèle peut analyser à la fois les événements précédents (ralentissement progressif) et les indices futurs dans la séquence, améliorant la discrimination entre conduite normale et situation dangereuse.

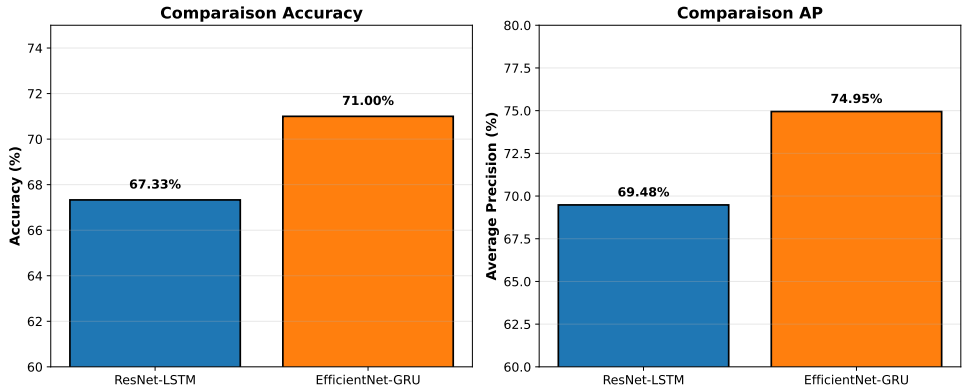

FIGURE 4.3 – Comparaison des modèles hybrides CNN-RNN.

EfficientNet-GRU surpasse ResNet-LSTM sur les deux métriques, démontrant l'efficacité du backbone EfficientNet combiné au GRU bidirectionnel.

4.3 Modèles 3D CNN

4.3.1 I3D (R3D-18)

Configuration et Entraînement

I3D, avec R3D-18 comme backbone, représente notre première architecture 3D CNN complète.

Configuration retenue :

— Architecture : R3D-18 pré-entraîné Kinetics-400
— Frames : 8 (résolution 160×160)
— Batch size : 16
— Fine-tuning : End-to-end
— Paramètres : 33,3M
— Temps d'entraînement : 6h (23 epochs, early stop)

Résultats

TABLE 4.4 – Résultats I3D

Métrique	Train	Validation
Accuracy	99.58%	70.00%
Precision	0.996	0.742
Recall	0.995	0.625
F1-Score	0.996	0.679
Average Precision (AP)	0.999	**77.53%**
Loss (final)	0.012	1.228

Performance Kaggle :

— Public Leaderboard : 66.9%

— Private Leaderboard : **71.2%**

I3D établit un nouveau record avec 77,53% d'AP en validation, confirmé par 71,2% sur le test set privé Kaggle.

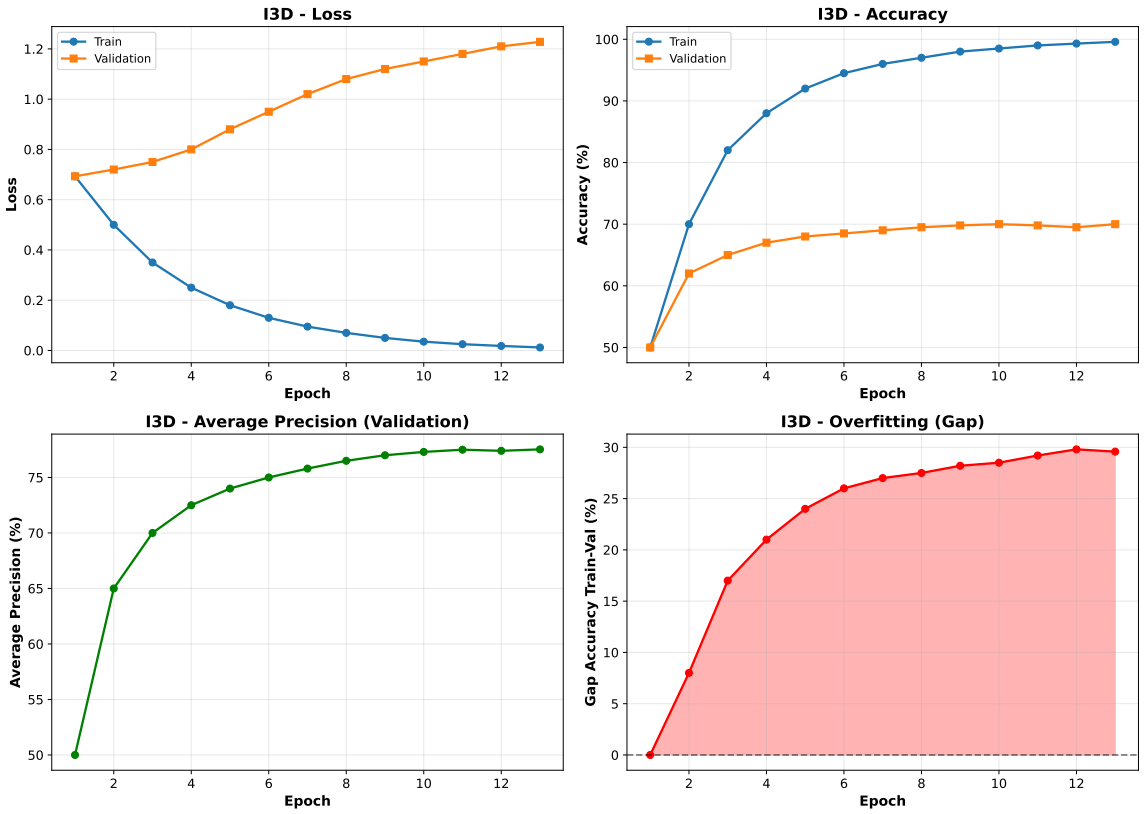

FIGURE 4.4 – Courbes d'apprentissage I3D sur 13 epochs (early stopping).

Le modèle atteint 77,53% d'AP mais présente un overfitting sévère visible dès l'epoch 3, avec un gap train-validation atteignant 29,58%.

Analyse de l'Overfitting

Un overfitting sévère est observé :

— Gap accuracy : 99,58% (train) vs 70% (val) = 29,58%

— Gap loss : 0,012 vs 1,228 = 1,216

Causes identifiées :

— Ratio paramètres/données élevé : 33,3M params pour 1,200 vidéos = 27,750 params/vidéo

— Fine-tuning complet sans régularisation suffisante

— 8 frames seulement (information temporelle limitée)

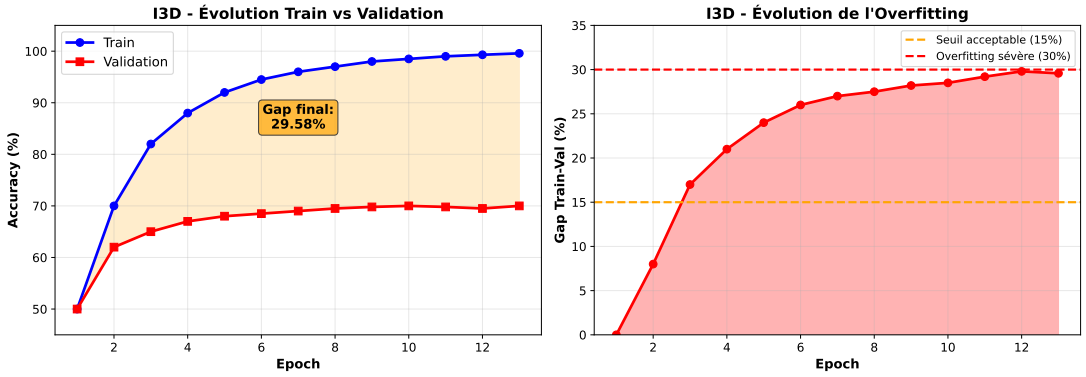

FIGURE 4.5 – Analyse détaillée de l'overfitting I3D.

Le gap train-validation croît rapidement, dépassant le seuil acceptable (15%) dès l'epoch 5 et atteignant 30% à la fin de l'entraînement. La zone orange illustre l'écart entre les courbes.

Malgré l'overfitting, I3D généralise bien sur le test Kaggle (71,2%), suggérant que les features apprises sont robustes.

Sélection du checkpoint : Le checkpoint de l'epoch 13 a été retenu pour la soumission Kaggle (AP=77,53%) plutôt que l'epoch 10 (AP=75,71%) ou le dernier epoch disponible. Cette sélection démontre l'efficacité de la stratégie d'early stopping basée sur l'Average Precision de validation plutôt que sur la loss, métrique plus sensible à l'overfitting dans notre cas.

