Classification CIFAR-10

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 CNNClassifier Namespace Reference

Classes

- class CNNArchitecture

  Architecture du réseau de neurones convolutif (CNN) définie avec PyTorch.
- class CNNClassifier

5.2 Controller Namespace Reference

Classes

- class Controller

5.3 DataHandler Namespace Reference

Classes

- class DataHandler

5.4 DecisionTreeClassifier Namespace Reference

Classes

- class DecisionTreeClassifier

5.5 functions Namespace Reference

Functions

- convert_time (secondes)

5.5.1 Function Documentation

5.5.1.1 convert_time()

```
functions.convert_time (
              secondes )
```

Definition at line 1 of file functions.py.

5.6 GBMClassifier Namespace Reference

Classes

- class GBMClassifier

5.7 ImageProcessor Namespace Reference

Classes

- class ImageProcessor

5.8 Logger Namespace Reference

Classes

- class Logger

Functions

- get_log_color (tag)

  Retourne la couleur Tkinter associée à un tag de log donné.

5.8.1 Function Documentation

5.8.1.1 get_log_color()

```
Logger.get_log_color (
              tag )
```

Retourne la couleur Tkinter associée à un tag de log donné.

Parameters

tag	(str) Le niveau ou type de log (ex: "WARNING", "ERROR", "SUCCESS", "RESULT").

Returns

(str) Le nom de la couleur correspondante (ex: "orange", "red", "green").

Definition at line 11 of file Logger.py.

5.9 LogisticRegressionClassifier Namespace Reference

Classes

- class LogisticRegressionClassifier

5.10 main Namespace Reference

Variables

- window = Window()
- Controller = Controller(window)

5.10.1 Variable Documentation

5.10.1.1 Controller

```
main.Controller = Controller(window)
```

Definition at line 14 of file main.py.

5.10.1.2 window

```
main.window = Window()
```

Definition at line 13 of file main.py.

5.11 Metrics Namespace Reference

Classes

- class Metrics

5.12 Model Namespace Reference

Classes

- class Model

5.13 RandomForestClassifier Namespace Reference

Classes

- class RandomForestClassifier

5.14 SingletonMeta Namespace Reference

Classes

- class SingletonMeta

5.15 SVMClassifier Namespace Reference

Classes

- class SVMClassifier

5.16 TqdmToLogger Namespace Reference

Classes

- class TqdmToLogger

5.17 Window Namespace Reference

Classes

- class Window

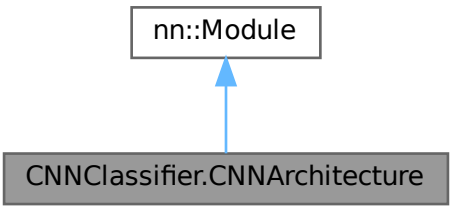
Chapter 6

Class Documentation

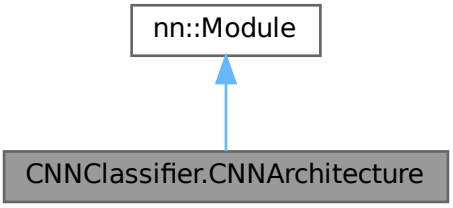
6.1 CNNClassifier.CNNArchitecture Class Reference

Architecture du réseau de neurones convolutif (CNN) définie avec PyTorch.

Inheritance diagram for CNNClassifier.CNNArchitecture:



Collaboration diagram for CNNClassifier.CNNArchitecture:



Public Member Functions

- __init__ (self)
- forward (self, x)

  Définit la passe avant (forward pass) du réseau.

Public Attributes

- conv1_1
- bn1_1
- conv1_2
- bn1_2
- pool1
- dropout1
- conv2_1
- bn2_1
- conv2_2
- bn2_2
- pool2
- dropout2
- conv3_1
- bn3_1
- conv3_2
- bn3_2
- pool3
- dropout3
- flatten
- fc1
- bn_fc
- dropout_fc
- fc2
- relu

6.1.1 Detailed Description

Architecture du réseau de neurones convolutif (CNN) définie avec PyTorch.

Comprend 3 blocs de convolution (Conv2d -> BatchNorm -> ReLU -> MaxPool -> Dropout) suivis de couches dense (Fully Connected). Conçu pour des entrées image 32x32x3.

Definition at line 21 of file CNNClassifier.py.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
CNNClassifier.CNNArchitecture.__init__ (
              self )
```

Definition at line 23 of file CNNClassifier.py.

6.1.3 Member Function Documentation

6.1.3.1 forward()

```
CNNClassifier.CNNArchitecture.forward (
              self,
              x )
```

Définit la passe avant (forward pass) du réseau.

Parameters

x	(torch.Tensor) Tenseur d'entrée (Batch, 3, 32, 32).

Returns

torch.Tensor Logits de sortie (Batch, 10).

Definition at line 57 of file CNNClassifier.py.

6.1.4 Member Data Documentation

6.1.4.1 bn1_1

```
CNNClassifier.CNNArchitecture.bn1_1
```

Definition at line 26 of file CNNClassifier.py.

6.1.4.2 bn1_2

```
CNNClassifier.CNNArchitecture.bn1_2
```

Definition at line 28 of file CNNClassifier.py.

6.1.4.3 bn2_1

```
CNNClassifier.CNNArchitecture.bn2_1
```

Definition at line 33 of file CNNClassifier.py.

6.1.4.4 bn2_2

```
CNNClassifier.CNNArchitecture.bn2_2
```

Definition at line 35 of file CNNClassifier.py.

6.1.4.5 bn3_1

`CNNClassifier.CNNArchitecture.bn3_1`

Definition at line 40 of file CNNClassifier.py.

6.1.4.6 bn3_2

`CNNClassifier.CNNArchitecture.bn3_2`

Definition at line 42 of file CNNClassifier.py.

6.1.4.7 bn_fc

`CNNClassifier.CNNArchitecture.bn_fc`

Definition at line 48 of file CNNClassifier.py.

6.1.4.8 conv1_1

`CNNClassifier.CNNArchitecture.conv1_1`

Definition at line 25 of file CNNClassifier.py.

6.1.4.9 conv1_2

`CNNClassifier.CNNArchitecture.conv1_2`

Definition at line 27 of file CNNClassifier.py.

6.1.4.10 conv2_1

`CNNClassifier.CNNArchitecture.conv2_1`

Definition at line 32 of file CNNClassifier.py.

6.1.4.11 conv2_2

`CNNClassifier.CNNArchitecture.conv2_2`

Definition at line 34 of file CNNClassifier.py.

6.1.4.12 conv3_1

`CNNClassifier.CNNArchitecture.conv3_1`

Definition at line 39 of file CNNClassifier.py.

6.1.4.13 conv3_2

`CNNClassifier.CNNArchitecture.conv3_2`

Definition at line 41 of file CNNClassifier.py.

6.1.4.14 dropout1

`CNNClassifier.CNNArchitecture.dropout1`

Definition at line 30 of file CNNClassifier.py.

6.1.4.15 dropout2

`CNNClassifier.CNNArchitecture.dropout2`

Definition at line 37 of file CNNClassifier.py.

6.1.4.16 dropout3

`CNNClassifier.CNNArchitecture.dropout3`

Definition at line 44 of file CNNClassifier.py.

6.1.4.17 dropout_fc

`CNNClassifier.CNNArchitecture.dropout_fc`

Definition at line 49 of file CNNClassifier.py.

6.1.4.18 fc1

`CNNClassifier.CNNArchitecture.fc1`

Definition at line 47 of file CNNClassifier.py.

6.1.4.19 fc2

`CNNClassifier.CNNArchitecture.fc2`

Definition at line 50 of file CNNClassifier.py.

6.1.4.20 flatten

`CNNClassifier.CNNArchitecture.flatten`

Definition at line 46 of file CNNClassifier.py.

6.1.4.21 pool1

`CNNClassifier.CNNArchitecture.pool1`

Definition at line 29 of file CNNClassifier.py.

6.1.4.22 pool2

`CNNClassifier.CNNArchitecture.pool2`

Definition at line 36 of file CNNClassifier.py.

6.1.4.23 pool3

`CNNClassifier.CNNArchitecture.pool3`

Definition at line 43 of file CNNClassifier.py.

6.1.4.24 relu

`CNNClassifier.CNNArchitecture.relu`

Definition at line 51 of file CNNClassifier.py.

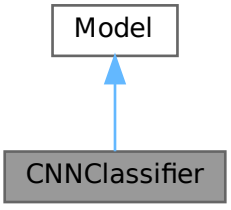
The documentation for this class was generated from the following file:

- src/CNNClassifier.py

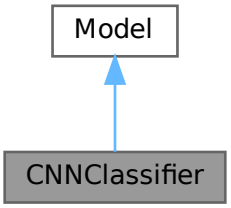
6.2 CNNClassifier Class Reference

Adaptateur pour le modèle CNN PyTorch intégrant la structure de la classe Model.

Inheritance diagram for CNNClassifier:



Collaboration diagram for CNNClassifier:



6.2.1 Detailed Description

Adaptateur pour le modèle CNN PyTorch intégrant la structure de la classe Model.

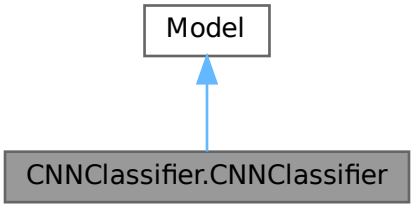
Gère spécifiquement les tenseurs PyTorch, l'utilisation du GPU (CUDA), la boucle d'entraînement manuelle et l'Early Stopping.

The documentation for this class was generated from the following file:

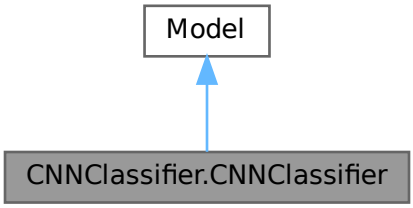
- src/CNNClassifier.py

6.3 CNNClassifier.CNNClassifier Class Reference

Inheritance diagram for CNNClassifier.CNNClassifier:



Collaboration diagram for CNNClassifier.CNNClassifier:



Public Member Functions

- __init__ (self, logger, n_train, n_val)
- tuple train (self, dict data_dict, class_names=None)

  Surcharge de la méthode d'entraînement pour PyTorch.
- dict test (self, dict data_dict, class_names=None)

  Surcharge de la méthode de test pour PyTorch.
- int classify (self, np.ndarray image)

  Inférence sur une seule image avec PyTorch.
- None save_model (self)

  Sauvegarde les poids du modèle (state_dict) via torch.save.
- None load_model (self)

  Charge les poids du modèle (state_dict) via torch.load.

Public Attributes

- device
- model_path
- history
- tqdm_out
- model
- logger

Protected Member Functions

- _create_model (self, ∗∗kwargs)
- _predict_batch (self, X_tensor, y_tensor)

  Méthode utilitaire interne pour prédire sur un grand jeu de données par lots.

6.3.1 Detailed Description

Definition at line 86 of file CNNClassifier.py.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
CNNClassifier.CNNClassifier.__init__ (
              self,
              logger,
              n_train,
              n_val )
```

Definition at line 87 of file CNNClassifier.py.

6.3.3 Member Function Documentation

6.3.3.1 _create_model()

```
CNNClassifier.CNNClassifier._create_model (
              self,
             ** kwargs )  [protected]
```

Definition at line 95 of file CNNClassifier.py.

