Rapport de projet

Cours IFT712 : Techniques d'apprentissage

Romain Brouard - BROR6259
romain.brouard@usherbrooke.ca

Paul Henry - HENP7261
paul.henry@usherbrooke.ca

Université de Sherbrooke
Département d'informatique
Automne 2025

Table des matières

1 Introduction

1.1 Contexte

Ce projet se réalise dans le cadre du cours IFT712 - Techniques d'apprentissage de la période d'automne 2025.
Nous avons choisi de travailler avec 6 modèles que nous avons vus en classe afin de classifier des images provenant du dataset CIFAR-10 Kaggle.
Dans ce rapport, nous vous présenterons le jeu de données CIFAR-10, les différents modèles que nous avons utilisés et codés puis les résultats que nous obtenons.

Notre projet est accessible via le lien suivant : https://github.com/romain327/prj-IFT712 et les modèles pré-entraînés via celui-ci : https://drive.proton.me/urls/411Z1HC3A0#7h8UeXRQAyPu
Le dataset peut se télécharger via Kaggle au lien suivant : https://www.kaggle.com/datasets/pankrzysiu/cifar10-python/data

1.2 Dataset

Le dataset ou jeu de données CIFAR-10, trouvé sur le site Kaggle, comporte 60 000 images couleur de faible résolution, soit 32×32 pixels. Ces données sont réparties de manière équilibrée suivant 10 classes distinctes et mutuellement exclusives : avion, automobile, oiseau, chat, cerf, chien, grenouille, cheval, navire et camion. L'ensemble est divisé en deux sous-ensembles principaux pour les besoins de l'apprentissage :

- Un ensemble d'entraînement (*training set*) constitué de 50 000 images ;
- Un ensemble de test (*test set*) contenant les 10 000 images restantes.

Chaque image est codée sur trois canaux (Rouge, Vert, Bleu) et est de faible résolution ce qui permet d'évaluer la capacité de modèles sur des objets du quotidien.

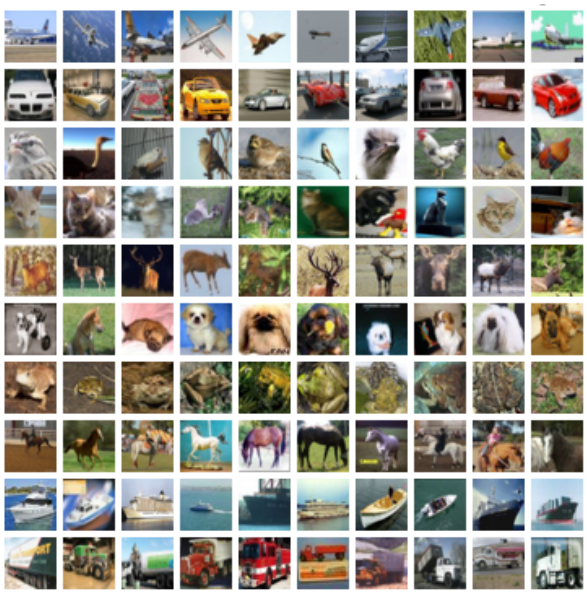


FIGURE 1 – Images du dataset CIFAR-10

1.3 Métrique d'évaluation : F1-score

Afin d'évaluer les solutions proposées par nos différents modèles et d'établir un classement pertinent, nous nous sommes appuyés sur le score F1 (ou F1-score) calculé sur l'ensemble de test. Le score F1 est une métrique qui compare les résultats prédits par l'intelligence artificielle à la vérité terrain (*ground truth*). Il est particulièrement utile car il combine deux autres mesures essentielles : la précision et le rappel.

Le score F1 est calculé selon la formule suivante :

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{1}$$

Où *Precision* correspond à la précision et *Recall* au rappel.

La formule de la précision est définie par :

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Et celle du rappel est :

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Dans ces équations, les termes signifient :

- TP (True Positive) : Vrais Positifs - L'IA a correctement prédit la classe positive.
- FP (False Positive) : Faux Positifs - L'IA a prédit la classe positive alors qu'elle était négative.
- FN (False Negative) : Faux Négatifs - L'IA a prédit la classe négative alors qu'elle était positive.

La figure suivante illustre ces concepts :

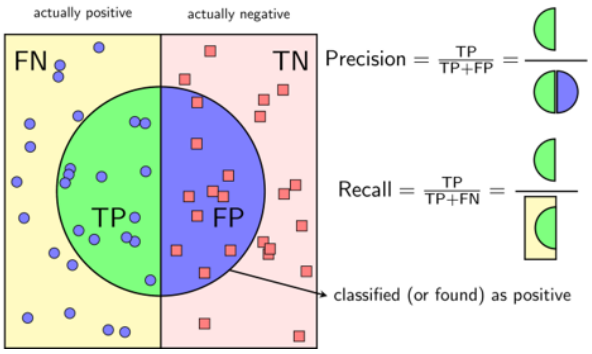


FIGURE 2 – Schéma explicatif des métriques (TP, FP, FN)

Dans le cadre de ce projet, les prédictions de chaque modèle ont été confrontées aux étiquettes réelles de l'ensemble de test, nous permettant ainsi de calculer ces métriques et de déterminer quel modèle offre les meilleures performances.

2 Génie logiciel

Pour rendre l'utilisation des modèles un peu interactive, nous avons décidé de développer une application avec une interface graphique pour interagir avec les fonctionnalités d'entraînement, de test et de classification d'une nouvelle image du modèle.
Pour cela, nous commençons par énumérer les cas d'utilisations.

2.1 Cas d'utilisation

Cas 1 : Entraîner un modèle :

- Description : L'utilisateur doit pouvoir (ré-)entraîner un modèle (au choix parmi les 6 disponibles) sur son propre dataset, en sélectionnant le volume de données souhaité. Le modèle doit pouvoir être testé dans la foulée si l'entraînement aboutit.
- Pré-condition : Le système est dans son état "idle", c'est à dire qu'il n'est ni dans son état d'entraînement, de test ou d'inférence.
- Scénario :
  - Fonctionnement normal : L'entraînement s'effectue sans lever d'exception et le système retourne à son état "idle".
  - Levée d'une exception : Le système retourne à son état "idle", l'erreur est loguée et un message d'alerte est affiché.
- Post-condition : Le système retourne à son état "idle" et un onglet avec les résultats de l'entraînement est généré et sélectionné en fonctionnement normal, ou l'onglet log est sélectionné en erreur.

Cas 2 : Test d'un modèle

- Description : L'utilisateur doit pouvoir tester un modèle entraîné. Le modèle doit-être testé sur le même dataset que sur lequel il a été entraîné.
- Pré-condition : Le système est dans son état "idle", c'est à dire qu'il n'est ni dans son état d'entraînement, de test ou d'inférence. Le modèle a été entraîné et le dataset utilisé pour l'entraînement est disponible.
- Scénario :
  - Fonctionnement normal : Le test s'effectue sans lever d'exception et le système retourne à son état "idle".
  - Levée d'une exception : Le système retourne à son état "idle", l'erreur est loguée et un message d'alerte est affiché.
- Post-condition : Le système retourne à son état "idle" et un onglet avec les résultats du test est généré et sélectionné en fonctionnement normal, ou l'onglet log est sélectionné en erreur.