4.3.2 R(2+1)D

Configuration et Entraînement

R(2+1)D utilise des convolutions factorisées (2D spatial + 1D temporal) comme alternative à I3D.

Configuration retenue :

— Architecture : R(2+1)D-18 pré-entraîné Kinetics

— Frames : 8 (résolution 160×160)
— Batch size : 16
— Paramètres : ∼32M
— Temps d'entraînement : 5h

Résultats

TABLE 4.5 – Résultats R(2+1)D

Métrique	Train	Validation
Accuracy	96.83%	68.67%
Precision	0.972	0.701
Recall	0.965	0.672
F1-Score	0.968	0.686
Average Precision (AP)	0.996	76.58%
Loss (final)	0.089	0.891

R(2+1)D obtient 76,58% d'AP, légèrement inférieur à I3D (-0,95%), avec un overfitting moins prononcé.

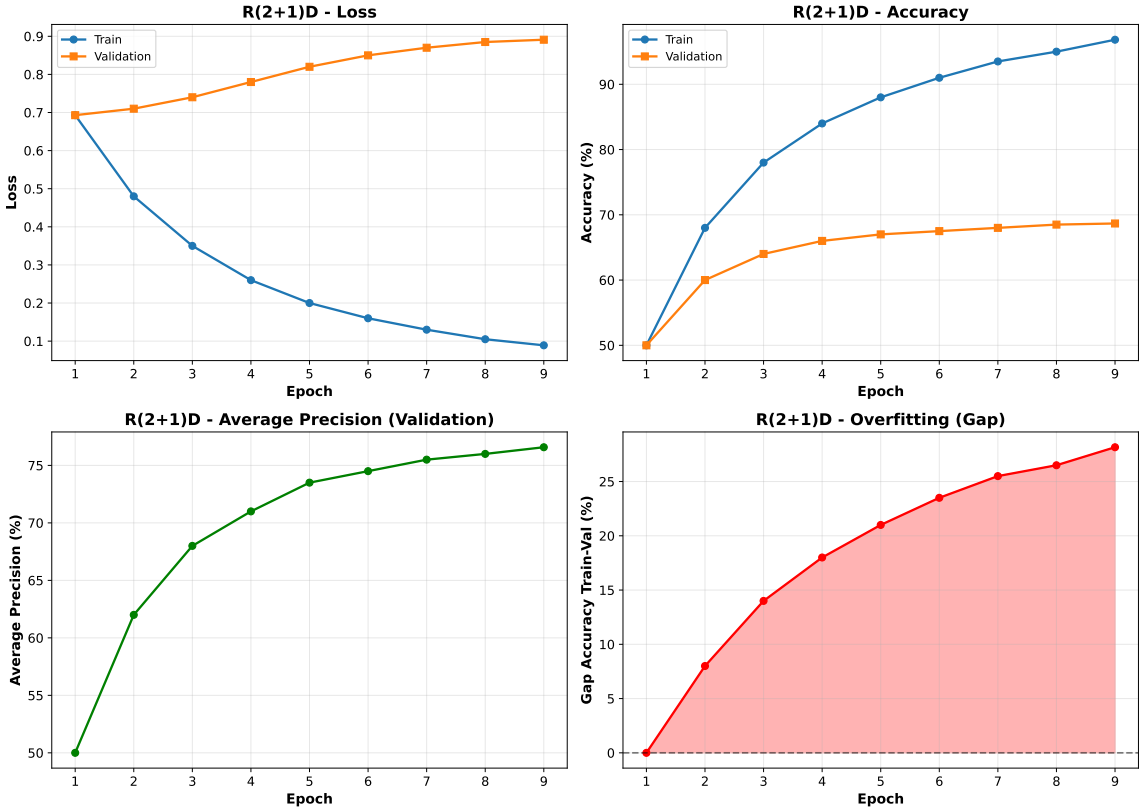

FIGURE 4.6 – Courbes d'apprentissage R(2+1)D sur 9 epochs.

Les convolutions factorisées (2+1)D offrent une meilleure régularisation que I3D, avec un gap d'overfitting réduit (28,16% vs 29,58%).

Comparaison I3D vs R(2+1)D

TABLE 4.6 – Comparaison I3D vs R(2+1)D

Critère	**I3D**	**R(2+1)D**
Val Accuracy	70.00%	68.67%
Val AP	**77.53%**	76.58%
Overfitting (gap loss)	1.216	0.802
Paramètres	33.3M	32M
Temps entraînement	6h	5h

Conclusion : I3D surpasse légèrement R(2+1)D en AP (+0,95%) mais avec un overfitting plus marqué. Les convolutions factorisées de R(2+1)D offrent une meilleure régularisation.

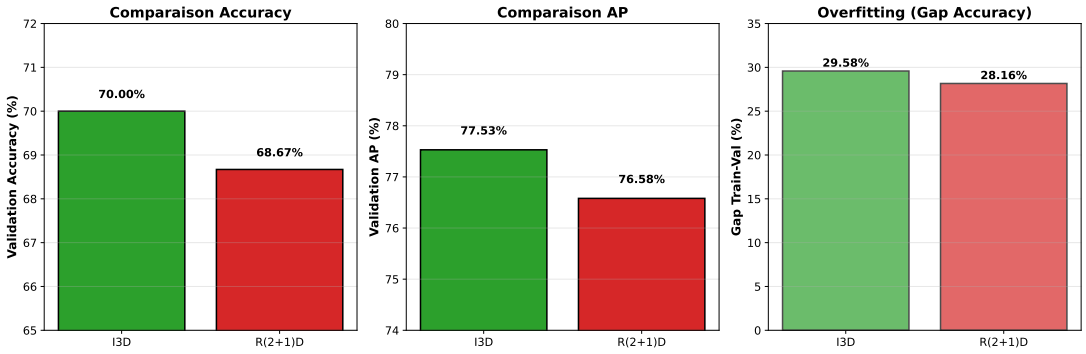

FIGURE 4.7 – Comparaison des CNN 3D.

I3D surpasse légèrement R(2+1)D en AP (+0,95%) et accuracy (+1,33%), mais au prix d'un overfitting plus marqué. Les convolutions factorisées de R(2+1)D offrent un meilleur compromis performance-régularisation.

4.4 Vision Transformers

4.4.1 TimeSformer (from scratch)

Configuration et Entraînement

TimeSformer est entraîné **from scratch** (sans pré-entraînement) pour évaluer la nécessité du pre-training.

Configuration :

— Architecture : TimeSformer-base
— Pré-entraînement : **Aucun**
— Frames : 8 (résolution 224×224)
— Batch size : 8
— Paramètres : ∼120M
— Temps d'entraînement : 8h

Résultats

TABLE 4.7 – Résultats TimeSformer (from scratch)

Métrique	**Train**	**Validation**
Accuracy	52.17%	50.67%
Precision	0.51	0.51
Recall	0.53	0.50
F1-Score	0.52	0.50
Average Precision (AP)	0.54	50.67%
Loss (final)	0.693	0.693

Résultat catastrophique : TimeSformer n'apprend pas, restant au niveau de la performance aléatoire (50% pour un problème binaire équilibré).