6.3.3.2 _predict_batch()

```
CNNClassifier.CNNClassifier._predict_batch (
              self,
              X_tensor,
              y_tensor )  [protected]
```

Méthode utilitaire interne pour prédire sur un grand jeu de données par lots.

Parameters

X_tensor	(torch.Tensor) Données d'entrée.
y_tensor	(torch.Tensor) Labels (utilisés uniquement pour créer le Dataset compatible).

Returns

np.ndarray Tableau numpy des classes prédites.

Definition at line 327 of file CNNClassifier.py.

6.3.3.3 classify()

```
int CNNClassifier.CNNClassifier.classify (
              self,
             np.ndarray image )
```

Inférence sur une seule image avec PyTorch.

Gère le redimensionnement (32, 32, 3) et le passage sur le device (CPU/GPU).

Parameters

image	(np.ndarray) Image brute.

Returns

int Indice de la classe prédite.

Definition at line 356 of file CNNClassifier.py.

6.3.3.4 load_model()

```
None CNNClassifier.CNNClassifier.load_model (
              self )
```

Charge les poids du modèle (state_dict) via torch.load.

Assure le mapping correct sur le device (CPU/GPU).

Definition at line 382 of file CNNClassifier.py.

6.3.3.5 save_model()

```
None CNNClassifier.CNNClassifier.save_model (
              self )
```

Sauvegarde les poids du modèle (state_dict) via torch.save.

Definition at line 374 of file CNNClassifier.py.

6.3.3.6 test()

```
dict CNNClassifier.CNNClassifier.test (
              self,
             dict data_dict,
              class_names = None )
```

Surcharge de la méthode de test pour PyTorch.

Convertit les données de test en tenseurs et effectue l'inférence par batch pour éviter les erreurs de mémoire (OOM).

Parameters

data_dict	(dict) Données de test.
class_names	(list, optional) Noms des classes.

Returns

dict Métriques de performance sur le jeu de test.

Definition at line 289 of file CNNClassifier.py.

6.3.3.7 train()

```
tuple CNNClassifier.CNNClassifier.train (
              self,
             dict data_dict,
              class_names = None )
```

Surcharge de la méthode d'entraînement pour PyTorch.

Gère :

- La conversion des données Numpy vers des Tenseurs PyTorch (permute channels).
- La création des DataLoaders.
- La boucle d'entraînement sur 50 époques avec barre de progression (tqdm).
- L'optimiseur Adam et le Scheduler ReduceLROnPlateau.
- L'Early Stopping basé sur la `val_loss`.
- La sauvegarde de l'historique d'apprentissage.

Parameters

data_dict	(dict) Données d'entraînement.
class_names	(list, optional) Noms des classes.

Returns

tuple (model, metrics) Le modèle PyTorch (nn.Module) et les métriques finales.

Definition at line 110 of file CNNClassifier.py.

6.3.4 Member Data Documentation

6.3.4.1 device

```
CNNClassifier.CNNClassifier.device
```

Definition at line 89 of file CNNClassifier.py.

6.3.4.2 history

```
CNNClassifier.CNNClassifier.history
```

Definition at line 92 of file CNNClassifier.py.

6.3.4.3 logger

`CNNClassifier.CNNClassifier.logger`

Definition at line 264 of file CNNClassifier.py.

6.3.4.4 model

`CNNClassifier.CNNClassifier.model`

Definition at line 133 of file CNNClassifier.py.

6.3.4.5 model_path

`CNNClassifier.CNNClassifier.model_path`

Definition at line 90 of file CNNClassifier.py.

6.3.4.6 tqdm_out

`CNNClassifier.CNNClassifier.tqdm_out`

Definition at line 93 of file CNNClassifier.py.

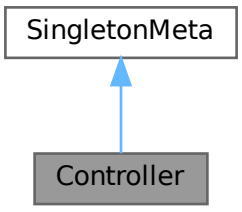
The documentation for this class was generated from the following file:

- src/CNNClassifier.py

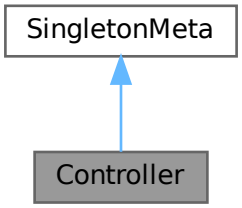
6.4 Controller Class Reference

Contrôleur principal de l'application (Pattern MVC).

Inheritance diagram for Controller:



Collaboration diagram for Controller:



6.4.1 Detailed Description

Contrôleur principal de l'application (Pattern MVC).

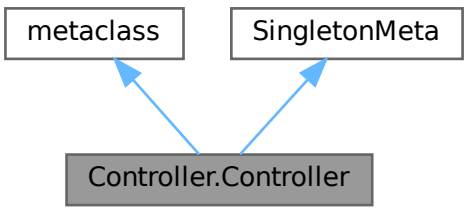
Cette classe fait le lien entre l'interface graphique (Window) et la logique métier (Modèles, DataHandler). Elle gère les événements utilisateur, valide les entrées, et exécute les tâches lourdes (entraînement, test, inférence) dans des threads séparés pour maintenir la réactivité de l'interface.

The documentation for this class was generated from the following file:

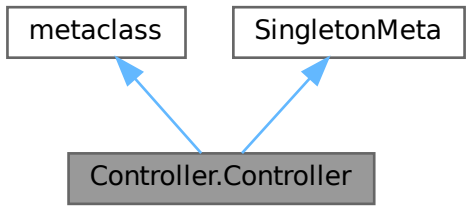
- Controller.py

6.5 Controller.Controller Class Reference

Inheritance diagram for Controller.Controller:



Collaboration diagram for Controller.Controller:



Public Member Functions

- __init__ (self, window)

  Constructeur de la classe Controller.
- start_train (self)

  Déclenche le processus d'entraînement.
- start_test (self)

  Déclenche le processus de test.
- start_classify (self)

  Déclenche le processus d'inférence (classification).
- check_inputs (self, action)

  Valide les entrées du formulaire de l'interface graphique.

Public Attributes

- window
- logger
- dataset
- train_data
- val_data
- data_handler
- model
- training_counter
- inference_path
- model_classes

Protected Member Functions

- _train_model_thread (self, model_name)

  Logique métier de l'entraînement (exécutée dans un thread).
- _display_training_results (self, model_name, metrics_data)

  Met à jour l'interface graphique avec les résultats de l'entraînement.
- _reset_train_button (self)

  Réactive le bouton d'entraînement à la fin du thread.
- _test_model_thread (self, model_name)

Logique métier du test (exécutée dans un thread).

- _display_test_results (self, model_name, metrics_data)

  Affiche les résultats du test dans l'interface graphique.
- _reset_test_button (self)

  Réactive le bouton de test à la fin du thread.
- _classify_model_thread (self, model_name)

  Logique métier de l'inférence (exécutée dans un thread).
- _display_classification_results (self, results_data)

  Affiche les résultats de classification dans l'interface.
- _reset_classify_button (self)

  Réactive le bouton d'inférence à la fin du thread.

Protected Attributes

- _display_training_results
- _reset_train_button
- _display_test_results
- _reset_test_button
- _display_classification_results
- _reset_classify_button

6.5.1 Detailed Description

Definition at line 24 of file Controller.py.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
Controller.Controller.__init__ (
              self,
              window )
```

Constructeur de la classe Controller.

Initialise le logger, configure les callbacks des boutons de l'interface graphique et initialise le dictionnaire des classes de modèles disponibles.

Parameters

window	(Window) L'instance de la fenêtre principale de l'application.

Definition at line 30 of file Controller.py.

6.5.3 Member Function Documentation

6.5.3.1 _classify_model_thread()

```
Controller.Controller._classify_model_thread (
```

```
                self,
                model_name )  [protected]
```

Logique métier de l'inférence (exécutée dans un thread).

Charge le modèle, prépare les images (fichier unique ou dossier) et exécute la prédiction.

Parameters

model_name	(str) Le nom du modèle à utiliser.

Definition at line 256 of file Controller.py.

6.5.3.2 _display_classification_results()

```
Controller.Controller._display_classification_results (
                self,
                results_data )  [protected]
```

Affiche les résultats de classification dans l'interface.

Parameters

results_data	(list) Liste de tuples contenant les noms de fichiers, prédictions et images.

Definition at line 338 of file Controller.py.

6.5.3.3 _display_test_results()

```
Controller.Controller._display_test_results (
                self,
                model_name,
                metrics_data )  [protected]
```

Affiche les résultats du test dans l'interface graphique.

Parameters

model_name	(str) Nom du modèle testé.
metrics_data	(dict) Dictionnaire des métriques de test.

Definition at line 214 of file Controller.py.

6.5.3.4 _display_training_results()

```
Controller.Controller._display_training_results (
                self,
```

```
                model_name,
                metrics_data )  [protected]
```

Met à jour l'interface graphique avec les résultats de l'entraînement.

Crée un nouvel onglet dans la fenêtre via `window.create_training_results_tab`.

Parameters

model_name	(str) Le nom du modèle entraîné.
metrics_data	(dict) Les métriques et graphiques résultants de l'entraînement.

Definition at line 126 of file Controller.py.

6.5.3.5 _reset_classify_button()

```
Controller.Controller._reset_classify_button (
                self )  [protected]
```

Réactive le bouton d'inférence à la fin du thread.

Definition at line 347 of file Controller.py.

6.5.3.6 _reset_test_button()

```
Controller.Controller._reset_test_button (
                self )  [protected]
```

Réactive le bouton de test à la fin du thread.

Definition at line 230 of file Controller.py.

6.5.3.7 _reset_train_button()

```
Controller.Controller._reset_train_button (
                self )  [protected]
```

Réactive le bouton d'entraînement à la fin du thread.

Definition at line 153 of file Controller.py.

6.5.3.8 _test_model_thread()

```
Controller.Controller._test_model_thread (
                self,
                model_name )  [protected]
```

Logique métier du test (exécutée dans un thread).

Charge le modèle sélectionné (sans l'entraîner) et lance la méthode `test` du modèle.

Parameters

model_name	(str) Le nom du modèle à tester.

Definition at line 179 of file Controller.py.

6.5.3.9 _train_model_thread()

```
Controller.Controller._train_model_thread (
              self,
              model_name )  [protected]
```

Logique métier de l'entraînement (exécutée dans un thread).

Charge les données via DataHandler, instancie le modèle sélectionné, lance l'entraînement et demande l'affichage des résultats une fois terminé.

Parameters

model_name	(str) Le nom du modèle à entraîner.

Definition at line 89 of file Controller.py.

6.5.3.10 check_inputs()

```
Controller.Controller.check_inputs (
              self,
              action )
```

Valide les entrées du formulaire de l'interface graphique.

Parameters

action	(str) L'action demandée : "train", "test" ou "classify".

Returns

(bool) True si toutes les entrées requises sont valides, False sinon.

Definition at line 356 of file Controller.py.

6.5.3.11 start_classify()

```
Controller.Controller.start_classify (
              self )
```

Déclenche le processus d'inférence (classification).

Vérifie les entrées, désactive le bouton d'inférence et lance le thread associé.

Definition at line 236 of file Controller.py.

6.5.3.12 start_test()

```
Controller.Controller.start_test (
              self )
```

Déclenche le processus de test.

Vérifie les entrées, désactive le bouton de test et lance le thread de test.

Definition at line 159 of file Controller.py.

6.5.3.13 start_train()