Cas 3 : Classification d'une nouvelle image ou de plusieurs nouvelles images

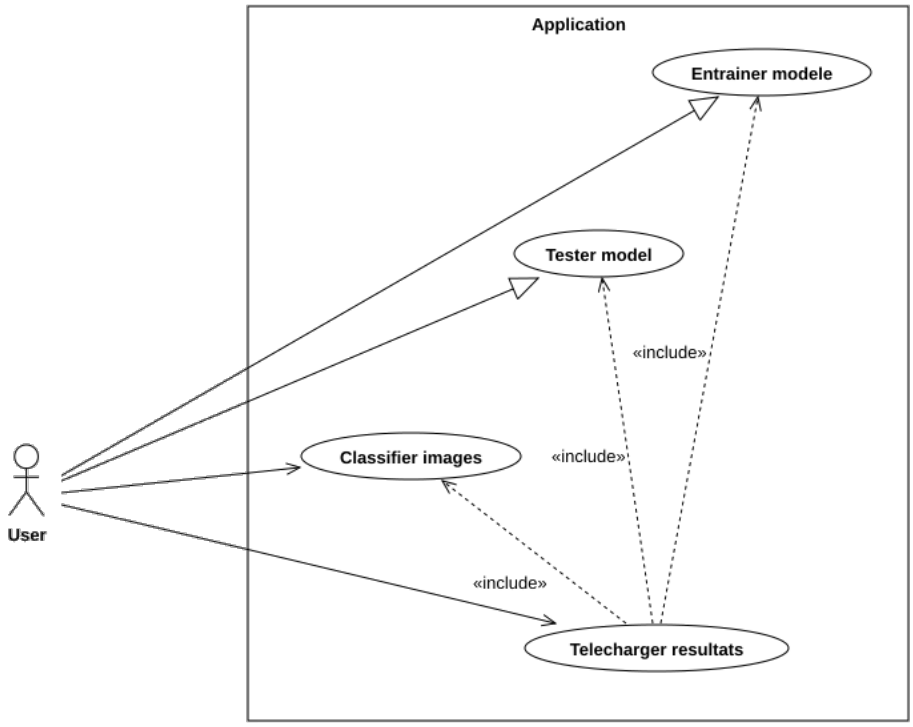
- Description : L'utilisateur doit pouvoir charger une image ou un dossier contenant plusieurs images pour qu'elles soient classifiées.
- Pré-condition : Le système est dans son état "idle", c'est à dire qu'il n'est ni dans son état d'entraînement, de test ou d'inférence. Les images doivent respecter le format précisé.

- Scénario :
  - Fonctionnement normal : La classification s'effectue sans lever d'exception et le système retourne à son état "idle".
  - Levée d'une exception : Le système retourne à son état "idle", l'erreur est loguée et un message d'alerte est affiché.
- Post-condition : Le système retourne à son état "idle" et un onglet avec les images et leur classe sont disponibles dans un onglet généré et sélectionné en fonctionnement normal, ou l'onglet log est sélectionné en erreur.

Cas 4 : Téléchargement des résultats de la classification

- Description : L'utilisateur doit pouvoir télécharger les résultats issus de la classification.
- Pré-condition : Le système est dans son état "idle", c'est à dire qu'il n'est ni dans son état d'entraînement, de test ou d'inférence. Une classification doit avoir été effectuée et doit avoir abouti (le système n'est pas en erreur).
- Scénario :
  - Fonctionnement normal : Les résultats sont téléchargés sans lever d'exception et le système retourne à son état "idle".
  - Levée d'une exception : Le système retourne à son état "idle", l'erreur est loguée et un message d'alerte est affiché.
- Post-condition : Le système retourne à son état "idle".

Nous représentons ces cas d'utilisations dans le diagramme suivant :



2.2 Analyse métier

Pour répondre aux cas d'utilisations, nous pouvons lister quelques objets métiers qui vont nous permettre d'effectuer un premier jet pour notre application. Si notre application suit une architecture Modèle-Vue-Contrôleur (MVC) classique, nous obtenons les objets suivants :

— Window : Interface graphique de l'application

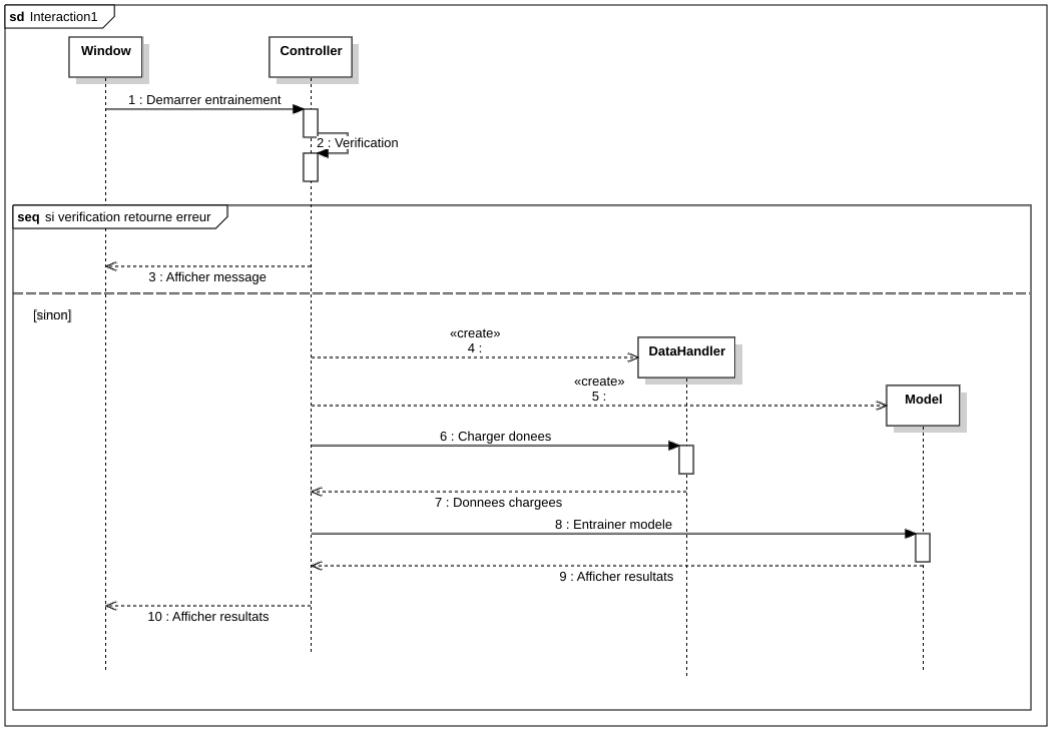
— Controller : Contrôleur de l'interface graphique (fait le lien entre l'interface et le reste du backend)

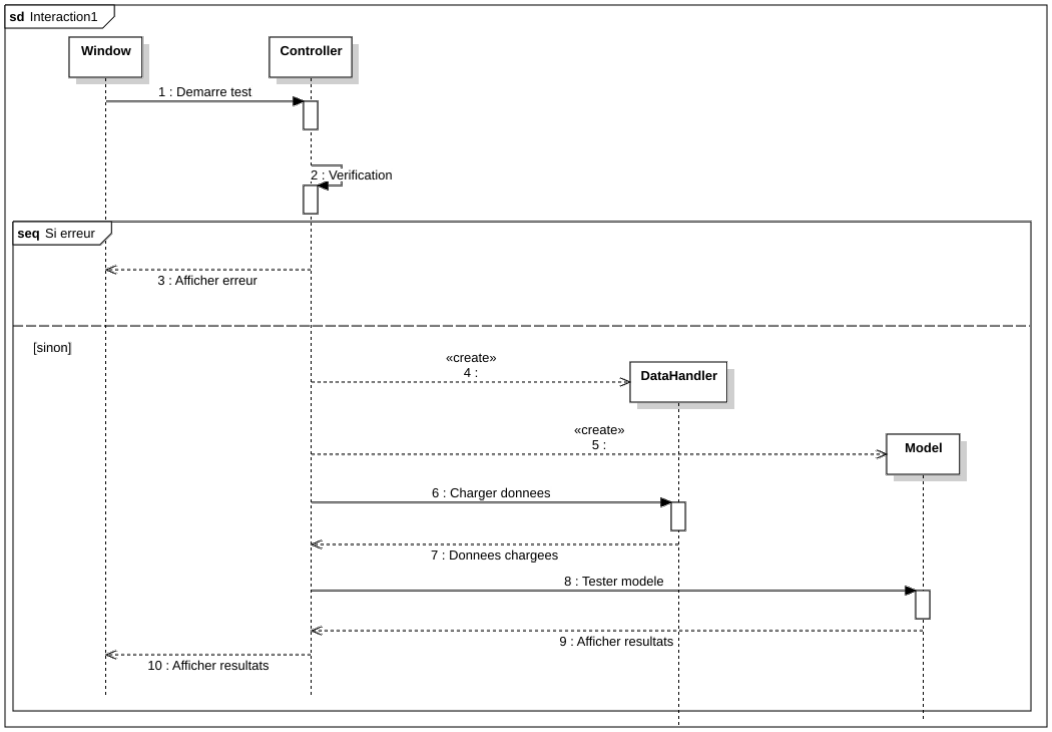
— DataHandler : Objet permettant d'avoir des outils pour charger et formater les données (dataset)