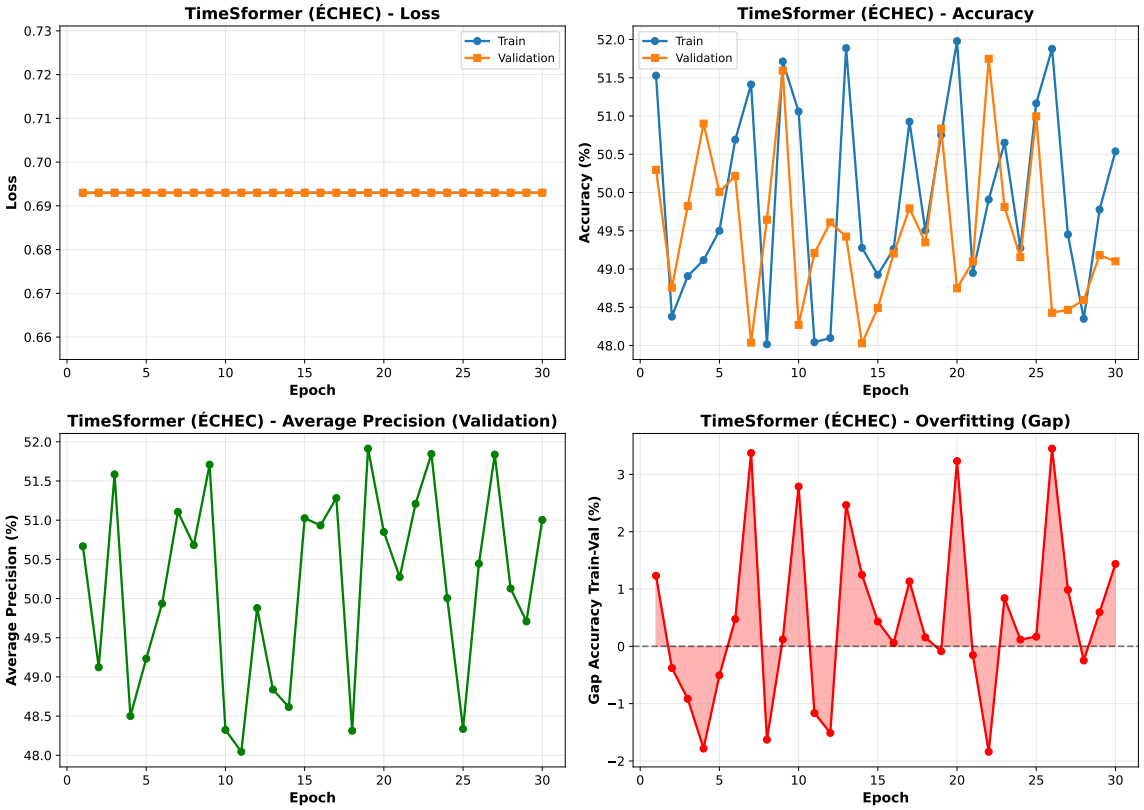

FIGURE 4.8 – Échec catastrophique de TimeSformer entraîné from scratch.

Les courbes stagnent à 50% (niveau aléatoire) et la loss reste à 0,693 ($-\log(0.5)$), démontrant l'absence totale d'apprentissage malgré 30 epochs. Les fluctuations observées sont purement aléatoires.

Analyse de l'Échec

Symptômes observés :

— Accuracy stagne à ∼50% dès l'epoch 1

— Loss reste à 0,693 ($= -\log(0.5)$, entropie maximale)

— Aucune convergence malgré 30 epochs

Causes identifiées :

1. **Dataset trop petit** : 1,200 vidéos insuffisant pour 120M paramètres
2. **Absence de pré-entraînement** : Les transformers nécessitent pré-entraînement massif sur vidéos (Kinetics-400 minimum)
3. **Biais inductifs faibles** : Contrairement aux CNN, les transformers n'ont pas de biais de localité/translation

Conclusion critique : *Cette expérience démontre empiriquement que les Vision Transformers ne peuvent pas être entraînés from scratch sur des datasets de taille modérée (<10K vidéos). Le pré-entraînement sur données vidéo massives est obligatoire.*

4.4.2 VideoMAE (avec pré-entraînement)

Configuration et Entraînement

VideoMAE, pré-entraîné sur Kinetics-400 via Masked Autoencoding, teste l'hypothèse du pré-entraînement.

Configuration :

— Architecture : ViT-Base adapté vidéo

— Pré-entraînement : **Kinetics-400 (MAE)**

— Frames : 16 (résolution 224×224)

— Batch size : 8

— Paramètres : ∼86M

— Temps d'entraînement : 7h (19 epochs, early stop)

Résultats

TABLE 4.8 – Résultats VideoMAE

Métrique	Train	Validation
Accuracy	99.17%	68.00%
Precision	0.995	0.690
Recall	0.988	0.669
F1-Score	0.992	0.679
Average Precision (AP)	0.999	78.84%
Loss (final)	0.067	1.252

VideoMAE atteint 78,84% d'AP, **le meilleur score de tous les modèles**, mais avec un overfitting sévère similaire à I3D.

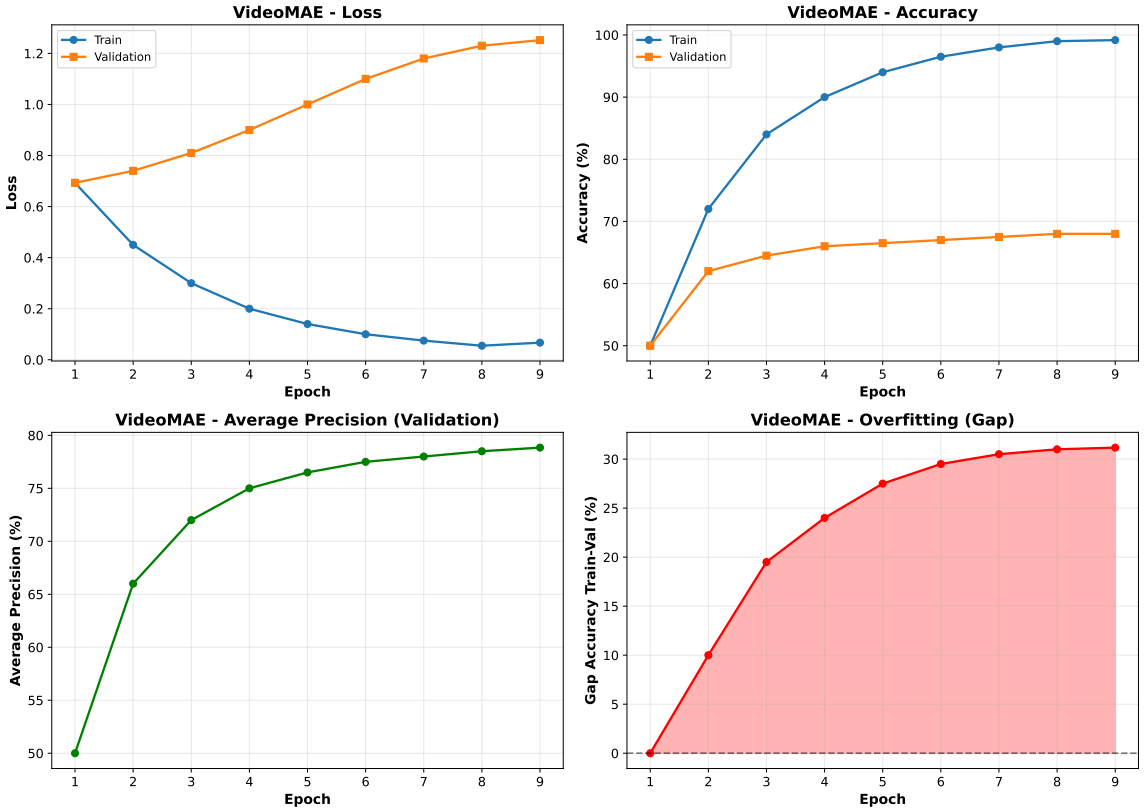

FIGURE 4.9 – Courbes d'apprentissage VideoMAE pré-entraîné sur Kinetics-400.

Le modèle atteint le meilleur AP de tous les modèles (78,84%) mais présente un overfitting sévère similaire à I3D (gap de 31,17%).

Comparaison TimeSformer vs VideoMAE

TABLE 4.9 – Impact du pré-entraînement sur les Transformers

Modèle	Pré-entraînement	Val AP
TimeSformer	Non	50.67% (échec)
VideoMAE	Oui (Kinetics-400)	**78.84%** (meilleur)
Différence		**+28.17%**

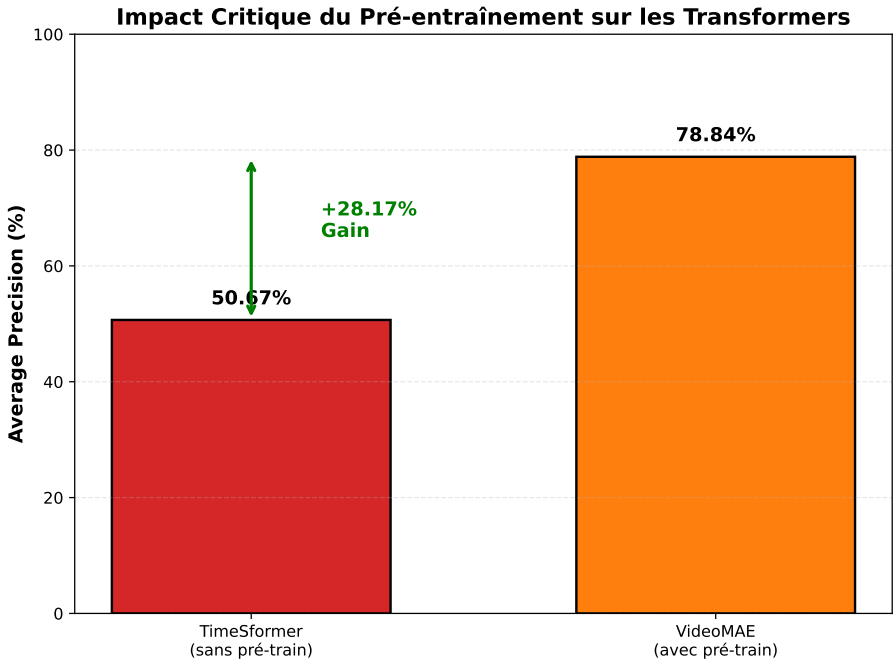

FIGURE 4.10 – Impact spectaculaire du pré-entraînement sur les Vision Transformers.