```
Controller.Controller.start_train (
              self )
```

Déclenche le processus d'entraînement.

Vérifie les entrées utilisateur via check_inputs(), désactive le bouton d'entraînement et lance l'exécution de `train_model_thread` dans un thread séparé.

Definition at line 59 of file Controller.py.

6.5.4 Member Data Documentation

6.5.4.1 _display_classification_results

```
Controller.Controller._display_classification_results  [protected]
```

Definition at line 324 of file Controller.py.

6.5.4.2 _display_test_results

```
Controller.Controller._display_test_results  [protected]
```

Definition at line 199 of file Controller.py.

6.5.4.3 _display_training_results

```
Controller.Controller._display_training_results  [protected]
```

Definition at line 110 of file Controller.py.

6.5.4.4 _reset_classify_button

```
Controller.Controller._reset_classify_button  [protected]
```

Definition at line 333 of file Controller.py.

6.5.4.5 _reset_test_button

`Controller.Controller._reset_test_button [protected]`

Definition at line 208 of file Controller.py.

6.5.4.6 _reset_train_button

`Controller.Controller._reset_train_button [protected]`

Definition at line 119 of file Controller.py.

6.5.4.7 data_handler

`Controller.Controller.data_handler`

Definition at line 41 of file Controller.py.

6.5.4.8 dataset

`Controller.Controller.dataset`

Definition at line 38 of file Controller.py.

6.5.4.9 inference_path

`Controller.Controller.inference_path`

Definition at line 44 of file Controller.py.

6.5.4.10 logger

`Controller.Controller.logger`

Definition at line 32 of file Controller.py.

6.5.4.11 model

`Controller.Controller.model`

Definition at line 42 of file Controller.py.

6.5.4.12 model_classes

`Controller.Controller.model_classes`

Definition at line 46 of file Controller.py.

6.5.4.13 train_data

`Controller.Controller.train_data`

Definition at line 39 of file Controller.py.

6.5.4.14 training_counter

`Controller.Controller.training_counter`

Definition at line 43 of file Controller.py.

6.5.4.15 val_data

`Controller.Controller.val_data`

Definition at line 40 of file Controller.py.

6.5.4.16 window

`Controller.Controller.window`

Definition at line 31 of file Controller.py.

The documentation for this class was generated from the following file:

- Controller.py

6.6 DataHandler Class Reference

Gestionnaire de données pour le chargement et le prétraitement du dataset CIFAR-10.

6.6.1 Detailed Description

Gestionnaire de données pour le chargement et le prétraitement du dataset CIFAR-10.

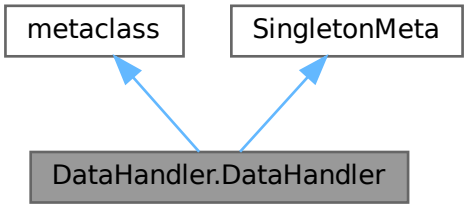
Cette classe s'occupe de lire les fichiers binaires (pickle), de charger les données d'entraînement et de test, de normaliser les pixels et de redimensionner les images selon les besoins des modèles.

The documentation for this class was generated from the following file:

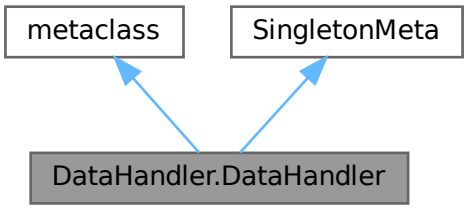
- utils/DataHandler.py

6.7 DataHandler.DataHandler Class Reference

Inheritance diagram for DataHandler.DataHandler:



Collaboration diagram for DataHandler.DataHandler:



Public Member Functions

- __init__ (self, logger, str base_path)

  Initialise le gestionnaire de données.
- dict unpickle (self, str filename)

  Méthode utilitaire interne pour désérialiser un fichier pickle.
- dict load_data (self, bool normalize=True, bool flatten=True)

  Charge l'intégralité du dataset (Train + Test) en mémoire.
- tuple get_train_data (self)

  Retourne les données d'entraînement (X, y).
- tuple get_test_data (self)

  Retourne les données de test (X, y).
- dict get_data_dict (self)

  Retourne le dictionnaire complet des données chargées.
- dict get_subset (self, int n_train=None, int n_test=None)

  Extrait un sous-ensemble des données chargées (utile pour le débogage ou les tests rapides).
- dict get_class_distribution (self, str dataset="train")

Calcule et affiche la distribution des classes dans un jeu de données.

- tuple get_sample (self, int index, str dataset="train")

  Récupère un échantillon unique par son index.
- tuple get_batch (self, list indices, str dataset="train")

  Récupère un lot d'échantillons donnés par une liste d'indices.
- get_class_names (self)

  Retourne la liste des noms lisibles des classes (ex: "Airplane", "Bird"...).
- get_data (self)

  Retourne les données.

Public Attributes

- logger
- base_path
- data
- train_data
- train_labels
- test_data
- test_labels
- class_names

6.7.1 Detailed Description

Definition at line 13 of file DataHandler.py.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 __init__()

```
DataHandler.DataHandler.__init__ (
              self,
              logger,
             str base_path )
```

Initialise le gestionnaire de données.

Parameters

logger	(utils.Logger) logger pour affichage dans la console
base_path	(str) Chemin vers le dossier contenant les fichiers du dataset (data_batch_x).

Exceptions

FileNotFoundError	Si le dossier spécifié n'existe pas.

Definition at line 20 of file DataHandler.py.

6.7.3 Member Function Documentation

6.7.3.1 get_batch()

```
tuple DataHandler.DataHandler.get_batch (
              self,
             list indices,
             str  dataset = "train" )
```

Récupère un lot d'échantillons donnés par une liste d'indices.

Parameters

indices	(list) Liste des indices à récupérer.
dataset	(str) "train" ou "test".

Returns

tuple (images, labels).

Definition at line 211 of file DataHandler.py.

6.7.3.2 get_class_distribution()

```
dict DataHandler.DataHandler.get_class_distribution (
              self,
             str  dataset = "train" )
```

Calcule et affiche la distribution des classes dans un jeu de données.

Parameters

dataset	(str) "train" pour les données d'entraînement, "test" pour les données de test.

Returns

dict Dictionnaire associant chaque nom de classe à son nombre d'occurrences.

Exceptions

ValueError	Si les données ne sont pas chargées.

Definition at line 174 of file DataHandler.py.

6.7.3.3 get_class_names()

```
DataHandler.DataHandler.get_class_names (
              self )
```

Retourne la liste des noms lisibles des classes (ex: "Airplane", "Bird"...).

Returns

list Liste de chaînes de caractères.

Definition at line 223 of file DataHandler.py.

6.7.3.4 get_data()

```
DataHandler.DataHandler.get_data (
              self )
```

Retourne les données.

Returns

dict Dictionnaire complet contenant "train_data", "train_labels", "test_data", "test_labels".

Definition at line 229 of file DataHandler.py.

6.7.3.5 get_data_dict()

```
dict DataHandler.DataHandler.get_data_dict (
              self )
```

Retourne le dictionnaire complet des données chargées.

Returns

dict Dictionnaire avec clés 'train_data', 'train_labels', 'test_data', 'test_labels'.

Exceptions

ValueError	Si load_data() n'a pas été appelé au préalable.

Definition at line 142 of file DataHandler.py.

6.7.3.6 get_sample()

```
tuple DataHandler.DataHandler.get_sample (
              self,
             int index,
             str  dataset = "train" )
```

Récupère un échantillon unique par son index.

Parameters

index	(int) L'index de l'échantillon.
dataset	(str) "train" ou "test".

Returns

tuple (image, label).

Definition at line 197 of file DataHandler.py.

6.7.3.7 get_subset()

```
dict DataHandler.DataHandler.get_subset (
              self,
             int  n_train = None,
             int  n_test = None )
```

Extrait un sous-ensemble des données chargées (utile pour le débogage ou les tests rapides).

Parameters

n_train	(int, optional) Nombre d'échantillons d'entraînement à conserver.
n_test	(int, optional) Nombre d'échantillons de test à conserver.

Returns

dict Dictionnaire contenant les sous-ensembles de données.

Exceptions

ValueError	Si les données n'ont pas encore été chargées.

Definition at line 153 of file DataHandler.py.

6.7.3.8 get_test_data()

```
tuple DataHandler.DataHandler.get_test_data (
              self )
```

Retourne les données de test (X, y).

Returns

tuple (test_data, test_labels).

Exceptions

ValueError	Si load_data() n'a pas été appelé au préalable.

Definition at line 133 of file DataHandler.py.

6.7.3.9 get_train_data()

```
tuple DataHandler.DataHandler.get_train_data (
              self )
```

Retourne les données d'entraînement (X, y).

Returns

tuple (train_data, train_labels).

Exceptions

ValueError	Si load_data() n'a pas été appelé au préalable.

Definition at line 124 of file DataHandler.py.

6.7.3.10 load_data()

```
dict DataHandler.DataHandler.load_data (
              self,
             bool  normalize = True,
             bool  flatten = True )
```

Charge l'intégralité du dataset (Train + Test) en mémoire.

Lit les 5 batches d'entraînement et le batch de test.

Parameters

normalize	(bool) Si True, divise les valeurs des pixels par 255.0 (float). Sinon, garde les valeurs brutes.
flatten	(bool) Si True, retourne les images sous forme de vecteurs (N, 3072). Si False, retourne les images sous forme de tenseurs (N, 3, 32, 32).

Definition at line 67 of file DataHandler.py.

6.7.3.11 unpickle()