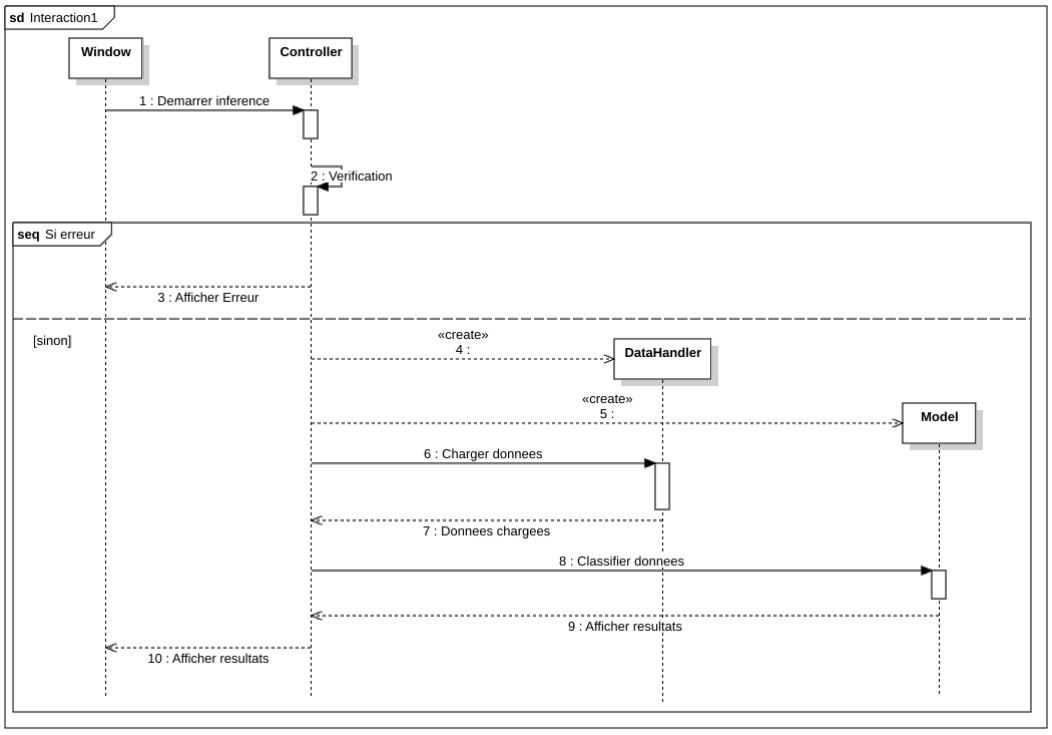
— Model : Objet représentant un modèle de classification

2.3 Modélisation

Les diagrammes ci-dessous décrivent le fonctionnement de l'application du point de vue des interactions entre les objets métiers pour chaque cas d'utilisation.