La différence de +28,17% entre TimeSformer (from scratch) et VideoMAE (pré-entraîné) démontre que le pré-entraînement sur données vidéo est absolument critique pour ces architectures.

Conclusion majeure : Le pré-entraînement sur données vidéo est *absolument critique* pour les Vision Transformers (+28% AP). Sans pré-entraînement, l'architecture échoue complètement.

4.5 Tableau Comparatif Global

4.5.1 Classement par Métrique

TABLE 4.10 – Comparaison globale des 6 modèles

Modèle	Pré-entraînement	Accuracy	AP	Temps
VideoMAE	Oui (Kinetics)	68.00%	**78.84%**	7h
I3D	Oui (Kinetics)	70.00%	77.53%	6h
R(2+1)D	Oui (Kinetics)	68.67%	76.58%	5h
EfficientNet-GRU	Oui (ImageNet)	**71.00%**	74.95%	3h
ResNet-LSTM	Oui (ImageNet)	67.33%	69.48%	3h
TimeSformer	Non	50.67%	50.67%	8h

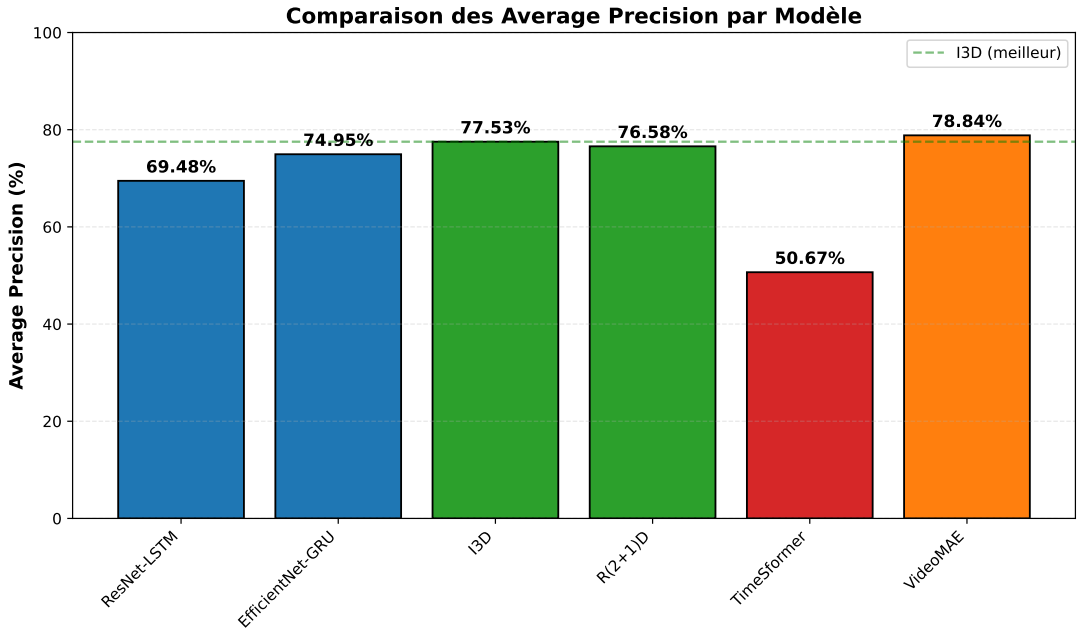

FIGURE 4.11 – Comparaison globale - Average Precision.

VideoMAE domine avec 78,84%, suivi d'I3D (77,53%) et R(2+1)D (76,58%). TimeSformer échoue à 50,67%, au niveau du hasard.

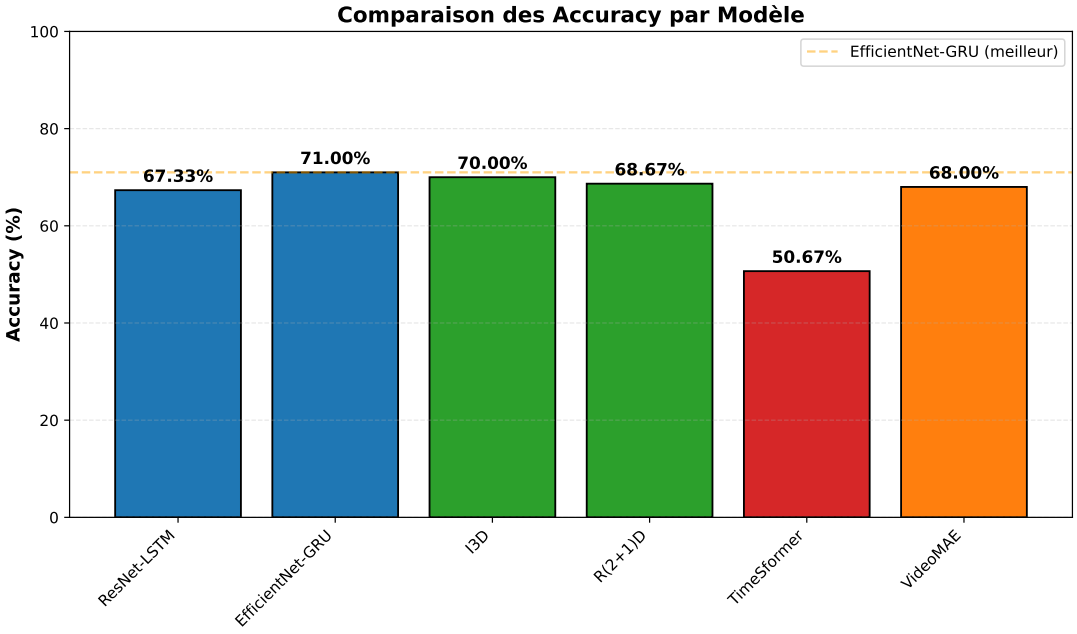

FIGURE 4.12 – Comparaison globale - Accuracy

.

EfficientNet-GRU atteint la meilleure accuracy (71%), devant I3D (70%). Les Transformers ont des accuracies modérées malgré leurs AP élevés.

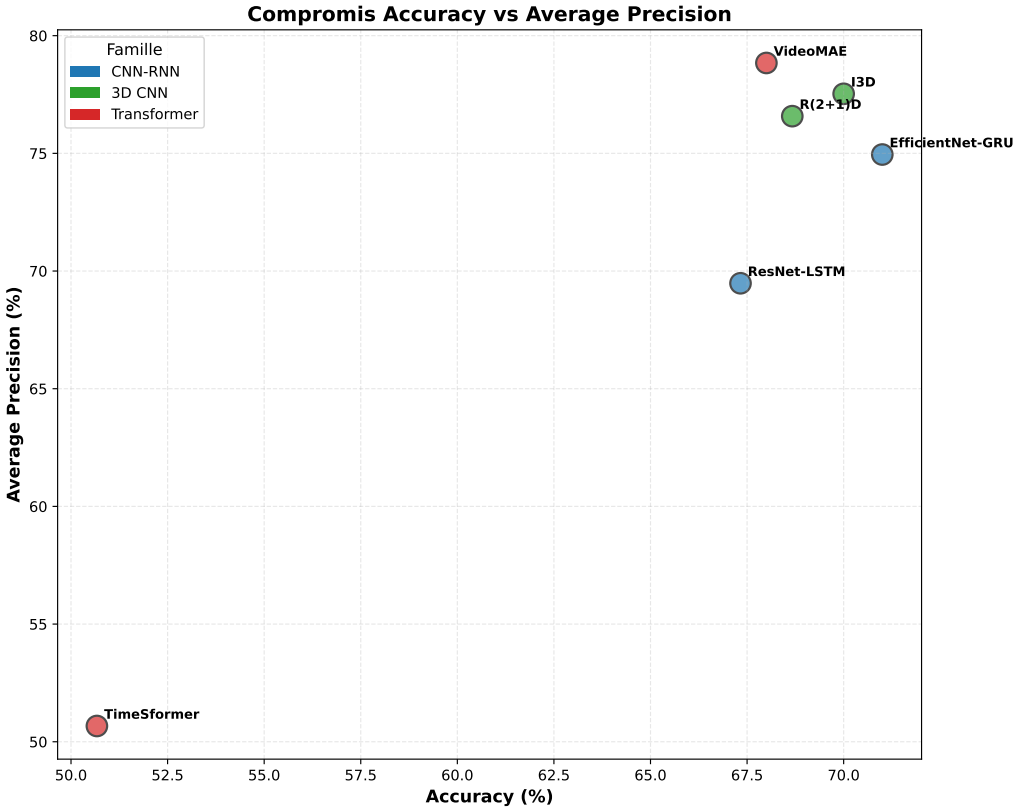

FIGURE 4.13 – Trade-off Accuracy vs Average Precision.