```
dict DataHandler.DataHandler.unpickle (
              self,
            str filename )
```

Méthode utilitaire interne pour désérialiser un fichier pickle.

Parameters

filename	(str) Nom du fichier à charger (relatif au base_path).

Returns

dict Le contenu brut du fichier pickle.

Exceptions

FileNotFoundError	Si le fichier n'est pas trouvé.

Definition at line 50 of file DataHandler.py.

6.7.4 Member Data Documentation

6.7.4.1 base_path

`DataHandler.DataHandler.base_path`

Definition at line 22 of file DataHandler.py.

6.7.4.2 class_names

`DataHandler.DataHandler.class_names`

Definition at line 29 of file DataHandler.py.

6.7.4.3 data

`DataHandler.DataHandler.data`

Definition at line 23 of file DataHandler.py.

6.7.4.4 logger

`DataHandler.DataHandler.logger`

Definition at line 21 of file DataHandler.py.

6.7.4.5 test_data

`DataHandler.DataHandler.test_data`

Definition at line 26 of file DataHandler.py.

6.7.4.6 test_labels

`DataHandler.DataHandler.test_labels`

Definition at line 27 of file DataHandler.py.

6.7.4.7 train_data

`DataHandler.DataHandler.train_data`

Definition at line 24 of file DataHandler.py.

6.7.4.8 train_labels

`DataHandler.DataHandler.train_labels`

Definition at line 25 of file DataHandler.py.

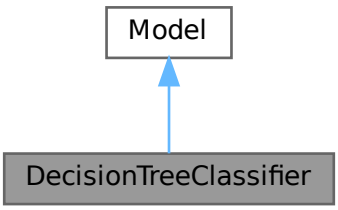
The documentation for this class was generated from the following file:

- utils/DataHandler.py

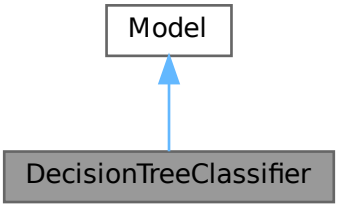
6.8 DecisionTreeClassifier Class Reference

Implémentation d'un classificateur par Arbre de Décision.

Inheritance diagram for DecisionTreeClassifier:



Collaboration diagram for DecisionTreeClassifier:



6.8.1 Detailed Description

Implémentation d'un classificateur par Arbre de Décision.

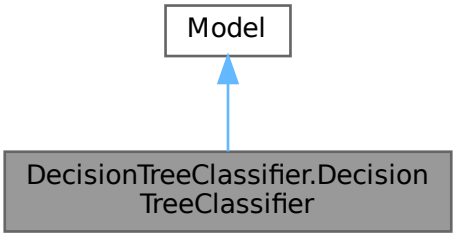
Utilise `sklearn.tree.DecisionTreeClassifier` avec une profondeur maximale de 30 et le critère de Gini.

The documentation for this class was generated from the following file:

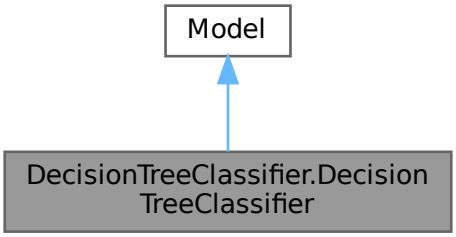
- src/DecisionTreeClassifier.py

6.9 DecisionTreeClassifier.DecisionTreeClassifier Class Reference

Inheritance diagram for DecisionTreeClassifier.DecisionTreeClassifier:



Collaboration diagram for DecisionTreeClassifier.DecisionTreeClassifier:



Public Member Functions

- __init__ (self, logger, n_train, n_val)

Protected Member Functions

- _create_model (self, ∗∗kwargs)

  Instancie l'arbre de décision avec des hyperparamètres pré-définis.

6.9.1 Detailed Description

Definition at line 11 of file DecisionTreeClassifier.py.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 __init__()

```
DecisionTreeClassifier.DecisionTreeClassifier.__init__ (
              self,
              logger,
              n_train,
              n_val )
```

Definition at line 13 of file DecisionTreeClassifier.py.

6.9.3 Member Function Documentation

6.9.3.1 _create_model()

```
DecisionTreeClassifier.DecisionTreeClassifier._create_model (
              self,
             ** kwargs )  [protected]
```

Instancie l'arbre de décision avec des hyperparamètres pré-définis.

Parameters

kwargs	Arguments supplémentaires pour le constructeur DecisionTreeClassifier.

Returns

sklearn.tree.DecisionTreeClassifier Le modèle configuré.

Definition at line 20 of file DecisionTreeClassifier.py.

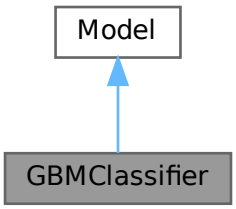
The documentation for this class was generated from the following file:

- src/DecisionTreeClassifier.py

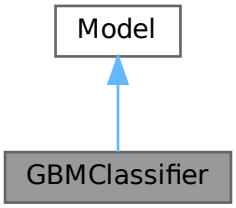
6.10 GBMClassifier Class Reference

Implémentation d'un classificateur Gradient Boosting Machine (Histogram-based).

Inheritance diagram for GBMClassifier:



Collaboration diagram for GBMClassifier:



6.10.1 Detailed Description

Implémentation d'un classificateur Gradient Boosting Machine (Histogram-based).

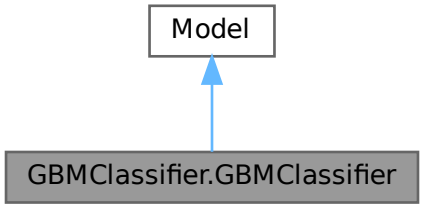
Optimisé pour les grands jeux de données, utilise `HistGradientBoostingClassifier` précédé d'une standardisation des données.

The documentation for this class was generated from the following file:

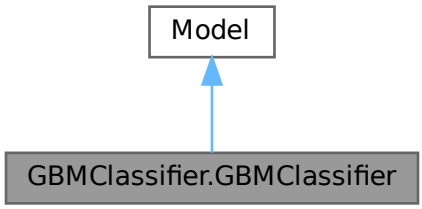
- src/GBMClassifier.py

6.11 GBMClassifier.GBMClassifier Class Reference

Inheritance diagram for GBMClassifier.GBMClassifier:



Collaboration diagram for GBMClassifier.GBMClassifier:



Public Member Functions

- __init__ (self, logger, n_train, n_val)

Protected Member Functions

- _create_model (self, ∗∗kwargs)

  Crée le pipeline GBM.

6.11.1 Detailed Description

Definition at line 13 of file GBMClassifier.py.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 __init__()

```
GBMClassifier.GBMClassifier.__init__ (
              self,
              logger,
              n_train,
              n_val )
```

Definition at line 15 of file GBMClassifier.py.

6.11.3 Member Function Documentation

6.11.3.1 _create_model()

```
GBMClassifier.GBMClassifier._create_model (
              self,
             ** kwargs )  [protected]
```

Crée le pipeline GBM.

Parameters

kwargs	Arguments supplémentaires pour `HistGradientBoostingClassifier`.

Returns

sklearn.pipeline.Pipeline Pipeline configuré (StandardScaler -> HistGradientBoostingClassifier).

Definition at line 22 of file GBMClassifier.py.

The documentation for this class was generated from the following file:

- src/GBMClassifier.py

6.12 ImageProcessor Class Reference

Classe utilitaire pour le chargement et le prétraitement des images d'inférence.

6.12.1 Detailed Description

Classe utilitaire pour le chargement et le prétraitement des images d'inférence.

The documentation for this class was generated from the following file:

- utils/ImageProcessor.py

6.13 ImageProcessor.ImageProcessor Class Reference

Static Public Member Functions

- np.ndarray load_and_preprocess (str file_path)

  Charge une image depuis un fichier, la redimensionne et la normalise.

6.13.1 Detailed Description

Definition at line 8 of file ImageProcessor.py.

6.13.2 Member Function Documentation

6.13.2.1 load_and_preprocess()

```
np.ndarray ImageProcessor.ImageProcessor.load_and_preprocess (
            str file_path )  [static]
```

Charge une image depuis un fichier, la redimensionne et la normalise.

Convertit l'image en RGB, redimensionne en 32x32 (format CIFAR-10), et normalise les pixels (float entre 0.0 et 1.0).

Parameters

file_path	(str) Le chemin complet vers le fichier image.

Returns

np.ndarray Un tableau numpy de forme (32, 32, 3).

Exceptions

ValueError	Si l'image ne peut pas être chargée ou traitée.

Definition at line 18 of file ImageProcessor.py.

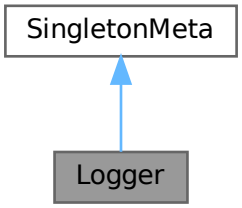
The documentation for this class was generated from the following file:

- utils/ImageProcessor.py

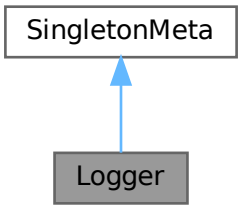
6.14 Logger Class Reference

Classe gérant l'affichage des logs dans la console de l'interface graphique.

Inheritance diagram for Logger:



Collaboration diagram for Logger:



6.14.1 Detailed Description

Classe gérant l'affichage des logs dans la console de l'interface graphique.

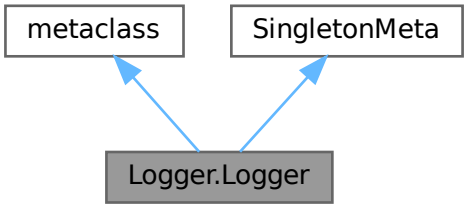
Utilise le pattern Singleton pour être accessible globalement depuis n'importe quelle partie du code. Gère l'écriture thread-safe (via update Tkinter) dans un widget Text.

The documentation for this class was generated from the following file:

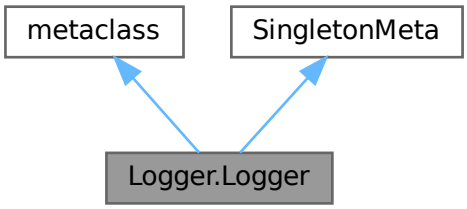
- utils/Logger.py

6.15 Logger.Logger Class Reference

Inheritance diagram for Logger.Logger:



Collaboration diagram for Logger.Logger:



Public Member Functions

- __init__ (self, root, console)

  Constructeur de la classe Logger.
- log (self, message, tag="INFO", buffered=False)

  Ajoute un message dans la console de logs.

Public Attributes

- console
- root

6.15.1 Detailed Description

Definition at line 33 of file Logger.py.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 __init__()

```
Logger.Logger.__init__ (
              self,
              root,
              console )
```

Constructeur de la classe Logger.

Parameters

root	(tk.Tk) L'objet racine de la fenêtre (nécessaire pour forcer la mise à jour graphique).
console	(tk.Text) Le widget Text de l'interface où les logs seront insérés.

Definition at line 39 of file Logger.py.

6.15.3 Member Function Documentation

6.15.3.1 log()

```
Logger.Logger.log (
              self,
              message,
              tag = "INFO",
              buffered = False )
```

Ajoute un message dans la console de logs.

Active temporairement l'édition du widget, insère le message avec un timestamp et la couleur appropriée, scrolle automatiquement vers le bas, puis désactive l'édition.

Parameters

message	(str) Le texte du message à logger.
tag	(str) Le type de message déterminant la couleur (par défaut "INFO").
buffered	(bool) Si True, supprime les lignes précédentes avant d'écrire (utile pour les barres de progression dynamiques).

Definition at line 50 of file Logger.py.

6.15.4 Member Data Documentation

6.15.4.1 console

```
Logger.Logger.console
```

Definition at line 40 of file Logger.py.

6.15.4.2 root

`Logger.Logger.root`

Definition at line 41 of file Logger.py.

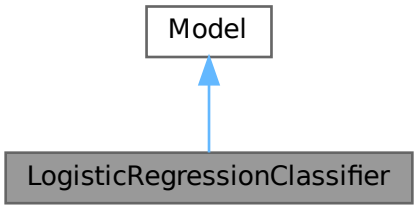
The documentation for this class was generated from the following file:

- utils/Logger.py

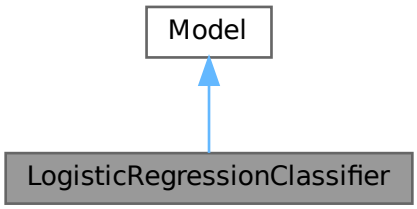
6.16 LogisticRegressionClassifier Class Reference

Implémentation d'un classificateur par Régression Logistique.

Inheritance diagram for LogisticRegressionClassifier:



Collaboration diagram for LogisticRegressionClassifier:



6.16.1 Detailed Description

Implémentation d'un classificateur par Régression Logistique.

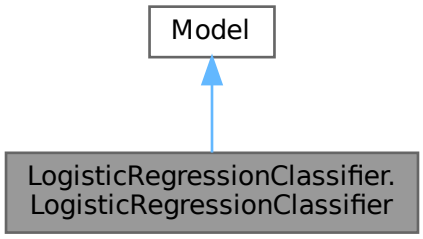
Utilise un pipeline avec Standardisation et PCA (95% variance) avant la régression logistique (solver lbfgs).

The documentation for this class was generated from the following file:

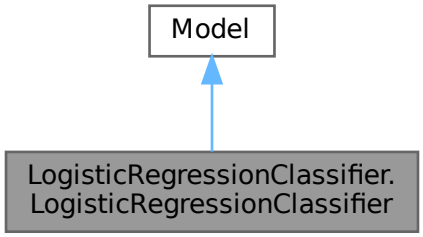
- src/LogisticRegressionClassifier.py

6.17 LogisticRegressionClassifier.LogisticRegressionClassifier Class Reference

Inheritance diagram for LogisticRegressionClassifier.LogisticRegressionClassifier:



Collaboration diagram for LogisticRegressionClassifier.LogisticRegressionClassifier:



Public Member Functions

- __init__ (self, logger, n_train, n_val)

Protected Member Functions

- _create_model (self, **kwargs)

  Crée le pipeline de Régression Logistique.

6.17.1 Detailed Description

Definition at line 14 of file LogisticRegressionClassifier.py.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 __init__()