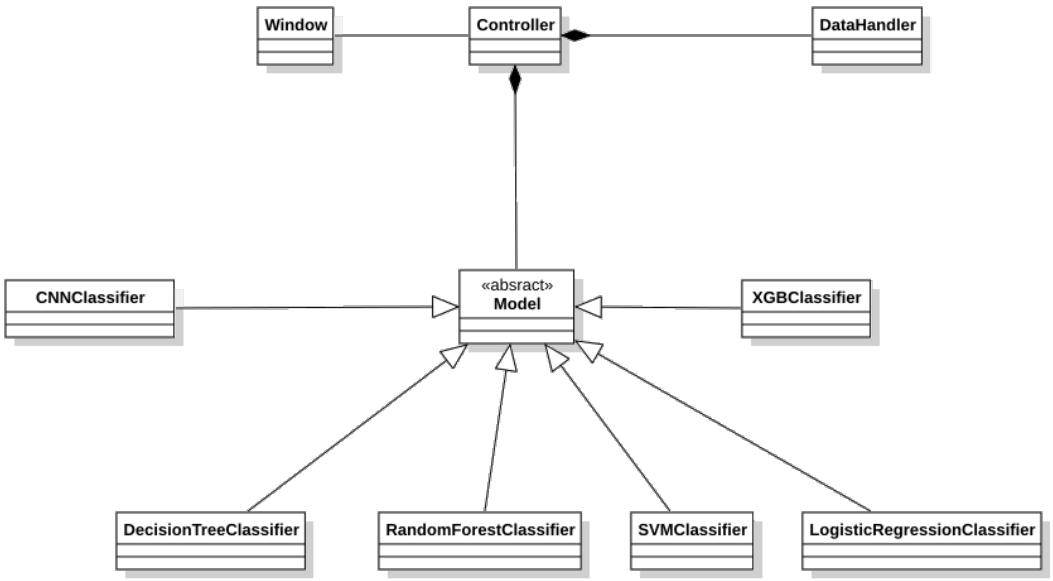




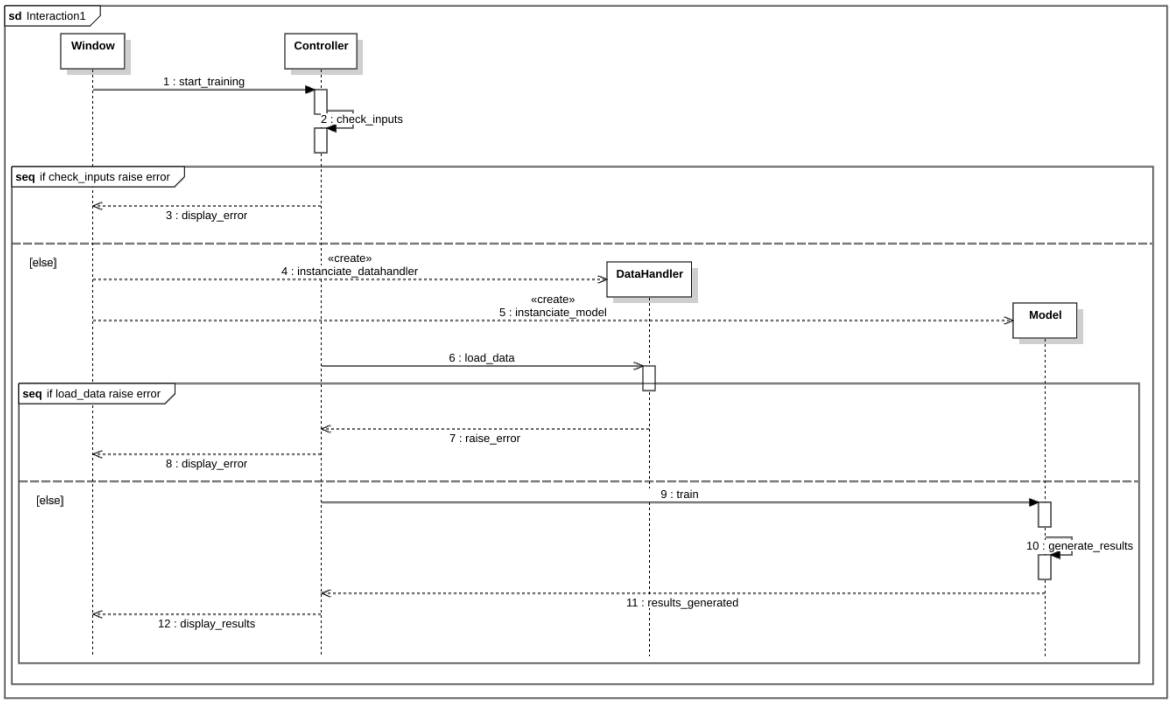


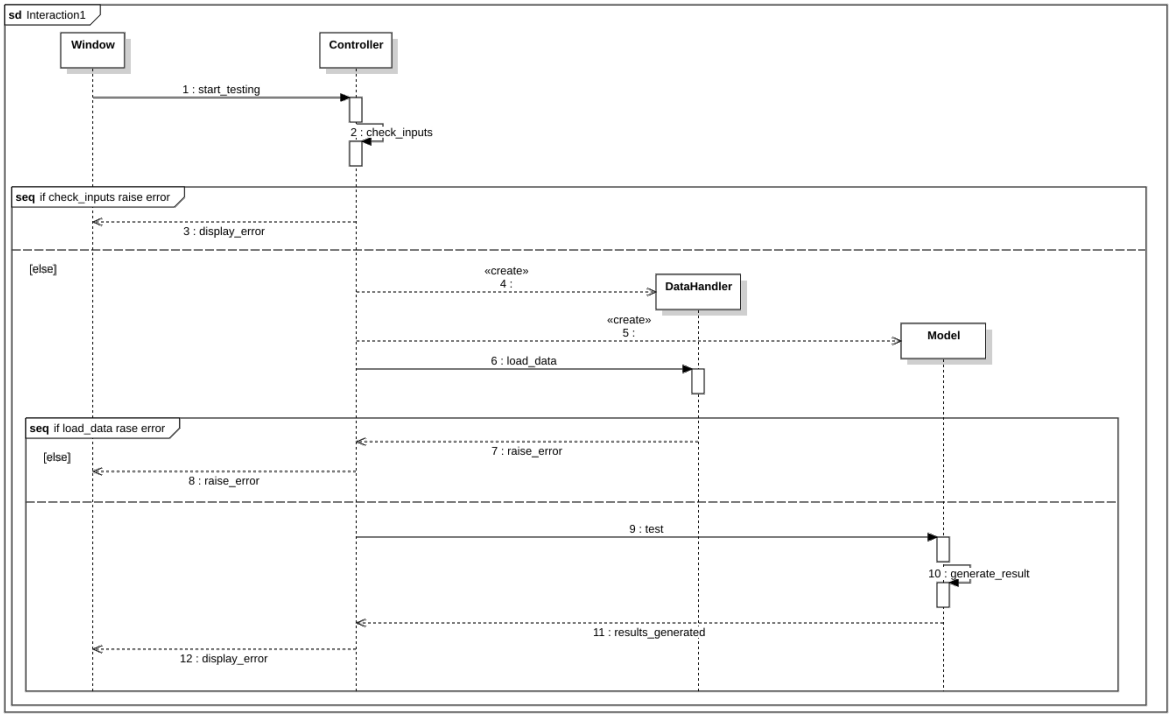
Ces diagrammes nous permettent de visualiser de manière globale le fonctionnement de notre application. Ces diagrammes sont valables pour n'importe quel modèle, mais nous devons implémenter 6 manières (modèles) différentes pour classifier nos images. Nous pouvons donc

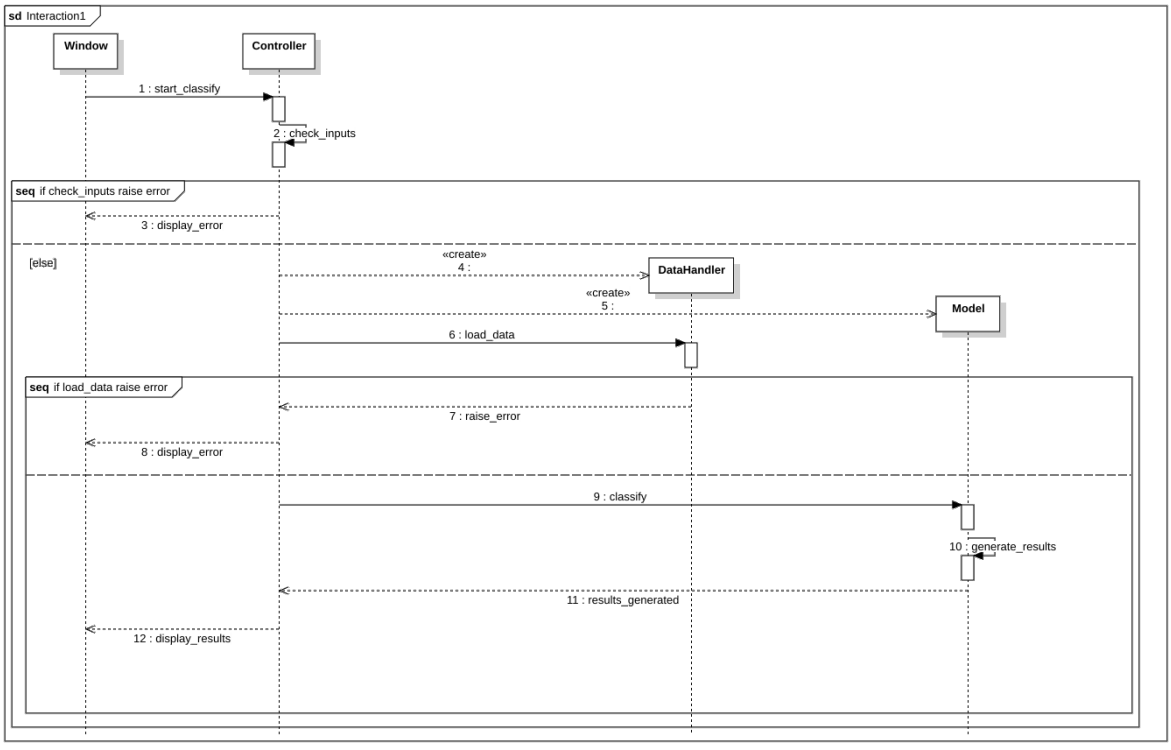
considérer la classe modèle comme abstraite, et définir une sous-classe par modèle implémenté. Tout cela nous donne le premier diagramme de classes suivant :



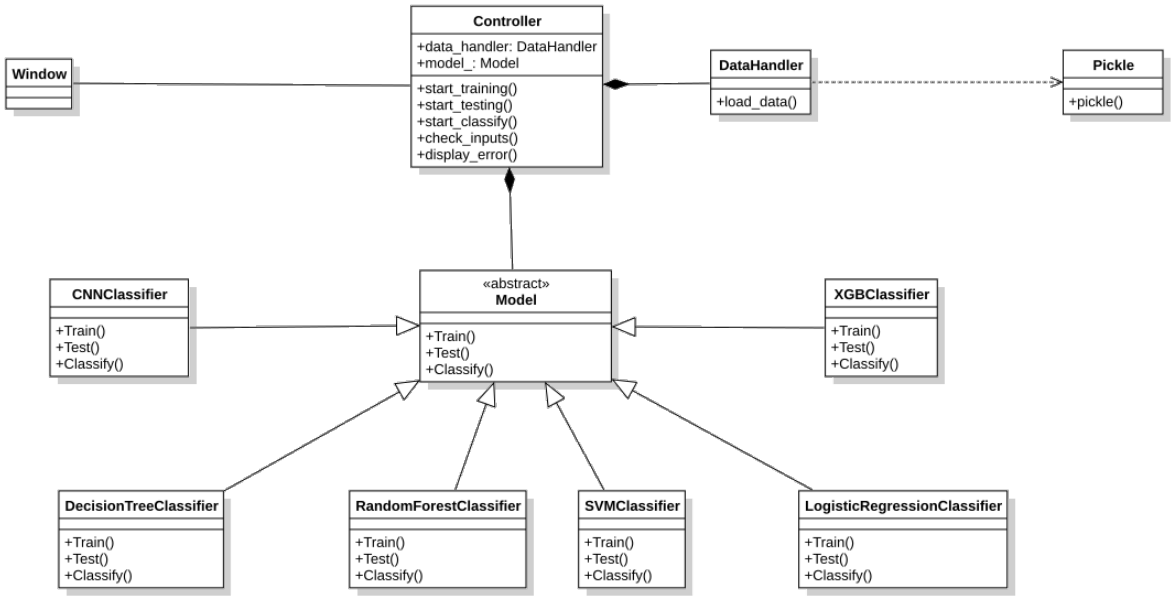
En affinant nos diagrammes pour détailler le fonctionnement de chaque cas d'utilisation, et en remplaçant les appels par des noms de fonctions, nous obtenons les diagrammes de séquences suivant :




sd Interaction1
Window
Controller
1 : start_testing
2 : check_inputs
seq if check_inputs raise error
3 : display_error
[else]
«create»
4 :
DataHandler
«create»
5 :
Model
6 : load_data
seq if load_data rase error
7 : raise_error
[else]
8 : raise_error
9 : test
10 : generate_result
11 : results_generated
12 : display_error


sd Interaction1
Window
Controller
1 : start_classify
2 : check_inputs
seq if check_inputs raise error
3 : display_error
[else]
«create»
4 :
DataHandler
«create»
5 :
Model
6 : load_data
seq if load_data raise error
7 : raise_error
8 : display_error
9 : classify
10 : generate_results
11 : results_generated
12 : display_results

Ce qui nous permet de compléter notre diagramme de classe :



3 Modèles utilisés

Pour répondre à cette problématique de classification multi-classes sur le jeu de données CIFAR-10, nous avons implémenté et comparé six architectures distinctes. Notre approche consiste à partir de modèles classiques et interprétables pour aller vers des méthodes d'ensemble et enfin vers l'apprentissage profond.

3.1 Modèles classiques

Arbre de décision (Decision Tree)
Nous avons débuté par un classificateur par arbre de décision. Le principe fondamental de ce modèle est de diviser récursivement l'ensemble de données d'apprentissage en sous-ensembles plus homogènes basés sur des tests effectués sur les attributs. Un nœud de l'arbre correspond à un test sur la valeur d'un ou plusieurs attributs, tandis qu'une feuille est associée à une valeur de la classe cible. Pour construire l'arbre, l'algorithme cherche à chaque étape la variable qui divise le mieux les exemples en maximisant le gain d'information, basé sur la réduction de l'entropie.