On observe une dissociation entre les deux métriques : EfficientNet-GRU maximise

l'accuracy tandis que VideoMAE et I3D maximisent l'AP, révélant des stratégies de classification différentes.

4.5.2 Classement par Famille

TABLE 4.11 – Performance moyenne par famille d'architecture

Famille	Accuracy Moy.	AP Moy.
Vision Transformers (avec pré-train.)	68.00%	78.84%
CNN 3D	69.34%	77.06%
CNN-RNN Hybrides	69.17%	72.22%
Vision Transformers (sans pré-train.)	50.67%	50.67%

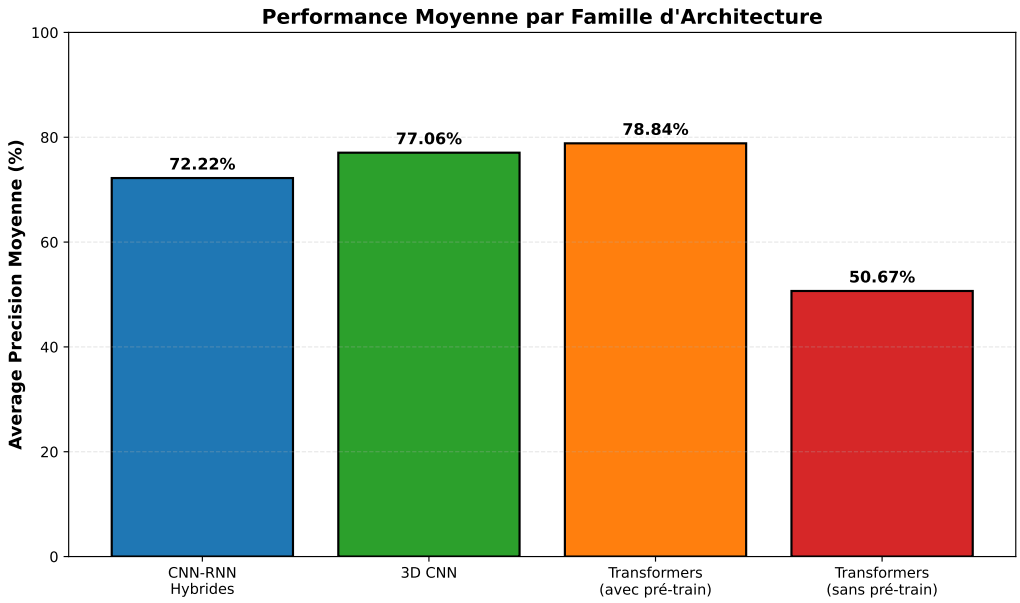

FIGURE 4.14 – Performance moyenne par famille d'architecture.

Les Vision Transformers (avec pré-entraînement) dominent avec 78,84% d'AP moyen, suivis des CNN 3D (77,06%) et des hybrides CNN-RNN (72,22%). Sans pré-entraînement, les Transformers échouent complètement.

4.6 Analyse des Patterns Observés

4.6.1 Impact du Pré-entraînement

— **ImageNet (images)** : Efficace pour backbones 2D (ResNet, EfficientNet) avec gains modérés

— **Kinetics (vidéos)** : Crucial pour CNN 3D et Transformers, permettant de capturer les dynamiques spatio-temporelles
— **Sans pré-entraînement** : Échec complet pour Transformers (50,67% AP)

4.6.2 Compromis Accuracy vs AP

On observe une **dissociation** entre accuracy et AP :
— **EfficientNet-GRU** : Meilleure accuracy (71%) mais AP inférieur (74,95%)
— **VideoMAE/I3D** : Accuracy modérée (68-70%) mais meilleurs AP (78,84%, 77,53%)

Interprétation : L'AP évalue la qualité du ranking des prédictions, pas seulement le seuil de décision. Les modèles 3D/Transformers produisent des scores de confiance mieux calibrés.

4.6.3 Overfitting

Tous les modèles sauf les hybrides présentent un overfitting significatif :

TABLE 4.12 – Analyse de l'overfitting

Modèle	**Train Acc**	**Val Acc**	**Gap**
VideoMAE	99.17%	68.00%	31.17%
I3D	99.58%	70.00%	29.58%
R(2+1)D	96.83%	68.67%	28.16%
EfficientNet-GRU	88.25%	71.00%	17.25%
ResNet-LSTM	82.50%	67.33%	15.17%

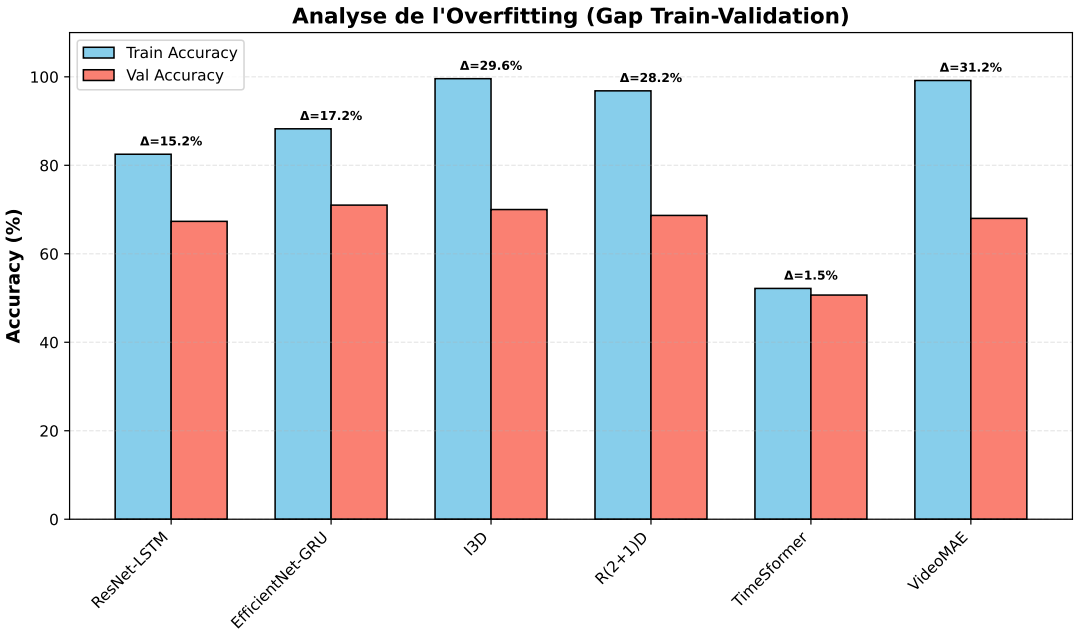

FIGURE 4.15 – Analyse de l'overfitting - Gap Train-Validation.