```
LogisticRegressionClassifier.LogisticRegressionClassifier.__init__ (
              self,
              logger,
              n_train,
              n_val )
```

Definition at line 16 of file LogisticRegressionClassifier.py.

6.17.3 Member Function Documentation

6.17.3.1 _create_model()

```
LogisticRegressionClassifier.LogisticRegressionClassifier._create_model (
              self,
             ** kwargs )  [protected]
```

Crée le pipeline de Régression Logistique.

Parameters

kwargs	Arguments supplémentaires pour `LogisticRegression`.

Returns

sklearn.pipeline.Pipeline Pipeline configuré (StandardScaler -> PCA -> LogisticRegression).

Definition at line 23 of file LogisticRegressionClassifier.py.

The documentation for this class was generated from the following file:

- src/LogisticRegressionClassifier.py

6.18 Metrics Class Reference

Classe utilitaire statique pour le calcul, la visualisation et la comparaison des performances des modèles.

Public Attributes

- list class_names

6.18.1 Detailed Description

Classe utilitaire statique pour le calcul, la visualisation et la comparaison des performances des modèles.

Cette classe regroupe des méthodes statiques permettant de calculer les métriques standards (Accuracy, F1, etc.), de générer des matrices de confusion, de tracer l'historique d'entraînement et de comparer plusieurs modèles entre eux. Elle gère également l'inférence spécifique selon le type de framework (Sklearn, PyTorch, Keras).

6.18.2 Member Data Documentation

6.18.2.1 class_names

```
list Metrics.class_names
```

Initial value:

```
=  [
    "airplane",
    "automobile",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
```

Definition at line 16 of file Metrics.py.

The documentation for this class was generated from the following file:

- utils/Metrics.py

6.19 Metrics.Metrics Class Reference

Static Public Member Functions

- dict calculate_metrics (logger, y_true, y_pred, model_name="Model")

  Calcule un ensemble complet de métriques de classification.
- np.ndarray plot_confusion_matrix (y_true, y_pred, model_name="Model", save_path=None)

  Génère et affiche la matrice de confusion sous forme de heatmap.
- Figure plot_training_history (history, model_name="Model", save_path="_results")

  Trace les courbes d'apprentissage (Accuracy et Loss) pour l'entraînement et la validation.
- None compare_models (dict metrics_dict, save_path="model_comparison.png")

  Compare visuellement et textuellement les performances de plusieurs modèles.
- dict get_precision_recall_f1 (y_true, y_pred, model_name="Model", average="macro")

  Récupère les métriques de précision, rappel et F1, soit par classe, soit moyennées.
- np.ndarray get_predictions (model, data, model_type="sklearn")

  Wrapper universel pour obtenir les prédictions d'un modèle quel que soit son type.

6.19.1 Detailed Description

Definition at line 36 of file Metrics.py.

6.19.2 Member Function Documentation

6.19.2.1 calculate_metrics()

```
dict Metrics.Metrics.calculate_metrics (
              logger,
              y_true,
              y_pred,
              model_name = "Model" )  [static]
```

Calcule un ensemble complet de métriques de classification.

Affiche un rapport textuel dans la console (Accuracy, F1 Macro/Weighted, Precision, Recall) et retourne un dictionnaire contenant ces valeurs ainsi que la matrice de confusion brute.

Parameters

logger	(utils.Logger) logger pour affichage dans la console
y_true	(array-like) Les étiquettes réelles (ground truth).
y_pred	(array-like) Les étiquettes prédites par le modèle.
model_name	(str) Nom du modèle pour l'affichage (par défaut "Model").

Returns

dict Dictionnaire contenant les clés : 'accuracy', 'f1_macro', 'f1_weighted', 'f1_per_class', 'precision_macro', 'recall_macro', 'confusion_matrix'.

Definition at line 49 of file Metrics.py.

6.19.2.2 compare_models()

```
None Metrics.Metrics.compare_models (
             dict metrics_dict,
              save_path = "model_comparison.png" )  [static]
```

Compare visuellement et textuellement les performances de plusieurs modèles.

Génère un tableau comparatif dans la console et sauvegarde un graphique en barres comparant l'Accuracy, le F1 Macro et le F1 Weighted.

Parameters

metrics_dict	(dict) Dictionnaire où la clé est le nom du modèle et la valeur est son dictionnaire de métriques (issu de calculate_metrics).
save_path	(str) Nom du fichier image de sortie (par défaut 'model_comparison.png').

Definition at line 175 of file Metrics.py.

6.19.2.3 get_precision_recall_f1()

```
dict Metrics.Metrics.get_precision_recall_f1 (
              y_true,
              y_pred,
              model_name = "Model",
              average = "macro" )  [static]
```

Récupère les métriques de précision, rappel et F1, soit par classe, soit moyennées.

Parameters

y_true	(array-like) Étiquettes réelles.
y_pred	(array-like) Étiquettes prédites.
model_name	(str) Nom du modèle.
average	(str, optional) Type de moyenne ('macro', 'weighted', 'micro') ou None pour obtenir les scores par classe.

Returns

dict Dictionnaire contenant les scores demandés (clés dépendantes du paramètre average).

Definition at line 227 of file Metrics.py.

6.19.2.4 get_predictions()

```
np.ndarray Metrics.Metrics.get_predictions (
              model,
              data,
              model_type = "sklearn" )  [static]
```

Wrapper universel pour obtenir les prédictions d'un modèle quel que soit son type.

Gère le formatage des données (reshape, conversion tenseurs) et l'appel d'inférence pour Sklearn, PyTorch et Keras.

Parameters

model	L'objet modèle entraîné.
data	(np.ndarray) Les données d'entrée brutes.
model_type	(str) Le type de framework : 'sklearn', 'pytorch', ou 'keras'.

Returns

np.ndarray Tableau numpy 1D contenant les indices des classes prédites.

Definition at line 272 of file Metrics.py.

6.19.2.5 plot_confusion_matrix()

```
np.ndarray Metrics.Metrics.plot_confusion_matrix (
              y_true,
              y_pred,
              model_name = "Model",
              save_path = None )  [static]
```

Génère et affiche la matrice de confusion sous forme de heatmap.

Utilise Seaborn pour l'affichage graphique.

Parameters

y_true	(array-like) Les étiquettes réelles.
y_pred	(array-like) Les étiquettes prédites.
model_name	(str) Nom du modèle pour le titre du graphique.
save_path	(str, optional) Chemin complet pour sauvegarder l'image (ex: "cm.png"). Si None, ne sauvegarde pas.

Returns

np.ndarray La matrice de confusion brute (numpy array).

Definition at line 97 of file Metrics.py.

6.19.2.6 plot_training_history()

```
Figure Metrics.Metrics.plot_training_history (
              history,
              model_name = "Model",
              save_path = "_results" )  [static]
```

Trace les courbes d'apprentissage (Accuracy et Loss) pour l'entraînement et la validation.

Crée une figure avec deux sous-graphiques : un pour l'accuracy, un pour la loss.

Parameters

history	(dict ou object) Objet retourné par l'entraînement (doit contenir les clés 'accuracy', 'val_accuracy', 'loss', 'val_loss'). Peut être un objet Keras History ou un dictionnaire.
model_name	(str) Nom du modèle.
save_path	(str) Dossier de destination pour la sauvegarde de l'image (par défaut "_results").

Definition at line 136 of file Metrics.py.

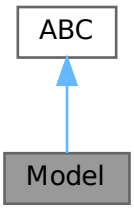
The documentation for this class was generated from the following file:

- utils/Metrics.py

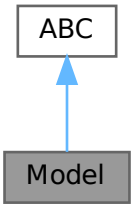
6.20 Model Class Reference

Classe abstraite de base pour tous les modèles de classification.

Inheritance diagram for Model:



Collaboration diagram for Model:



6.20.1 Detailed Description

Classe abstraite de base pour tous les modèles de classification.

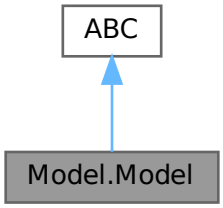
Cette classe gère le cycle de vie complet des modèles : préparation des données, entraînement, évaluation, tests, inférence unique et persistance (sauvegarde/chargement).

The documentation for this class was generated from the following file:

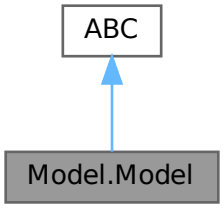
- src/Model.py

6.21 Model.Model Class Reference

Inheritance diagram for Model.Model:



Collaboration diagram for Model.Model:



Public Member Functions

- __init__ (self, logger, n_train, n_val, model_name="BaseModel")

  Constructeur de la classe Model.
- tuple train (self, dict data_dict, class_names=None)

  Exécute le pipeline d'entraînement complet.
- dict test (self, dict data_dict, class_names=None)

  Évalue le modèle sur un jeu de données de test indépendant.
- classify (self, np.ndarray image)

  Effectue une inférence sur une image unique.
- None save_model (self)

  Sauvegarde l'instance du modèle sur le disque via pickle.
- None load_model (self)

  Charge le modèle depuis le disque s'il existe.

Public Attributes

- logger
- n_train
- n_val
- model_name
- model_path
- model

Protected Member Functions

- _create_model (self, ∗∗kwargs)

  Méthode abstraite pour l'instanciation du modèle sous-jacent.
- _prepare_data (self, data_dict)

  Prépare et divise les données brutes en ensembles d'entraînement et de validation.
- _generate_cm_figure (self, cm, class_names)

6.21.1 Detailed Description

Definition at line 24 of file Model.py.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 __init__()

```
Model.Model.__init__ (
               self,
               logger,
               n_train,
               n_val,
               model_name = "BaseModel" )
```

Constructeur de la classe Model.

Parameters

logger	(utils.Logger) logger pour affichage dans la console
n_train	(int) Nombre d'échantillons à utiliser pour l'entraînement.
n_val	(int) Nombre d'échantillons à utiliser pour la validation.
model_name	(str) Nom identifiant du modèle (par défaut "BaseModel").

Note

Définit le chemin de sauvegarde automatique dans le dossier `_models/`.

Definition at line 32 of file Model.py.

6.21.3 Member Function Documentation

6.21.3.1 _create_model()