Régression Logistique (Logistic Regression)
La régression logistique est une méthode discriminative qui tente d'apprendre directement une fonction liant les entrées aux classes. Contrairement à la régression linéaire qui prédit une valeur réelle, ce modèle généralisé utilise une fonction d'activation non-linéaire, la sigmoïde, pour projeter les sorties dans l'intervalle [0,1], ce qui permet de les interpréter comme des probabilités a posteriori. La frontière de décision reste linéaire, c'est-à-dire un hyperplan séparant l'espace des données. Pour gérer notre problème multi-classes ($K > 2$), le modèle utilise une approche combinatoire ou une fonction Softmax pour assigner une entrée x à la classe maximisant la probabilité.

Machine à Vecteurs de Support (SVM)
La Machine à Vecteurs de Support (SVM) est un classificateur linéaire qui cherche à définir une frontière de décision optimale sous la forme d'un hyperplan. L'objectif principal de l'algorithme est de maximiser la marge, définie comme la distance perpendiculaire entre la frontière de décision et les données les plus proches de chaque classe, appelées vecteurs de support. Pour traiter les cas où les données ne sont pas linéairement séparables, comme c'est souvent le cas avec les images, nous avons utilisé l'astuce du noyau ("kernel trick"). Cette méthode permet de projeter implicitement les données dans un espace de dimension supérieure où la séparation linéaire devient possible, sans avoir à calculer explicitement les transformations. Nous avons opté pour un noyau Gaussien (RBF) pour sa capacité à modéliser des frontières complexes.

3.2 Méthodes d'ensemble

Forêt Aléatoire (Random Forest)
Afin de pallier les limites de l'arbre de décision unique, nous avons utilisé une forêt aléatoire. Cette méthode d'ensemble construit une multitude d'arbres de décision lors de l'entraînement. Chaque arbre est construit selon les principes d'entropie et de gain d'information vus précédemment, mais sur des sous-ensembles de données différents. Le résultat final est obtenu par vote majoritaire des classes prédites par les arbres individuels, ce qui permet généralement de réduire la variance et d'améliorer la généralisation par rapport à un arbre unique.

Gradient Boosting sur histogrammes (HistGradientBoosting)
Nous avons également implémenté une approche de boosting via l'algorithme HistGradientBoostingClassifier de Scikit-learn. Cette méthode, inspirée de LightGBM, est une variante du Gradient Boosting optimisée pour les grands jeux de données. Elle discretise les données d'entrée en histogrammes de nombres entiers, ce qui accélère considérablement l'entraînement. Comme pour une forêt aléatoire, le modèle est construit séquentiellement : chaque nouvel arbre tente de corriger les erreurs (résidus) des arbres précédents.

3.3 Apprentissage profond

Réseau de Neurones Convolutif (CNN)
Enfin, nous avons implémenté un réseau de neurones convolutif (CNN) profond. L'architecture choisie suit une structure de type VGG simplifiée, composée de trois blocs de convolution successifs. Chaque bloc contient deux couches de convolution, suivies d'une normalisation par lots (Batch Normalization) pour stabiliser l'apprentissage, d'une fonction d'activation ReLU, d'un Max Pooling pour réduire la dimension spatiale, et d'un Dropout progressif (0.3, 0.4, 0.5) pour limiter le sur-apprentissage. Le réseau se termine par des couches dense (Fully Connected) pour effectuer la classification finale.

4 Protocole Expérimental

4.1 Prétraitement des données

Le jeu de données CIFAR-10 brut est constitué de valeurs de pixels entiers compris entre 0 et 255. Avant toute phase d'apprentissage, nous avons appliqué les transformations suivantes via notre classe `DataHandler` :

— Normalisation : Toutes les valeurs de pixels ont été divisées par 255.0 afin de les ramener dans l'intervalle $[0, 1]$. Cette étape est essentielle, particulièrement pour le Réseau de Neurones Convolutif (CNN) et le SVM, car elle facilite la convergence des algorithmes d'optimisation (comme la descente de gradient).

— Mise en forme (Reshaping) :

- Pour les modèles classiques (Arbre de décision, Forêt Aléatoire, GBM), les images ont été "aplaties" (flatten) en vecteurs unidimensionnels de taille $32 \times 32 \times 3 = 3072$.
- Pour le CNN, nous avons conservé la structure spatiale tridimensionnelle $(3 \times 32 \times 32)$ nécessaire aux opérations de convolution.

— Réduction de dimension (ACP) : Pour la Régression Logistique et le SVM, travailler directement sur 3072 dimensions s'est avéré coûteux en temps de calcul. Nous avons donc intégré une Analyse en Composantes Principales (PCA) dans le pipeline, en conservant 95% de la variance expliquée. Cela permet de réduire le bruit et d'accélérer l'entraînement sans perte significative d'information.

4.2 Stratégie de partitionnement

Afin d'éviter le sur-apprentissage (overfitting) et d'évaluer honnêtement nos modèles, nous avons respecté une stricte séparation des données :

1. Ensemble d'entraînement (Training set) : 42 000 images utilisées pour l'ajustement des poids des modèles.
2. Ensemble de validation (Validation set) : 8 000 images extraites du jeu d'entraînement initial. Cet ensemble a servi à monitorer la performance en cours d'apprentissage (notamment pour le CNN et l'arrêt précoce) et à ajuster certains hyper-paramètres.
3. Ensemble de test (Test set) : 10 000 images, strictement isolées jusqu'à la phase finale. Aucun modèle n'a eu accès à ces données durant l'entraînement. C'est sur cet ensemble que sont calculés les scores finaux présentés dans ce rapport.

4.3 Validation croisée et hyper-paramètres

Pour assurer la robustesse de nos résultats sur les modèles classiques, nous avons utilisé la validation croisée (*k-fold cross-validation* avec k=3 ou k=5) via la bibliothèque `scikit-learn`. Cela nous a permis de vérifier que nos modèles ne dépendaient pas d'un découpage particulier des données.

Concernant les hyper-paramètres, nous avons procédé par une recherche manuelle en nous aidant des autres solutions présentes sur le web. En testant itérativement différentes configurations et en observant l'évolution du score sur l'ensemble de validation, nous avons sélectionné les valeurs offrant le meilleur compromis entre coûts de calculs et performance :

— Arbres de décision : Nous avons limité la profondeur (`max_depth=30`) et imposé un nombre minimum d'échantillons par feuille. De plus, nous avons restreint le nombre de caractéristiques évaluées à chaque nœud à la racine carrée du total (`max_features='sqrt'`).
Ces restrictions visent à élaguer l'arbre pour réduire le sur-apprentissage, au prix d'une performance individuelle plus faible (28%) due à un biais plus élevé.

— Forêt Aléatoire : Nous avons utilisé 100 estimateurs (`n_estimators=100`) avec une parallélisation complète (`n_jobs=-1`).
Ce nombre d'arbres permet de stabiliser les prédictions en réduisant la variance par moyennage, tandis que l'utilisation de tous les cœurs CPU optimise le temps de calcul.

— Régression Logistique : Nous avons opté pour le solveur `lbfgs`, bien adapté aux problèmes multiclasses, et augmenté le nombre d'itérations (`max_iter=1000`).
Cette augmentation était nécessaire pour garantir la convergence de l'algorithme de descente de gradient sur des données de haute dimension (pixels aplatis).

— SVM : Nous avons utilisé un noyau RBF (`kernel='rbf'`) avec un paramètre de régularisation standard ($C = 1.0$).
Le choix du noyau gaussien (RBF) est crucial ici pour gérer les frontières de décision non linéaires inhérentes à la classification d'images complexes.

— GBM : Nous avons configuré un taux d'apprentissage de 0.1 (`learning_rate=0.1`) avec 100 itérations et limité la complexité des arbres via un nombre maximum de feuilles (`max_leaf_nodes=31`).
Ces paramètres contrôlent la vitesse de correction des erreurs et la complexité du modèle, offrant un bon équilibre pour éviter de mémoriser le bruit (overfitting).