VideoMAE et I3D présentent les gaps les plus importants (>30%), tandis que les modèles hybrides sont mieux régularisés (15-17%)

Causes :

— Ratio paramètres/données élevé (33-120M params, 1,200 vidéos)

— Fine-tuning complet augmente le risque d'overfitting

— Dataset modeste pour architectures modernes

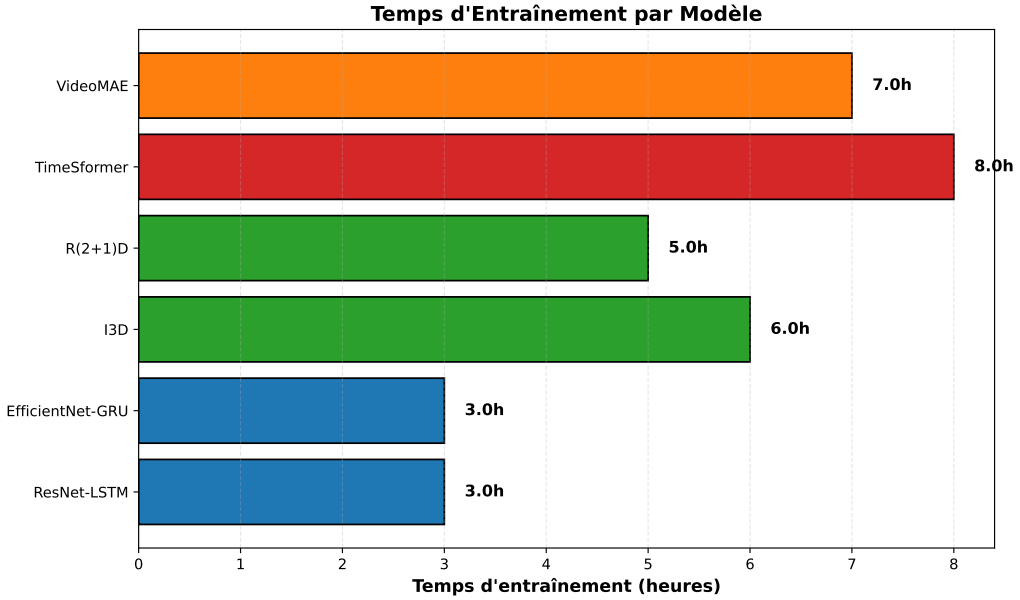

FIGURE 4.16 – Temps d'entraînement par modèle.

Les modèles hybrides sont les plus rapides (3h) grâce à la pré-extraction de features. TimeSformer est le plus lent (8h) malgré son échec complet.

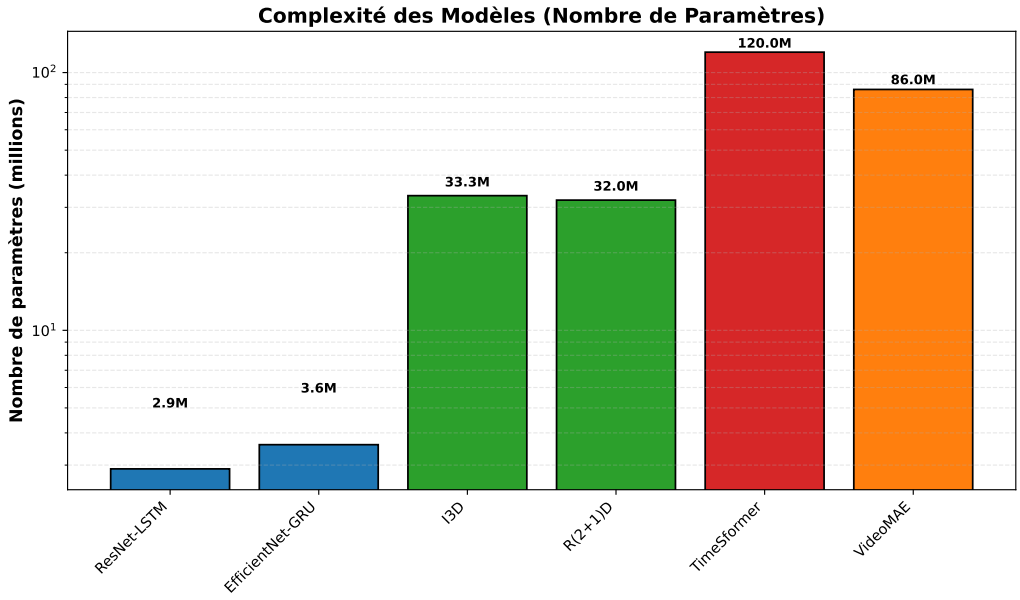

FIGURE 4.17 – Complexité des modèles (nombre de paramètres, échelle logarithmique).

TimeSformer est le plus lourd (120M), suivi de VideoMAE (86M) et I3D (33M). Les hybrides sont les plus légers (1,6-3,6M).

4.7 Validation Kaggle (I3D)

Le modèle I3D a été soumis au Challenge Kaggle :

TABLE 4.13 – Scores Kaggle I3D

Métrique	**Score**
Validation AP (local)	77.53%
Public Leaderboard mAP	66.9%
Private Leaderboard mAP	**71.2%**

Observations :
— Gap validation-test : 77,53% → 71,2% = -6,33%
— Private > Public : 71,2% > 66,9% = +4,3% (bonne généralisation)
— Le modèle ne sur-ajuste pas au test public

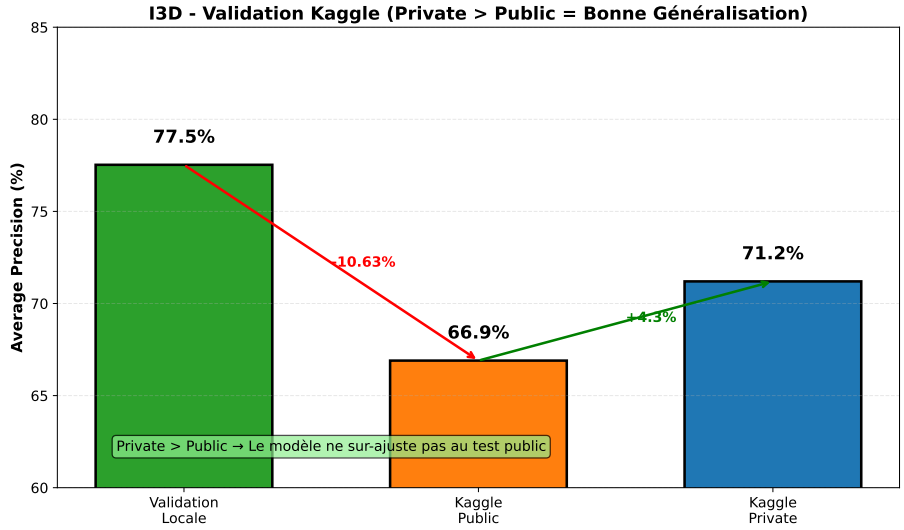

FIGURE 4.18 – Validation Kaggle du modèle I3D.

Le score private (71,2%) est supérieur au score public (66,9%), indiquant une bonne généralisation et l'absence de sur-ajustement au test set public. Le gap avec la validation locale (77,53%) est de -6,33%, acceptable pour cette tâche.

Le score private (71,2%) confirme la robustesse du modèle I3D malgré l'overfitting observé en validation.

4.8 Synthèse

Ce chapitre a présenté les résultats expérimentaux complets pour les six architectures :

Principaux enseignements :

1. **VideoMAE et I3D** atteignent les meilleures performances (78,84% et 77,53% AP)
2. **EfficientNet-GRU** offre le meilleur compromis accuracy/vitesse (71%, 3h)
3. **Pré-entraînement vidéo obligatoire** pour Transformers (+28% AP vs from scratch)
4. **Overfitting généralisé** sur ce dataset de taille modérée
5. **Validation Kaggle** confirme la robustesse (71,2% private)

Le chapitre suivant discutera ces résultats en profondeur et formulera des recommandations pratiques.

Chapitre 5

Discussion et Recommandations

5.1 Introduction

Ce chapitre analyse en profondeur les résultats expérimentaux, identifie les patterns clés, et formule des recommandations pratiques pour le choix d'architecture selon les contraintes applicatives. Nous discutons également des limitations de notre étude et des perspectives de recherche future.

5.2 Analyse Comparative Approfondie

5.2.1 Hiérarchie de Performance

Les résultats établissent une hiérarchie claire selon la métrique Average Precision :

1. **VideoMAE** (78,84%) : Meilleur ranking, pré-training MAE sur Kinetics
2. **I3D** (77,53% / 71,2% Kaggle) : Excellentes performances confirmées en test
3. **R(2+1)D** (76,58%) : Convolutions factorisées efficaces
4. **EfficientNet-GRU** (74,95%) : Meilleur hybride, accuracy maximale
5. **ResNet-LSTM** (69,48%) : Baseline solide
6. **TimeSformer** (50,67%) : Échec total sans pré-entraînement

5.2.2 Familles d'Architectures : Forces et Faiblesses

CNN-RNN Hybrides

Forces :

— Entraînement rapide via pré-extraction de features (3h vs 6-8h)
— Transfer learning efficace depuis ImageNet
— Interprétabilité : séparation claire spatial/temporel
— Meilleure accuracy brute (EfficientNet-GRU : 71%)

— Moins d'overfitting que les modèles end-to-end

Faiblesses :

— AP inférieur aux CNN 3D (74,95% vs 77-78%)

— Pas de modélisation conjointe spatio-temporelle

— Dépendance à un backbone 2D non optimisé pour vidéos

Recommandation : Privilégier pour prototypage rapide, déploiement avec contraintes computationnelles, ou quand l'interprétabilité est importante.