```
Model.Model._create_model (
              self,
             ** kwargs )  [protected]
```

Méthode abstraite pour l'instanciation du modèle sous-jacent.

Parameters

kwargs	Arguments variables pour la configuration spécifique du modèle.

Returns

L'objet modèle brut (sklearn ou autre) non entraîné.

Note

Doit être implémentée par toutes les classes filles.

Definition at line 46 of file Model.py.

6.21.3.2 _generate_cm_figure()

```
Model.Model._generate_cm_figure (
              self,
              cm,
              class_names )  [protected]
```

Definition at line 75 of file Model.py.

6.21.3.3 _prepare_data()

```
Model.Model._prepare_data (
              self,
              data_dict )  [protected]
```

Prépare et divise les données brutes en ensembles d'entraînement et de validation.

Parameters

data_dict	(dict) Dictionnaire contenant les clés "train_data" et "train_labels".

Returns

tuple (X_train, X_val, y_train, y_val) Les données divisées et stratifiées.

Definition at line 53 of file Model.py.

6.21.3.4 classify()

```
Model.Model.classify (
              self,
             np.ndarray image )
```

Effectue une inférence sur une image unique.

Parameters

image	(np.ndarray) L'image d'entrée (vecteur ou matrice).

Returns

int La classe prédite par le modèle.

Definition at line 160 of file Model.py.

6.21.3.5 load_model()

```
None Model.Model.load_model (
              self )
```

Charge le modèle depuis le disque s'il existe.

Exceptions

FileNotFoundError	Si le fichier du modèle n'existe pas.

Definition at line 185 of file Model.py.

6.21.3.6 save_model()

```
None Model.Model.save_model (
              self )
```

Sauvegarde l'instance du modèle sur le disque via pickle.

Note

Le fichier est stocké dans `_models/{model_name}.pkl`.

Definition at line 176 of file Model.py.

6.21.3.7 test()

```
dict Model.Model.test (
              self,
             dict data_dict,
              class_names = None )
```

Évalue le modèle sur un jeu de données de test indépendant.

Charge le modèle sauvegardé, effectue les prédictions et calcule les métriques.

Parameters

data_dict	(dict) Dictionnaire contenant "test_data" et "test_labels".
class_names	(list, optional) Liste des noms de classes pour la matrice de confusion.

Returns

dict Dictionnaire contenant les métriques de performance (accuracy, f1, confusion matrix, etc.).

Definition at line 138 of file Model.py.

6.21.3.8 train()

```
tuple Model.Model.train (
              self,
             dict data_dict,
              class_names = None )
```

Exécute le pipeline d'entraînement complet.

Prépare les données, redimensionne (flatten) si nécessaire, entraîne le modèle, et évalue les performances sur le set de validation.

Parameters

data_dict	(dict) Dictionnaire des données d'entraînement et labels.
class_names	(list, optional) Liste des noms de classes pour l'annotation de la matrice de confusion.

Returns

tuple (model, metrics) Le modèle entraîné et un dictionnaire de métriques de validation.

Definition at line 96 of file Model.py.

6.21.4 Member Data Documentation

6.21.4.1 logger

```
Model.Model.logger
```

Definition at line 33 of file Model.py.

6.21.4.2 model

```
Model.Model.model
```

Definition at line 38 of file Model.py.

6.21.4.3 model_name

`Model.Model.model_name`

Definition at line 36 of file Model.py.

6.21.4.4 model_path

`Model.Model.model_path`

Definition at line 37 of file Model.py.

6.21.4.5 n_train

`Model.Model.n_train`

Definition at line 34 of file Model.py.

6.21.4.6 n_val

`Model.Model.n_val`

Definition at line 35 of file Model.py.

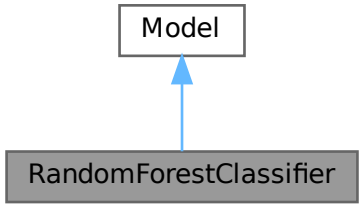
The documentation for this class was generated from the following file:

- src/Model.py

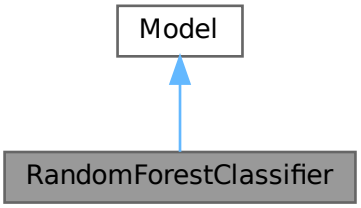
6.22 RandomForestClassifier Class Reference

Implémentation d'un classificateur Random Forest.

Inheritance diagram for RandomForestClassifier:



Collaboration diagram for RandomForestClassifier:



6.22.1 Detailed Description

Implémentation d'un classificateur Random Forest.

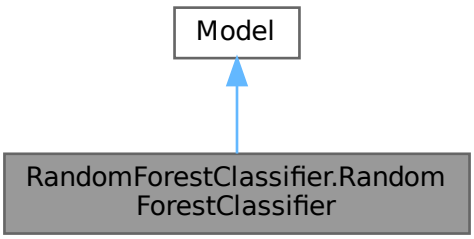
Ensemble de 100 arbres de décision, utilisant tous les cœurs CPU disponibles (n_jobs=-1).

The documentation for this class was generated from the following file:

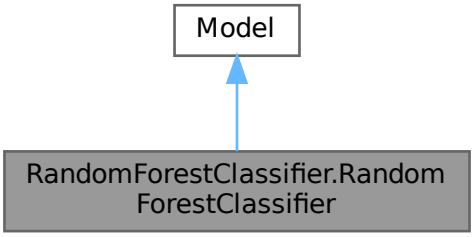
- src/RandomForestClassifier.py

6.23 RandomForestClassifier.RandomForestClassifier Class Reference

Inheritance diagram for RandomForestClassifier.RandomForestClassifier:



Collaboration diagram for RandomForestClassifier.RandomForestClassifier:



Public Member Functions

- __init__ (self, logger, n_train, n_val)

Protected Member Functions

- _create_model (self, ∗∗kwargs)

  Configure et retourne le classificateur Random Forest.

6.23.1 Detailed Description

Definition at line 10 of file RandomForestClassifier.py.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 __init__()

```
RandomForestClassifier.RandomForestClassifier.__init__ (
              self,
              logger,
              n_train,
              n_val )
```

Definition at line 11 of file RandomForestClassifier.py.

6.23.3 Member Function Documentation

6.23.3.1 _create_model()

```
RandomForestClassifier.RandomForestClassifier._create_model (
              self,
             ** kwargs )  [protected]
```

Configure et retourne le classificateur Random Forest.

Parameters

kwargs	Arguments supplémentaires pour `sklearn.ensemble.RandomForestClassifier`.

Returns

sklearn.ensemble.RandomForestClassifier Le modèle configuré.

Definition at line 18 of file RandomForestClassifier.py.

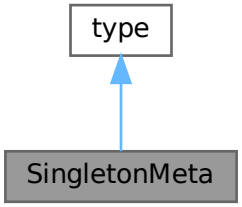
The documentation for this class was generated from the following file:

- src/RandomForestClassifier.py

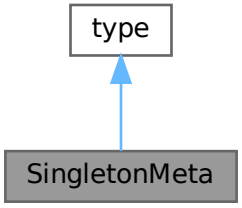
6.24 SingletonMeta Class Reference

Métaclasse implémentant le design pattern Singleton.

Inheritance diagram for SingletonMeta:



Collaboration diagram for SingletonMeta:



6.24.1 Detailed Description

Métaclasse implémentant le design pattern Singleton.

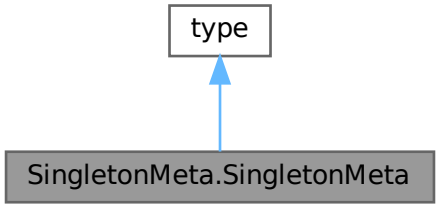
Cette classe assure qu'une classe qui l'utilise comme métaclasse n'aura qu'une seule instance partagée tout au long du cycle de vie de l'application.

The documentation for this class was generated from the following file:

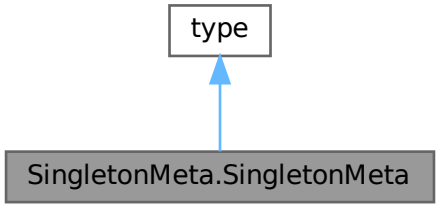
- utils/SingletonMeta.py

6.25 SingletonMeta.SingletonMeta Class Reference

Inheritance diagram for SingletonMeta.SingletonMeta:



Collaboration diagram for SingletonMeta.SingletonMeta:



Public Member Functions

- __call__ (cls, *args, **kwargs)

  Méthode spéciale appelée lors de l'instanciation de la classe.

Static Protected Attributes

- dict _instances = {}

6.25.1 Detailed Description

Definition at line 7 of file SingletonMeta.py.

6.25.2 Member Function Documentation

6.25.2.1 __call__()

```
SingletonMeta.SingletonMeta.__call__ (
              cls,
             * args,
             ** kwargs )
```

Méthode spéciale appelée lors de l'instanciation de la classe.

Vérifie si une instance de la classe existe déjà dans `_instances`. Si oui, retourne l'instance existante. Sinon, appelle le constructeur parent, stocke la nouvelle instance et la retourne.

Parameters

cls	La classe en cours d'instanciation.
args	Arguments positionnels passés au constructeur.
kwargs	Arguments nommés passés au constructeur.

Returns

L'instance unique de la classe.

Definition at line 19 of file SingletonMeta.py.

6.25.3 Member Data Documentation

6.25.3.1 _instances

```
dict SingletonMeta.SingletonMeta._instances = {}  [static], [protected]
```

Definition at line 8 of file SingletonMeta.py.

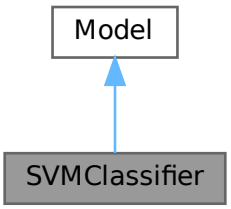
The documentation for this class was generated from the following file:

- utils/SingletonMeta.py

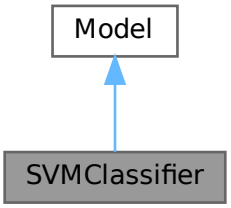
6.26 SVMClassifier Class Reference

Implémentation d'un classificateur Support Vector Machine (SVM) à noyau gaussien.

Inheritance diagram for SVMClassifier:



Collaboration diagram for SVMClassifier:



6.26.1 Detailed Description

Implémentation d'un classificateur Support Vector Machine (SVM) à noyau gaussien.

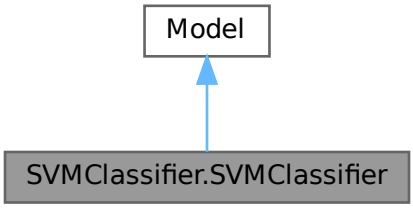
Utilise un pipeline Scikit-learn incluant une standardisation, une réduction de dimension (PCA) conservant 95% de variance, et un SVC à noyau RBF.

The documentation for this class was generated from the following file:

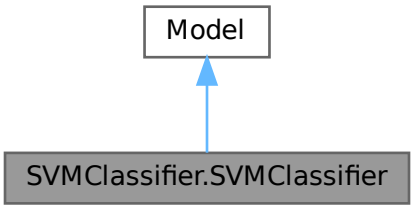
- src/SVMClassifier.py

6.27 SVMClassifier.SVMClassifier Class Reference

Inheritance diagram for SVMClassifier.SVMClassifier:



Collaboration diagram for SVMClassifier.SVMClassifier:



Public Member Functions

- __init__ (self, logger, n_train, n_val)

Protected Member Functions

- _create_model (self, ∗∗kwargs)

  Crée le pipeline SVM.

6.27.1 Detailed Description

Definition at line 14 of file SVMClassifier.py.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 __init__()