— CNN : L'entraînement a été réalisé sur 50 époques avec un optimiseur Adam ($lr = 0.001$) et un ordonnanceur (`ReduceLROnPlateau`). Nous avons également intégré un mécanisme d'arrêt précoce (*Early Stopping*) avec une patience de 5 époques.
Cette stratégie permet d'adapter la vitesse d'apprentissage en fonction de la stagnation de la perte et d'arrêter le processus dès que le modèle commence à sur-apprendre sur les données de validation.

5 Résultats

5.1 Comparatif global des performances

Le tableau ci-dessous résume les métriques principales pour chaque classificateur. Le temps d'entraînement inclut la validation croisée pour les modèles classiques.

Modèle	Accuracy	F1-Score (Macro)	Précision (Macro)	Temps d'entraînement
Arbre de Décision	0.2801	0.2804	0.2817	2s
Régression Logistique	0.4045	0.4007	0.3996	30s
Forêt Aléatoire	0.4642	0.4605	0.4596	17s
GBM (HistGradient)	0.5292	0.5269	0.5264	3min 35s
SVM (RBF)	0.5359	0.5343	0.5351	14min 40s
CNN (PyTorch)	**0.7479**	**0.7443**	**0.7479**	2min 8s (sur GPU)

TABLE 1 – Comparaison des performances des modèles sur l'ensemble de test

5.2 Analyse comparative des métriques

La figure 3 et la table 1 mettent en évidence une hiérarchie claire des performances qui se maintient à travers toutes les métriques. On observe une forte corrélation entre l'Accuracy et le F1-score pour tous les modèles. Le graphique confirme visuellement le saut de performance entre les méthodes d'apprentissage classiques (qui plafonnent autour de 53% avec le SVM) et l'approche par apprentissage profond (CNN à 74%). Ceci valide la pertinence des réseaux de neurones convolutifs pour ce type de données non structurées.

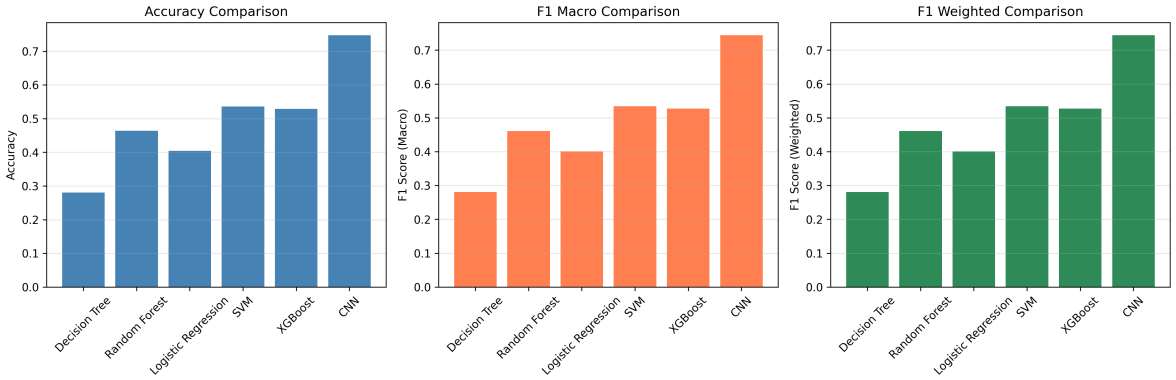


FIGURE 3 – Comparaison des résultats des modèles

5.3 Analyse des matrices de confusion

Pour mieux comprendre les erreurs, nous avons généré des matrices de confusion pour chaque modèle. Voici celles du modèle le moins performant (Arbre de Décision) et du meilleur modèle (CNN).

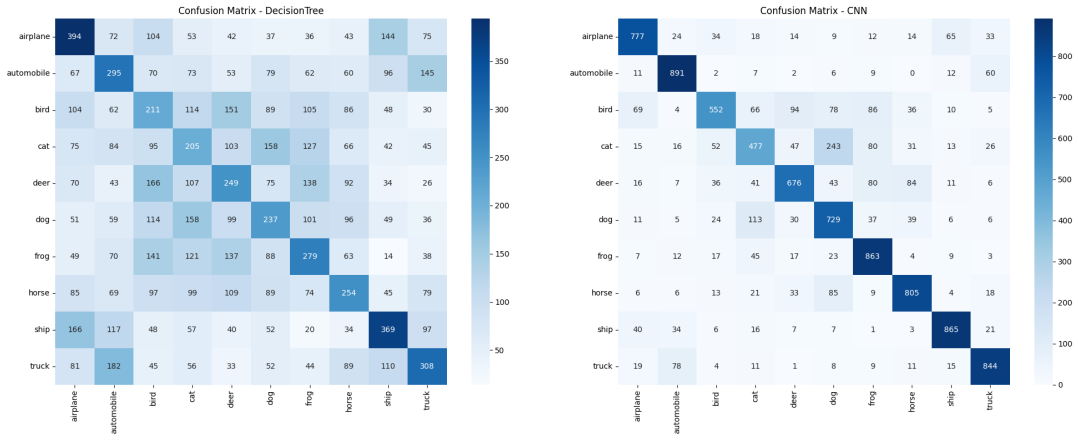


FIGURE 4 – Matrices de confusion : Arbre de Décision (gauche) vs CNN (droite)

5.4 Courbes d'apprentissage du CNN

Afin de vérifier la stabilité de l'entraînement de notre réseau de neurones, nous avons tracé l'évolution de la perte (Loss) et de la précision (Accuracy) au fil des 50 époques.

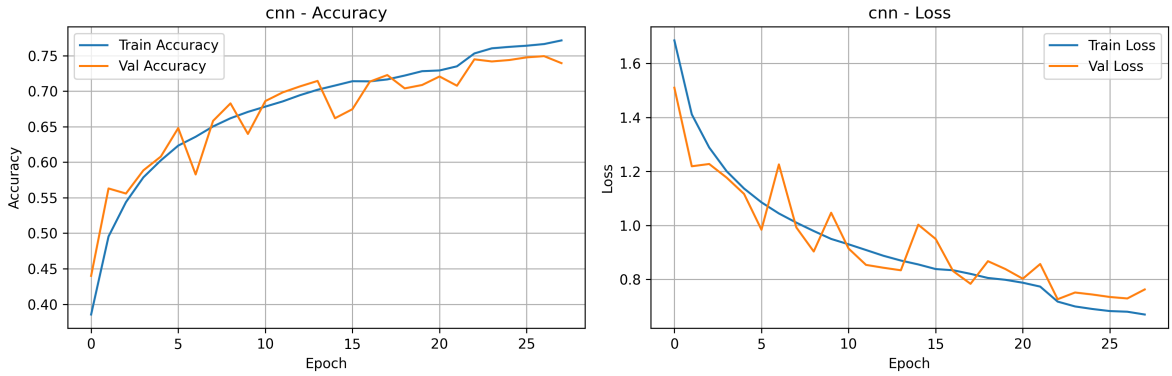


FIGURE 5 – Évolution de la perte et de la précision du CNN (Train vs Validation)

L'analyse des graphiques révèle une divergence progressive entre les courbes d'entraînement et de validation aux alentours de la 20ème époque. Alors que la performance sur le jeu d'entraînement continue de s'améliorer linéairement, la perte de validation stagne et la précision plafonne, ce qui est caractéristique d'un début de sur-apprentissage (overfitting). Cela indique que le modèle commence à mémoriser les spécificités et le bruit des données d'entraînement au détriment de sa capacité à généraliser sur de nouvelles images.

6 Analyse et Discussion

6.1 La meilleure solution

Le CNN obtient les meilleurs résultats avec une exactitude de 74.79%. Cette différence s'explique par la profondeur du réseau : avec ses 6 couches de convolution réparties en 3 blocs, le modèle est capable d'extraire des caractéristiques hiérarchiques complexes (bords, textures, puis formes complètes) tout en préservant la structure 2D de l'image, là où les modèles classiques "écrasent" l'image en un simple vecteur.

6.2 Compromis temps / performance

Un point intéressant concerne l'efficacité computationnelle des modèles :

— L'Arbre de décision est rapide mais peu performant en raison de la restriction sur les features.

— Le SVM s'avère être le modèle le plus coûteux en ressources.Avec un temps d'entraînement de 14min 40s, il est nettement plus lent que les autres approches. Cette lenteur s'explique par la complexité de l'algorithme et la difficulté de trouver l'hyperplan optimal avec un noyau RBF dans un espace de grande dimension, même après réduction PCA.