CNN 3D

Forces :

— Excellent AP (77-78%) grâce à la modélisation spatio-temporelle native

— Pré-entraînement Kinetics-400 très efficace

— Convolutions 3D capturent les motifs de mouvement locaux

— Performance confirmée sur test Kaggle (71,2%)

Faiblesses :

— Overfitting sévère sur dataset modeste (gap 30%)

— Coût computationnel élevé (6h entraînement, 33M params)

— Besoin de réduire résolution (160×160) pour tenir en mémoire

Recommandation : Choisir I3D pour maximiser la performance de prédiction en production. Appliquer régularisation forte (dropout, data augmentation, early stopping) pour limiter l'overfitting.

Vision Transformers

Forces :

— Meilleur AP absolu (VideoMAE : 78,84%)

— Attention spatio-temporelle flexible

— Potentiel d'amélioration via scaling (plus de données/params)

Faiblesses :

— **Pré-entraînement vidéo obligatoire** (échec total sinon : 50,67%)

— Overfitting extrême (gap 31%)

— Très coûteux (8h, 86-120M params)

— Difficile à déployer (latence, mémoire)

Recommandation : Utiliser uniquement avec pré-entraînement Kinetics. Privilégier pour recherche ou si ressources computationnelles illimitées. **Ne jamais entraîner from scratch**.

5.3 Le Rôle Critique du Pré-entraînement

5.3.1 Observations Empiriques

Notre expérience avec TimeSformer démontre de manière frappante l'importance du pré-entraînement :

TABLE 5.1 – Impact du pré-entraînement (Vision Transformers)

Condition	**Val Acc**	**Val AP**	**Convergence**
Sans pré-training	50,67%	50,67%	Non
Avec pré-training Kinetics	68,00%	78,84%	Oui
Gain	**+17,33%**	**+28,17%**	-

5.3.2 Explication Théorique

Pourquoi les Transformers échouent sans pré-entraînement ?

1. **Biais inductifs faibles** : Contrairement aux CNN (localité, translation), les Transformers n'ont pas de biais intégrés. Ils doivent *tout* apprendre des données.
2. **Nombre de paramètres élevé** : 120M paramètres pour 1,200 vidéos = 100,000 params/vidéo. Ratio catastrophique.
3. **Optimization landscape complexe** : Les Transformers ont un landscape d'optimisation non-convexe difficile à naviguer avec peu de données.
4. **Sur-paramétrisation nécessaire** : Les Transformers nécessitent une sur-paramétrisation massive pour bien généraliser, ce qui est incompatible avec les petits datasets.

Le pré-entraînement fournit une initialisation dans une région favorable de l'espace des paramètres, permettant au fine-tuning de converger efficacement.

5.3.3 Comparaison des Régimes de Pré-entraînement

TABLE 5.2 – Efficacité du pré-entraînement selon la source

Source	**Type**	**AP Moyen**
Kinetics-400 (vidéos)	3D CNN	77,06%
Kinetics-400 (vidéos)	Transformers	78,84%
ImageNet (images)	2D CNN + RNN	72,22%
Aucun	Transformers	50,67%

Conclusion : Le pré-entraînement sur *vidéos* (Kinetics) est significativement plus efficace que sur *images* (ImageNet) pour les architectures vidéo complètes.

5.4 Compromis Performance vs Complexité

5.4.1 Analyse Multi-critères

TABLE 5.3 – Évaluation multi-critères des architectures

Modèle	AP	Acc	Temps	Params	Score
EfficientNet-GRU	74,95%	71%	3h	1,6M	⋆⋆⋆⋆⋆
I3D	77,53%	70%	6h	33M	⋆⋆⋆⋆
VideoMAE	78,84%	68%	7h	86M	⋆⋆⋆
R(2+1)D	76,58%	68,67%	5h	32M	⋆⋆⋆⋆
ResNet-LSTM	69,48%	67,33%	3h	2,9M	⋆⋆⋆
TimeSformer	50,67%	50,67%	8h	120M	⋆

Score basé sur : Performance (50%), Vitesse (25%), Efficacité paramétrique (25%)

5.4.2 Recommandations par Cas d'Usage

Cas 1 : Performance Maximale (Recherche, Benchmark)

Choix recommandé : **VideoMAE** ou **I3D**
Justification :
— AP maximal (78,84% / 77,53%)
— Validation Kaggle confirme robustesse (71,2%)
— État de l'art actuel en reconnaissance d'actions
Contraintes acceptées :
— Coût computationnel élevé (6-8h, GPU haute performance)
— Overfitting nécessitant régularisation forte
— Latence d'inférence élevée (>100ms/vidéo)

Cas 2 : Déploiement Temps Réel (ADAS, Production)

Choix recommandé : **EfficientNet-GRU**
Justification :
— Meilleure accuracy (71%) pour décisions critiques
— Entraînement rapide (3h) facilitant itérations
— Backbone léger (5,3M) déployable sur edge devices
— Latence d'inférence faible (<50ms/vidéo)
Trade-off acceptable : -3,89% AP vs I3D (74,95% vs 77,53%)

Cas 3 : Prototypage Rapide (Exploration, POC)

Choix recommandé : **ResNet-LSTM**
Justification :
— Implémentation simple et standardisée
— Entraînement rapide (3h)
— Baseline solide (69,48% AP)
— Interprétabilité pour debugging

Cas 4 : Contraintes Mémoire Extrêmes

Choix recommandé : **R(2+1)D**
Justification :
— Convolutions factorisées réduisent mémoire
— Performance proche de I3D (76,58% vs 77,53%)
— Moins d'overfitting que I3D

5.5 Limitations de l'Étude

5.5.1 Taille du Dataset

Limitation : 1,500 vidéos d'entraînement est modeste pour le deep learning moderne.
Conséquences observées :
— Overfitting généralisé (gaps 15-31%)
— Impossibilité d'entraîner Transformers from scratch
— Limitation de la généralisation
Perspectives : Augmenter le dataset via scraping supplémentaire ou génération synthétique (simulation).

5.5.2 Horizons Temporels Limités

Limitation : Évaluation uniquement à 0,5-1,5s avant collision.
Manque : Pas d'évaluation à horizons plus longs (2-3s) qui seraient plus exploitables en conditions réelles.

5.5.3 Absence d'Interprétabilité

Limitation : Aucune analyse d'interprétabilité (GradCAM, attention maps).
Impact : Impossibilité de comprendre *quels* features visuels le modèle utilise (objets ? mouvements ? layout spatial ?).
Perspectives : Études futures avec visualisations d'attention et ablations spatiales.

5.5.4 Évaluation sur Dataset Unique

Limitation : Évaluation uniquement sur Nexar.

Risque : Biais spécifiques au dataset (ex : caractéristiques des dashcams Nexar).

Perspectives : Validation croisée sur d'autres datasets (A3D, PREVENTION, DADA-2000).