```
SVMClassifier.SVMClassifier.__init__ (
              self,
              logger,
              n_train,
              n_val )
```

Definition at line 16 of file SVMClassifier.py.

6.27.3 Member Function Documentation

6.27.3.1 _create_model()

```
SVMClassifier.SVMClassifier._create_model (
              self,
             ** kwargs )  [protected]
```

Crée le pipeline SVM.

Parameters

kwargs	Arguments supplémentaires passés au constructeur SVC.

Returns

sklearn.pipeline.Pipeline Pipeline configuré (StandardScaler -> PCA -> SVC).

Definition at line 23 of file SVMClassifier.py.

The documentation for this class was generated from the following file:

- src/SVMClassifier.py

6.28 TqdmToLogger Class Reference

Redirige la sortie standard (utilisée par tqdm) vers le logger de l'application.

6.28.1 Detailed Description

Redirige la sortie standard (utilisée par tqdm) vers le logger de l'application.

Permet d'afficher les barres de progression textuelles générées par la bibliothèque tqdm directement dans le widget Text de l'interface Tkinter, en gérant le rafraîchissement des lignes.

The documentation for this class was generated from the following file:

- utils/TqdmToLogger.py

6.29 TqdmToLogger.TqdmToLogger Class Reference

Public Member Functions

- __init__ (self, logger, tag="INFO")

  Constructeur de la classe TqdmToLogger.
- write (self, buf)

  Écrit le contenu du buffer dans le logger.
- flush (self)

  Méthode de vidage du buffer (requise par l'interface file-like).

Private Attributes

- __logger
- __tag
- __first_line

6.29.1 Detailed Description

Definition at line 6 of file TqdmToLogger.py.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 __init__()

```
TqdmToLogger.TqdmToLogger.__init__ (
              self,
              logger,
              tag = "INFO" )
```

Constructeur de la classe TqdmToLogger.

Parameters

logger	(Logger) L'instance du logger principal de l'application.
tag	(str) Le tag de couleur à utiliser pour l'affichage (par défaut "INFO").

Definition at line 11 of file TqdmToLogger.py.

6.29.3 Member Function Documentation

6.29.3.1 flush()

```
TqdmToLogger.TqdmToLogger.flush (
              self )
```

Méthode de vidage du buffer (requise par l'interface file-like).

Ne fait rien dans cette implémentation car l'affichage est géré directement dans write().

Definition at line 30 of file TqdmToLogger.py.

6.29.3.2 write()

```
TqdmToLogger.TqdmToLogger.write (
              self,
              buf )
```

Écrit le contenu du buffer dans le logger.

Cette méthode est appelée par tqdm. Elle détecte si le buffer contient du texte, et demande au logger de l'afficher. Gère le mode `buffered` pour éviter l'empilement des lignes de progression.

Parameters

buf	(str) La chaîne de caractères envoyée par tqdm.

Definition at line 22 of file TqdmToLogger.py.

6.29.4 Member Data Documentation

6.29.4.1 __first_line

```
TqdmToLogger.TqdmToLogger.__first_line  [private]
```

Definition at line 14 of file TqdmToLogger.py.

6.29.4.2 __logger

```
TqdmToLogger.TqdmToLogger.__logger  [private]
```

Definition at line 12 of file TqdmToLogger.py.

6.29.4.3 __tag

```
TqdmToLogger.TqdmToLogger.__tag  [private]
```

Definition at line 13 of file TqdmToLogger.py.

The documentation for this class was generated from the following file:

- utils/TqdmToLogger.py

6.30 Window Class Reference

Classe gérant l'interface graphique (GUI) de l'application.

Public Attributes

- bool TEST = True

6.30.1 Detailed Description

Classe gérant l'interface graphique (GUI) de l'application.

Cette classe initialise la fenêtre principale (Tkinter), construit les onglets, les formulaires de configuration (entraînement, test, inférence) et gère l'affichage des résultats (graphiques, logs, images).

6.30.2 Member Data Documentation

6.30.2.1 TEST

```
bool Window.TEST = True
```

Definition at line 9 of file Window.py.

The documentation for this class was generated from the following file:

- Window.py

6.31 Window.Window Class Reference

Public Member Functions

- __init__ (self)

  Constructeur de la classe Window.
- get_root (self)

  Retourne l'objet racine Tkinter.
- get_console (self)

  Retourne le widget de texte utilisé pour la console de logs.
- get_train_button (self)

  Retourne le bouton d'entraînement.
- get_dataset_folder (self)

  Retourne la variable Tkinter contenant le chemin du dossier dataset.
- get_train_data (self)

  Retourne la variable Tkinter contenant la taille des données d'entraînement.
- get_val_data (self)

  Retourne la variable Tkinter contenant la taille des données de validation.
- get_test_button (self)

Retourne le bouton de test.

- get_inference_button (self)

  Retourne le bouton d'inférence.
- update_model_info (self, event)

  Met à jour les informations affichées lors de la sélection d'un modèle.
- create_training_results_tab (self, model_name, metrics_data, confusion_matrix_fig=None, history=None)

  Crée et ajoute un onglet affichant les résultats d'un entraînement.
- create_inference_results_tab (self, results)

  Crée et ajoute un onglet affichant les résultats d'inférence.
- select_dataset (self)

  Ouvre un sélecteur de dossier pour choisir le dataset.
- close_tab (self, tab)

  Ferme un onglet spécifique.
- get_tab_system (self)

  Retourne le gestionnaire d'onglets (Notebook).
- get_selected_model (self)

  Retourne le nom du modèle actuellement sélectionné.
- select_inference_data (self)

  Ouvre un sélecteur de fichier ou de dossier pour l'inférence selon le mode choisi.
- get_inference_data (self)

  Retourne la variable Tkinter contenant le chemin des données d'inférence.
- run (self)

  Lance la boucle principale de l'interface graphique.

Public Attributes

- update_model_info

Private Attributes

- __root
- __models
- __form
- __tab_system
- __logs_tab
- __console
- __scrollbar
- __select_model_area
- __model_label
- __model_information
- __model_var
- __model_dropdown
- __train_area
- __dataset_folder_label
- __dataset_folder
- __dataset_folder_entry
- __dataset_folder_button
- __train_data_label
- __train_data
- __train_data_entry
- __val_data_label

- __val_data
- __val_data_entry
- __train_buttons_frame
- __train_button
- __test_area
- __test_buttons_frame
- __test_button
- __inference_area
- __inference_data_type_label
- __is_file
- __inference_type_frame
- __inference_data_type_file
- __inference_data_type_folder
- __inference_data_label
- __inference_data
- __inference_data_entry
- __inference_data_button
- __inference_buttons_frame
- __inference_button
- __error_label

6.31.1 Detailed Description

Definition at line 18 of file Window.py.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 __init__()

```
Window.Window.__init__ (
              self )
```

Constructeur de la classe Window.

Initialise l'environnement graphique, crée le dossier `_models` si nécessaire, et construit tous les widgets de l'interface (onglets, boutons, champs de saisie, console).

Definition at line 23 of file Window.py.

6.31.3 Member Function Documentation

6.31.3.1 close_tab()

```
Window.Window.close_tab (
              self,
              tab )
```

Ferme un onglet spécifique.

Parameters

tab	(tk.Widget) Le widget représentant l'onglet à fermer.

Definition at line 691 of file Window.py.

6.31.3.2 create_inference_results_tab()

```
Window.Window.create_inference_results_tab (
              self,
              results )
```

Crée et ajoute un onglet affichant les résultats d'inférence.

Affiche une liste déroulante d'images avec leur prédiction associée.

Parameters

results	(list) Liste de tuples (nom_fichier, prédiction, tableau_image).

Definition at line 572 of file Window.py.

6.31.3.3 create_training_results_tab()

```
Window.Window.create_training_results_tab (
              self,
              model_name,
              metrics_data,
              confusion_matrix_fig = None,
              history = None )
```

Crée et ajoute un onglet affichant les résultats d'un entraînement.

Construit l'interface pour afficher les métriques (précision, rappel, F1), le rapport de classification, la matrice de confusion et l'historique (si applicable).

Parameters

model_name	(str) Le nom du modèle entraîné.
metrics_data	(dict) Dictionnaire contenant les métriques calculées.
confusion_matrix_fig	(matplotlib.figure.Figure, optional) Figure de la matrice de confusion.
history	(matplotlib.figure.Figure, optional) Figure de l'historique d'entraînement (pour CNN).

Definition at line 395 of file Window.py.

6.31.3.4 get_console()

```
Window.Window.get_console (
              self )
```

Retourne le widget de texte utilisé pour la console de logs.

Returns

(tk.Text) Le widget de texte des logs.

Definition at line 323 of file Window.py.

6.31.3.5 get_dataset_folder()

```
Window.Window.get_dataset_folder (
              self )
```

Retourne la variable Tkinter contenant le chemin du dossier dataset.

Returns

(tk.StringVar) La variable liée au champ dataset.

Definition at line 335 of file Window.py.

6.31.3.6 get_inference_button()

```
Window.Window.get_inference_button (
              self )
```

Retourne le bouton d'inférence.

Returns

(tk.Button) Le bouton "Classify Image(s)".

Definition at line 359 of file Window.py.

6.31.3.7 get_inference_data()

```
Window.Window.get_inference_data (
              self )
```

Retourne la variable Tkinter contenant le chemin des données d'inférence.

Returns

(tk.StringVar) La variable liée au champ d'inférence.

Definition at line 721 of file Window.py.

6.31.3.8 get_root()

```
Window.Window.get_root (
              self )
```

Retourne l'objet racine Tkinter.

Returns

(tk.Tk) L'objet fenêtre principale.

Definition at line 317 of file Window.py.

6.31.3.9 get_selected_model()

```
Window.Window.get_selected_model (
              self )
```

Retourne le nom du modèle actuellement sélectionné.

Returns

(str) Le nom du modèle.

Definition at line 703 of file Window.py.

6.31.3.10 get_tab_system()

```
Window.Window.get_tab_system (
              self )
```

Retourne le gestionnaire d'onglets (Notebook).

Returns

(ttk.Notebook) Le gestionnaire d'onglets principal.

Definition at line 697 of file Window.py.

6.31.3.11 get_test_button()

```
Window.Window.get_test_button (
              self )
```

Retourne le bouton de test.

Returns

(tk.Button) Le bouton "Test Model".

Definition at line 353 of file Window.py.

6.31.3.12 get_train_button()

```
Window.Window.get_train_button (
              self )
```

Retourne le bouton d'entraînement.

Returns

(tk.Button) Le bouton "Train Model".

Definition at line 329 of file Window.py.

6.31.3.13 get_train_data()

```
Window.Window.get_train_data (
              self )
```

Retourne la variable Tkinter contenant la taille des données d'entraînement.

Returns

(tk.StringVar) La variable de taille d'entraînement.

Definition at line 341 of file Window.py.

6.31.3.14 get_val_data()

```
Window.Window.get_val_data (
              self )
```

Retourne la variable Tkinter contenant la taille des données de validation.

Returns

(tk.StringVar) La variable de taille de validation.

Definition at line 347 of file Window.py.

6.31.3.15 run()

```
Window.Window.run (
              self )
```

Lance la boucle principale de l'interface graphique.

Cette méthode bloque l'exécution jusqu'à la fermeture de la fenêtre.

Definition at line 727 of file Window.py.

6.31.3.16 select_dataset()