— Le GBM, bien que complexe, reste raisonnablement performant en termes de vitesse (3min 35s)

— Le CNN offre le meilleur compromis. Grâce à GPU CUDA, il s'entraîne en seulement 2min 8s. Il est donc non seulement le plus précis, mais aussi l'un des plus rapides à converger vers une solution optimale.

6.3 Analyse des erreurs par classe

En examinant les rapports du F1-score par classe, nous observons des disparités :

— Classes faciles : Les classes "Automobile" (F1=0.85 pour le CNN) et "Ship" (F1=0.86) sont très bien reconnues. Ce sont des objets artificiels avec des formes géométriques distinctes et souvent des fonds spécifiques (ciel/eau pour les bateaux, route pour les voitures).

— Classes difficiles : Les classes animales comme "Cat" (F1=0.51) et "Dog" (F1=0.63) posent problème, même au CNN. La matrice de confusion montre que le modèle confond fréquemment les chats avec les chiens (morphologie similaire) et les oiseaux avec les avions (présence de ciel bleu en fond, formes ailées).

6.4 Limites et perspectives

Bien que le CNN obtienne 74.79%, il montre des signes de stagnation. Les courbes d'apprentissage (figure 5) indiquent que le modèle commence à converger vers la 30ème époque. Pour améliorer ce score et atteindre l'état de l'art (>90%), plusieurs pistes pourraient être explorées :

— Augmentation de données : Appliquer des rotations, zooms ou retournements horizontaux pour rendre le modèle plus robuste aux variations.

— Transfer Learning : Utiliser un réseau pré-entraîné (comme ResNet18) plutôt que notre architecture simple à 3 couches de convolution.

6.5 Comparaison avec la littérature

Pour valider la cohérence de nos résultats, nous les avons confrontés aux performances standards rapportées dans la littérature pour le jeu de données CIFAR-10. Ce dataset est réputé difficile pour les algorithmes classiques car il possède de nombreuses intra-classes d'images.

— Modèles classiques : Les classificateurs linéaires (comme la Régression Logistique) ou les arbres simples peinent à dépasser les 30-40% d'exactitude sur les pixels bruts de CIFAR-10. Notre score de 40% en Régression Logistique est donc conforme aux attentes. Les Forêts Aléatoires et SVM (sans extraction de caractéristiques complexes comme HOG) plafonnent généralement entre 45% et 55% et nos résultats (46% pour RF, 53% pour SVM) se situent parfaitement dans cette fourchette, ce qui confirme la validité de nos résultats.

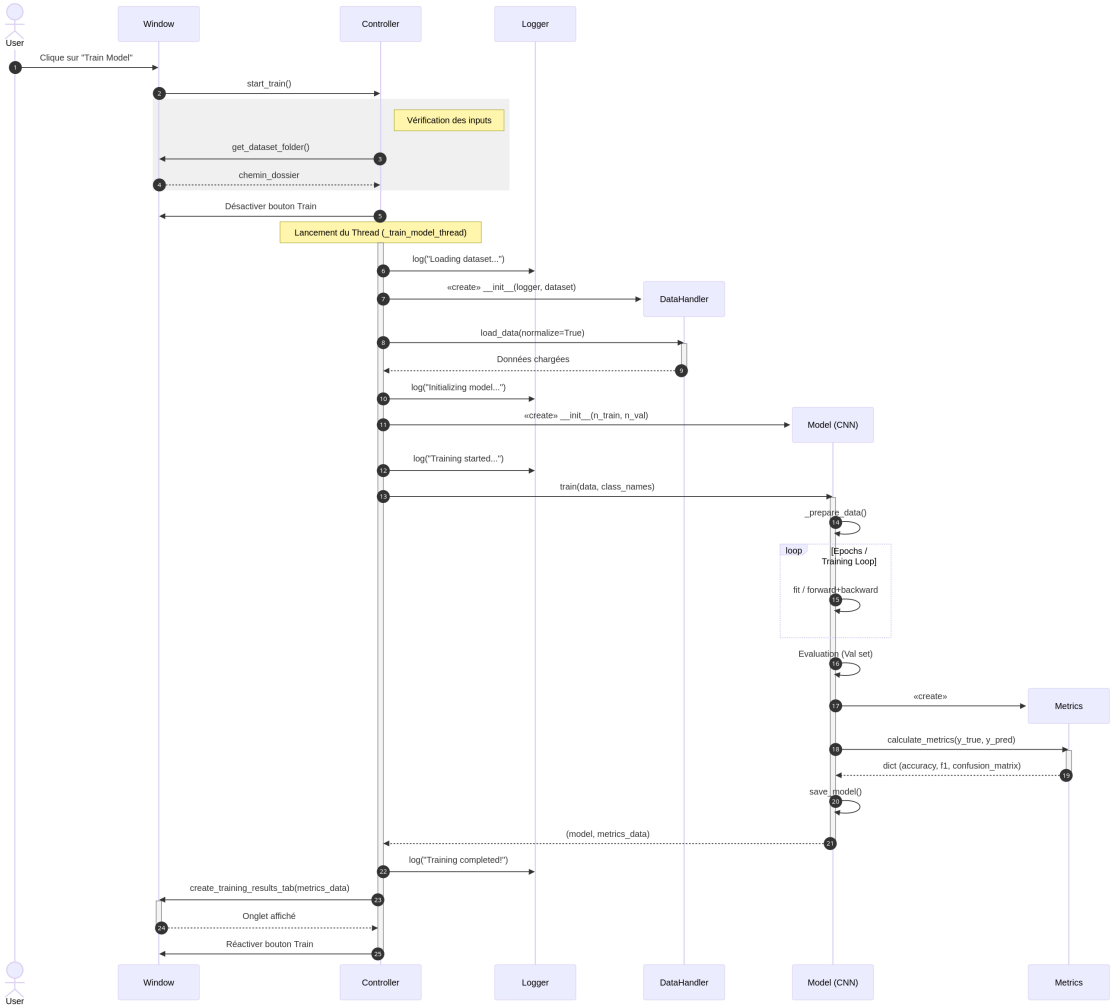
— Réseaux de neurones : Notre architecture CNN personnalisée atteint 74.79%. Des architectures simples similaires comparés à des tutoriels techniques (comme ceux de TensorFlow ou PyTorch) obtiennent typiquement entre 70% et 75% sans augmentation de données.

— État de l'art : Il est important de noter que l'état de l'art actuel sur CIFAR-10 dépasse les 99% d'exactitude. Cependant, ces scores sont atteints par des modèles extrêmement profonds (type ResNet, DenseNet ou Vision Transformers) comportant des millions de paramètres et utilisant des techniques d'augmentation de données agressives (Cutout, Mixup) que nous n'avons pas implémentées ici.

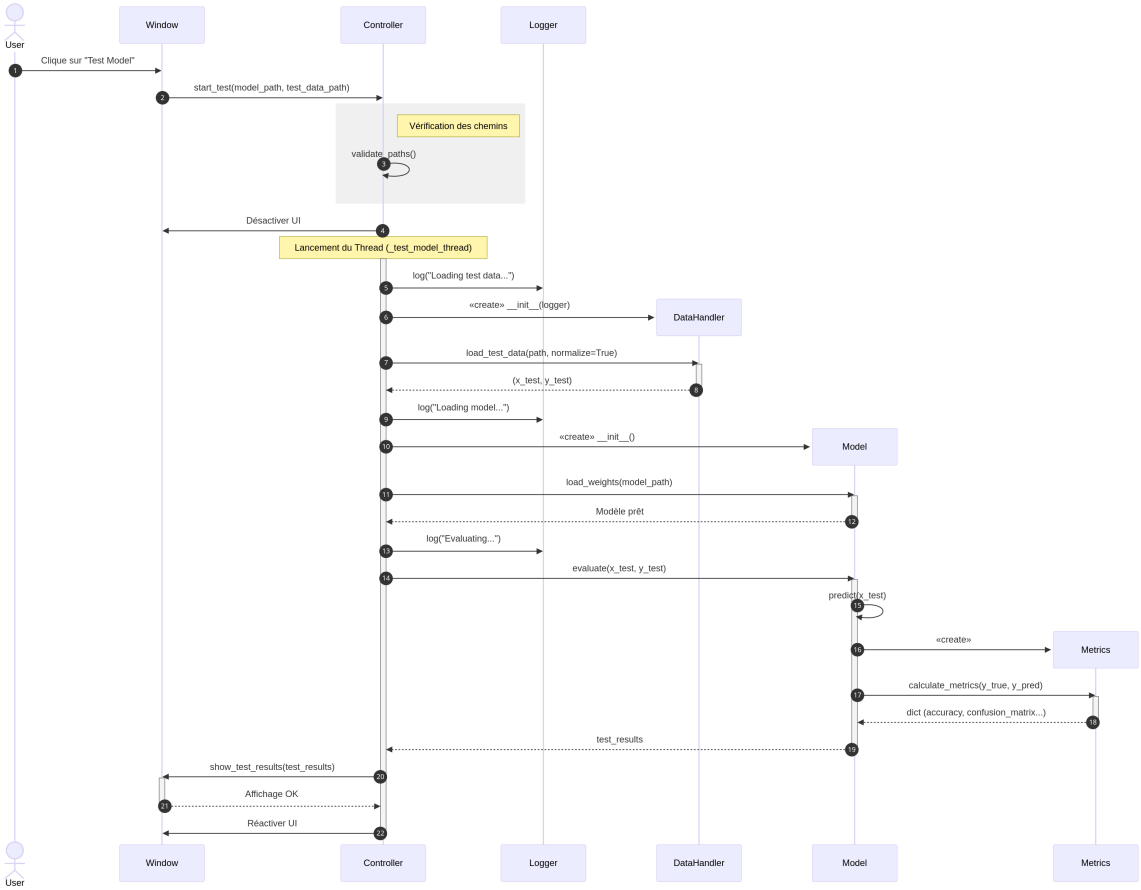
En conclusion, nos modèles se comportent de manière cohérente avec les standards académiques pour des implémentations "from scratch" sur données brutes.