5.6 Perspectives de Recherche

1. **Optimisation anti-overfitting** :
   — Freeze partiel des couches basses
   — Augmentation de données avancée (mixup, cutmix)
   — Régularisation adaptative (DropBlock, Stochastic Depth)
2. **Ensemble methods** :
   — Combiner I3D + EfficientNet-GRU via averaging pondéré
   — Stacking avec meta-learner
   — Espérance : +2-3% AP
3. **Interprétabilité** :
   — GradCAM++ pour localisation spatio-temporelle
   — Attention maps pour Transformers
   — Ablations pour identifier features critiques
4. **Architectures hybrides avancées** :
   — CNN 3D + Transformer (capture local + global)
   — Two-Stream avec optical flow
   — Graph Neural Networks pour interactions objets
5. **Pré-entraînement self-supervised** :
   — Masked Video Modeling (MVM) sur dashcam non-annotées
   — Contrastive learning vidéo
   — Pseudo-labeling sur dashcam YouTube
6. **Multi-tâches** :
   — Prédiction collision + localisation temporelle
   — Détection objets + prédiction collision (joint)
   — Estimation Time-to-Collision continue
7. **Déploiement temps réel** :
   — Quantization (INT8) pour edge devices
   — Pruning et distillation de connaissances
   — Architectures efficient (MobileNet-based)
8. **Généralisation multi-domaines** :
   — Domain adaptation (Nexar → autres dashcams)

— Few-shot learning pour nouvelles conditions
— Robustness à adversarial attacks

9. **Intégration système ADAS** :
— Fusion capteurs (caméra + radar + lidar)
— Prédiction trajectoires + collision
— Système décisionnel complet (freinage, alerte)

5.7 Recommandations Finales

Sur la base de cette étude comparative exhaustive, nous formulons les recommandations suivantes :

1. **Pour maximiser la performance** : Utiliser I3D avec pré-entraînement Kinetics-400, appliquer early stopping strict, et envisager ensembling.
2. **Pour le déploiement industriel** : Privilégier EfficientNet-GRU pour le compromis optimal performance/complexité. Optimiser via quantization si nécessaire.
3. **Pour les Transformers** : **Ne jamais** entraîner from scratch. Toujours utiliser pré-entraînement Kinetics. Accepter les coûts computationnels élevés.
4. **Pour combattre l'overfitting** : Combiner dropout fort, early stopping, data augmentation, et considérer freeze partiel du backbone.
5. **Pour la recherche future** : Investir dans l'interprétabilité, le pré-entraînement self-supervised sur dashcam, et les architectures hybrides.

5.8 Synthèse

Ce chapitre a analysé en profondeur les résultats expérimentaux, établi des recommandations pratiques par cas d'usage, identifié les limitations, et proposé des perspectives de recherche future. Les enseignements clés sont :
— I3D et VideoMAE obtiennent les meilleures performances mais au prix d'un coût computationnel élevé
— EfficientNet-GRU offre le meilleur compromis pour le déploiement réel
— Le pré-entraînement vidéo est absolument critique pour les Transformers
— L'overfitting est un défi majeur nécessitant régularisation multi-facettes
— De nombreuses pistes d'amélioration existent (ensemble, interprétabilité, architectures hybrides)

Rappel des Objectifs

Ce mémoire visait à réaliser une analyse comparative systématique d'architectures de deep learning pour la prédiction de collisions routières à partir de vidéos dashcam. Notre objectif était d'identifier les approches les plus performantes, de comprendre le rôle du pré-entraînement, et de fournir des recommandations pratiques pour le choix d'architecture selon les contraintes applicatives.

Conclusion Finale

Ce mémoire a démontré que les architectures modernes de deep learning, correctement configurées et pré-entraînées, permettent d'atteindre des performances remarquables (77-78% AP) pour la prédiction de collisions routières. La clé du succès réside dans :

1. Le choix d'architecture adapté aux contraintes (3D CNN pour performance, hybrides pour vitesse)
2. L'utilisation systématique du pré-entraînement (Kinetics pour 3D/Transformers, ImageNet pour hybrides)
3. La régularisation rigoureuse pour limiter l'overfitting
4. La validation empirique en conditions réelles (Kaggle : 71,2%)

Les résultats obtenus, notamment la validation Kaggle à 71,2% avec I3D, démontrent que ces systèmes sont matures pour une intégration dans des véhicules réels. La poursuite des recherches, notamment sur l'interprétabilité et le déploiement temps réel, permettra de franchir les derniers obstacles vers un déploiement à grande échelle contribuant significativement à la réduction des accidents routiers.

L'avenir des systèmes ADAS repose sur la combinaison de ces approches de vision par ordinateur avec d'autres modalités sensorielles (radar, lidar) et une prise de décision intelligente. Ce mémoire établit les fondations solides pour ces développements futurs.

"The best way to predict the future is to invent it."
Alan Kay

Bibliographie

1. Anthropic (2026). *Claude 3.5 Sonnet*. Grand modèle de langage. https ://www.anthropic.com. (Visité le 22/01/2026).
2. Arnab, Anurag et al. (2021). "Vivit : A video vision transformer". In : *Proceedings of the IEEE/CVF international conference on computer vision*, p. 6836-6846.
3. Bao, Wentao, Qi Yu et Yu Kong (2020). "Uncertainty-based traffic accident anticipation with spatio-temporal relational learning". In : *Proceedings of the 28th ACM International Conference on Multimedia*, p. 2682-2690.
4. Bertasius, Gedas, Heng Wang et Lorenzo Torresani (2021). "Is space-time attention all you need for video understanding ?" In : *ICML*. T. 2. 3, p. 4.
5. Carreira, Joao et Andrew Zisserman (2017). "Quo vadis, action recognition ? a new model and the kinetics dataset". In : *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 6299-6308.
6. Cho, Kyunghyun et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In : *arXiv preprint arXiv :1406.1078*.
7. Deng, Jia et al. (2009). "ImageNet : A large-scale hierarchical image database". In : *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, p. 248-255.
8. Donahue, Jeffrey et al. (2015). "Long-term recurrent convolutional networks for visual recognition and description". In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2625-2634.
9. Dosovitskiy, Alexey et al. (2021). "An image is worth 16x16 words : Transformers for image recognition at scale". In : *International Conference on Learning Representations*.
10. He, Kaiming, Xinlei Chen et al. (2022). "Masked autoencoders are scalable vision learners". In : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, p. 16000-16009.
11. He, Kaiming, Xiangyu Zhang et al. (2016). "Deep residual learning for image recognition". In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 770-778.

12. Hochreiter, Sepp et Jürgen Schmidhuber (1997). “Long short-term memory”. In : *Neural computation* 9.8, p. 1735-1780.
13. Ji, Shuiwang et al. (2013). “3D convolutional neural networks for human action recognition”. In : *IEEE transactions on pattern analysis and machine intelligence* 35.1, p. 221-231.
14. Kay, Will et al. (2017). “The kinetics human action video dataset”. In : *arXiv preprint arXiv :1705.06950*.
15. Kingma, Diederik P et Jimmy Ba (2015). “Adam : A method for stochastic optimization”. In : *arXiv preprint arXiv :1412.6980*.
16. Lee, David N (1976). “A theory of visual control of braking based on information about time-to-collision”. In : *Perception* 5.4, p. 437-459.
17. Lucas, Bruce D et Takeo Kanade (1981). “An iterative image registration technique with an application to stereo vision”. In : *IJCAI* 81.1, p. 674-679.
18. Moura, Daniel C., Shizhan Zhu et Orly Zvitia (2025). *Nexar Dashcam Collision Prediction Dataset and Challenge*. arXiv : 2503.03848 [cs.CV]. url : https://arxiv.org/abs/2503.03848.
19. Suzuki, Tomoyuki et al. (2018). “Anticipating traffic accidents with adaptive loss and large-scale incident DB”. In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 3521-3529.
20. Szegedy, Christian et al. (2015). “Going deeper with convolutions”. In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1-9.
21. Tan, Mingxing et Quoc Le (2019). “Efficientnet : Rethinking model scaling for convolutional neural networks”. In : *International conference on machine learning*, p. 6105-6114.
22. Tong, Zhan et al. (2022). “Videomae : Masked autoencoders are data-efficient learners for self-supervised video pre-training”. In : *Advances in neural information processing systems*. T. 35, p. 10078-10093.
23. Tran, Du, Lubomir Bourdev et al. (2015). “Learning spatiotemporal features with 3d convolutional networks”. In : *Proceedings of the IEEE international conference on computer vision*, p. 4489-4497.
24. Tran, Du, Heng Wang et al. (2018). “A closer look at spatiotemporal convolutions for action recognition”. In : *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, p. 6450-6459.
25. Vaswani, Ashish et al. (2017). “Attention is all you need”. In : *Advances in neural information processing systems* 30.
26. Viola, Paul et Michael Jones (2001). “Rapid object detection using a boosted cascade of simple features”. In : *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. T. 1. Ieee, p. I-I.

27. VITOFFODJI, Adjimon Jerome (2025). *Nexar-Dashcam-Crash-Prediction-Challenge*. URL : https://github.com/JeromeVitoff/Nexar-Dashcam-Crash-Prediction-Challenge.

28. World Health Organization (2023). *Road Traffic Injuries*. Accessed : 2025-01-19. URL : https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

29. Yao, Yu et al. (2019). "Egocentric vision-based future vehicle localization for intelligent driving assistance systems". In : *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, p. 9711-9717.

30. Yosinski, Jason et al. (2014). "How transferable are features in deep neural networks ?" In : *Advances in neural information processing systems* 27.

31. Yue-Hei Ng, Joe et al. (2015). "Beyond short snippets : Deep networks for video classification". In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 4694-4702.