7 Retour au génie logiciel

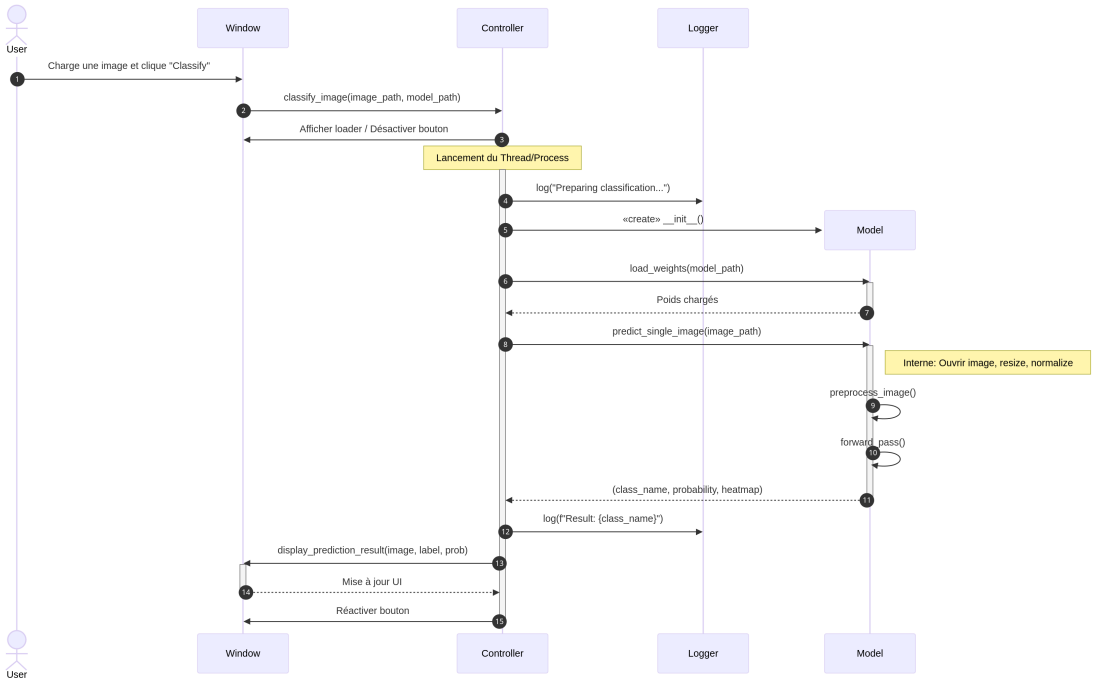
Nous avons généré les diagrammes de séquences et le diagramme de classe final pour nos cas d'utilisations :



Cas d'utilisation : Entraîner



Cas d'utilisation : Tester



Cas d'utilisation : Classifier

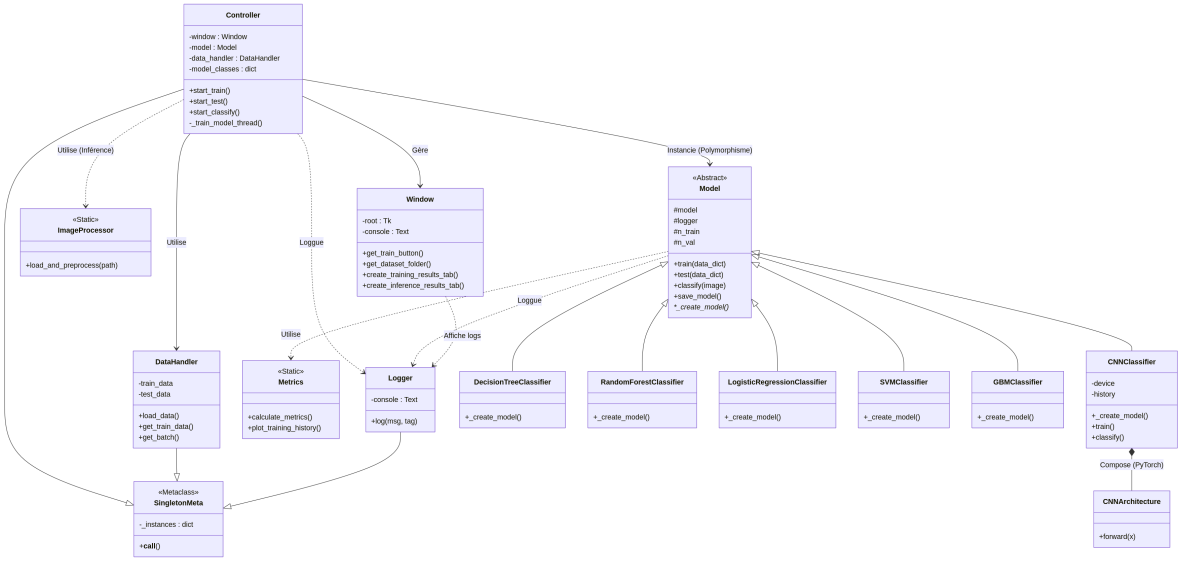


Diagramme de classe final

Nous observons que nous avons réussi à garder la même architecture, évidemment plus détaillée, pour notre notre conception initiale malgrè les difficultés posées par le développement.

8 Conclusion

Ce projet, réalisé dans le cadre du cours IFT712, nous a permis de mettre en œuvre et de comparer six approches distinctes de classification supervisée sur le jeu de données CIFAR-10. Grâce au prétraitement des données, la validation croisée et une séparation stricte des ensembles d'apprentissage et de test, nous avons pu évaluer la pertinence de chaque modèle pour une tâche de vision par ordinateur.

Les résultats obtenus confirment la supériorité des méthodes d'apprentissage profond pour le traitement d'images brutes. Avec une exactitude de 74.79%, notre Réseau de Neurones Convolutif (CNN) surpasse largement les approches classiques. Le GBM (52.92%) et le SVM (53.59%), bien que robustes sur des données tabulaires, peinent à généraliser efficacement sur des vecteurs de pixels aplatis, résultant en des temps de calcul parfois longs (plus de 14 minutes pour le SVM contre environ 2 minutes pour le CNN).

Notre étude présente néanmoins certaines limites qui pourraient servir de pistes d'amélioration pour des travaux futurs :

— Augmentation de données : Nous avons entraîné nos modèles sur les images brutes (simplement normalisées). L'utilisation de techniques d'augmentation (rotations, zooms, symétries) permettrait d'améliorer les résultats du CNN.

— Architecture des modèles : Notre architecture CNN reste relativement simple (6 couches de convolution). L'utilisation de réseaux plus profonds ou de techniques de *Transfer Learning* (en utilisant des modèles pré-entraînés comme ResNet ou VGG) permettrait de dépasser la barre des 90% d'exactitude.

— Optimisation des hyper-paramètres : L'utilisation de méthodes automatisées comme le *Grid Search* ou l'*Optuna* sur une plage de valeurs plus large pourrait permettre d'affiner encore les performances des modèles classiques comme le SVM ou la Forêt Aléatoire.