```
Window.Window.select_dataset (
              self )
```

Ouvre un sélecteur de dossier pour choisir le dataset.

Definition at line 685 of file Window.py.

6.31.3.17 select_inference_data()

```
Window.Window.select_inference_data (
              self )
```

Ouvre un sélecteur de fichier ou de dossier pour l'inférence selon le mode choisi.

Definition at line 708 of file Window.py.

6.31.3.18 update_model_info()

```
Window.Window.update_model_info (
              self,
              event )
```

Met à jour les informations affichées lors de la sélection d'un modèle.

Vérifie l'existence du modèle sur le disque et affiche un message d'erreur si nécessaire.

Parameters

event	L'événement Tkinter déclencheur (changement de sélection).

Definition at line 366 of file Window.py.

6.31.4 Member Data Documentation

6.31.4.1 __console

```
Window.Window.__console  [private]
```

Definition at line 65 of file Window.py.

6.31.4.2 __dataset_folder

```
Window.Window.__dataset_folder  [private]
```

Definition at line 134 of file Window.py.

6.31.4.3 __dataset_folder_button

```
Window.Window.__dataset_folder_button  [private]
```

Definition at line 138 of file Window.py.

6.31.4.4 __dataset_folder_entry

```
Window.Window.__dataset_folder_entry  [private]
```

Definition at line 135 of file Window.py.

6.31.4.5 __dataset_folder_label

```
Window.Window.__dataset_folder_label  [private]
```

Definition at line 127 of file Window.py.

6.31.4.6 __error_label

```
Window.Window.__error_label  [private]
```

Definition at line 296 of file Window.py.

6.31.4.7 __form

```
Window.Window.__form  [private]
```

Definition at line 55 of file Window.py.

6.31.4.8 __inference_area

```
Window.Window.__inference_area  [private]
```

Definition at line 219 of file Window.py.

6.31.4.9 __inference_button

```
Window.Window.__inference_button  [private]
```

Definition at line 286 of file Window.py.

6.31.4.10 __inference_buttons_frame

```
Window.Window.__inference_buttons_frame  [private]
```

Definition at line 280 of file Window.py.

6.31.4.11 __inference_data

```
Window.Window.__inference_data  [private]
```

Definition at line 265 of file Window.py.

6.31.4.12 __inference_data_button

```
Window.Window.__inference_data_button  [private]
```

Definition at line 269 of file Window.py.

6.31.4.13 __inference_data_entry

```
Window.Window.__inference_data_entry  [private]
```

Definition at line 266 of file Window.py.

6.31.4.14 __inference_data_label

```
Window.Window.__inference_data_label  [private]
```

Definition at line 258 of file Window.py.

6.31.4.15 __inference_data_type_file

```
Window.Window.__inference_data_type_file  [private]
```

Definition at line 236 of file Window.py.

6.31.4.16 __inference_data_type_folder

```
Window.Window.__inference_data_type_folder  [private]
```

Definition at line 242 of file Window.py.

6.31.4.17 __inference_data_type_label

```
Window.Window.__inference_data_type_label  [private]
```

Definition at line 226 of file Window.py.

6.31.4.18 __inference_type_frame

```
Window.Window.__inference_type_frame  [private]
```

Definition at line 234 of file Window.py.

6.31.4.19 __is_file

`Window.Window.__is_file [private]`

Definition at line 233 of file Window.py.

6.31.4.20 __logs_tab

`Window.Window.__logs_tab [private]`

Definition at line 61 of file Window.py.

6.31.4.21 __model_dropdown

`Window.Window.__model_dropdown [private]`

Definition at line 104 of file Window.py.

6.31.4.22 __model_information

`Window.Window.__model_information [private]`

Definition at line 89 of file Window.py.

6.31.4.23 __model_label

`Window.Window.__model_label [private]`

Definition at line 81 of file Window.py.

6.31.4.24 __model_var

`Window.Window.__model_var [private]`

Definition at line 103 of file Window.py.

6.31.4.25 __models

`Window.Window.__models [private]`

Definition at line 53 of file Window.py.

6.31.4.26 __root

`Window.Window.__root [private]`

Definition at line 26 of file Window.py.

6.31.4.27 __scrollbar

`Window.Window.__scrollbar [private]`

Definition at line 66 of file Window.py.

6.31.4.28 __select_model_area

`Window.Window.__select_model_area [private]`

Definition at line 71 of file Window.py.

6.31.4.29 __tab_system

`Window.Window.__tab_system [private]`

Definition at line 58 of file Window.py.

6.31.4.30 __test_area

`Window.Window.__test_area [private]`

Definition at line 196 of file Window.py.

6.31.4.31 __test_button

`Window.Window.__test_button [private]`

Definition at line 208 of file Window.py.

6.31.4.32 __test_buttons_frame

`Window.Window.__test_buttons_frame [private]`

Definition at line 202 of file Window.py.

6.31.4.33 __train_area

`Window.Window.__train_area [private]`

Definition at line 120 of file Window.py.

6.31.4.34 __train_button

`Window.Window.__train_button [private]`

Definition at line 185 of file Window.py.

6.31.4.35 __train_buttons_frame

`Window.Window.__train_buttons_frame [private]`

Definition at line 179 of file Window.py.

6.31.4.36 __train_data

`Window.Window.__train_data [private]`

Definition at line 153 of file Window.py.

6.31.4.37 __train_data_entry

`Window.Window.__train_data_entry [private]`

Definition at line 154 of file Window.py.

6.31.4.38 __train_data_label

`Window.Window.__train_data_label [private]`

Definition at line 146 of file Window.py.

6.31.4.39 __val_data

`Window.Window.__val_data [private]`

Definition at line 171 of file Window.py.

6.31.4.40 __val_data_entry

`Window.Window.__val_data_entry [private]`

Definition at line 172 of file Window.py.

6.31.4.41 __val_data_label

`Window.Window.__val_data_label [private]`

Definition at line 164 of file Window.py.

6.31.4.42 update_model_info

`Window.Window.update_model_info`

Definition at line 117 of file Window.py.

The documentation for this class was generated from the following file:

- Window.py

Chapter 7

File Documentation

7.1 Controller.py File Reference

Classes

- class Controller.Controller

Namespaces

- namespace Controller

7.2 Controller.py

Go to the documentation of this file.

```
00001 import os
00002 import threading
00003
00004 from utils.Logger import Logger
00005 from utils.SingletonMeta import SingletonMeta
00006 from utils.DataHandler import DataHandler
00007 from utils.ImageProcessor import ImageProcessor
00008
00009 from src.DecisionTreeClassifier import DecisionTreeClassifier
00010 from src.RandomForestClassifier import RandomForestClassifier
00011 from src.LogisticRegressionClassifier import LogisticRegressionClassifier
00012 from src.SVMClassifier import SVMClassifier
00013 from src.GBMClassifier import GBMClassifier
00014 from src.CNNClassifier import CNNClassifier
00015
00016
00017
00024 class Controller(metaclass=SingletonMeta):
00025
00030     def __init__(self, window):
00031         self.window = window
00032         self.logger = Logger(self.window.get_root(), self.window.get_console())
00033
00034         self.window.get_train_button().config(command=self.start_train)
00035         self.window.get_test_button().config(command=self.start_test)
00036         self.window.get_inference_button().config(command=self.start_classify)
00037
00038         self.dataset = None
00039         self.train_data = None
00040         self.val_data = None
00041         self.data_handler = None
00042         self.model = None
00043         self.training_counter = 0
00044         self.inference_path = None
```

```
00045
00046         self.model_classes = {
00047             "CNN": CNNClassifier,
00048             "Decision Tree": DecisionTreeClassifier,
00049             "Random Forest": RandomForestClassifier,
00050             "Logistic Regression": LogisticRegressionClassifier,
00051             "SVM": SVMClassifier,
00052             "Gradient Boosting": GBMClassifier,
00053         }
00054
00055
00059     def start_train(self):
00060         self.logger.log("Start training...", "INFO")
00061         if not self.check_inputs("train"):
00062             self.logger.log("Input error, aborting...", "ERROR")
00063         else:
00064             selected_model = self.window.get_selected_model()
00065
00066             if selected_model not in self.model_classes:
00067                 self.logger.log(
00068                     f"Model '{selected_model}' is not implemented yet!", "ERROR"
00069                 )
00070                 return
00071
00072             self.training_counter += 1
00073
00074             self.window.get_train_button().config(
00075                 state="disabled", text="Training in progress..."
00076             )
00077
00078             thread = threading.Thread(
00079                 target=self._train_model_thread, args=(selected_model,)
00080             )
00081             thread.daemon = True
00082             thread.start()
00083
00084
00089     def _train_model_thread(self, model_name):
00090         try:
00091             self.logger.log("Loading dataset...", "INFO")
00092             self.data_handler = DataHandler(self.logger, self.dataset)
00093             self.data_handler.load_data(normalize=True, flatten=True)
00094
00095             self.logger.log(f"Initializing {model_name} model...", "INFO")
00096             model_class = self.model_classes[model_name]
00097
00098             self.model = model_class(
00099                 self.logger, int(self.train_data), int(self.val_data)
00100             )
00101
00102             self.logger.log(f"Training {model_name} model...", "INFO")
00103             model_data, metrics_data = self.model.train(
00104                 self.data_handler.get_data(), self.data_handler.get_class_names()
00105             )
00106
00107             self.logger.log("Training completed successfully!", "SUCCESS")
00108
00109             self.window.get_root().after(
00110                 0, self._display_training_results_display_training_results, model_name, metrics_data
00111             )
00112
00113         except Exception as e:
00114             self.logger.log(f"Training error: {str(e)}", "ERROR")
00115             import traceback
00116
00117             self.logger.log(traceback.format_exc(), "ERROR")
00118         finally:
00119             self.window.get_root().after(0, self._reset_train_button_reset_train_button)
00120
00121
00126     def _display_training_results(self, model_name, metrics_data):
00127         try:
00128             training_history_fig = None
00129             if model_name == "CNN":
00130                 training_history_fig = metrics_data.get("history_fig", None)
00131
00132             confusion_matrix_fig = metrics_data.get("confusion_matrix_fig", None)
00133
00134             tab_name = f"{model_name} #{self.training_counter}"
00135
00136             self.window.create_training_results_tab(
00137                 model_name=tab_name,
00138                 metrics_data=metrics_data,
00139                 confusion_matrix_fig=confusion_matrix_fig,
00140                 history=training_history_fig,
00141             )
00142
```

```
00143             self.logger.log(f"Results tab created: {tab_name}", "INFO")
00144
00145         except Exception as e:
00146             self.logger.log(f"Error creating results tab: {str(e)}", "ERROR")
00147             import traceback
00148
00149             self.logger.log(traceback.format_exc(), "ERROR")
00150
00151
00153     def _reset_train_button(self):
00154         self.window.get_train_button().config(state="normal", text="Train Model")
00155
00156
00159     def start_test(self):
00160         self.logger.log("Start Testing...", "INFO")
00161         if not self.check_inputs("test"):
00162             self.logger.log("Input error, aborting...", "ERROR")
00163         else:
00164             selected_model = self.window.get_selected_model()
00165             self.window.get_test_button().config(
00166                 state="disabled", text="Testing in progress..."
00167             )
00168
00169             thread = threading.Thread(
00170                 target=self._test_model_thread, args=(selected_model,)
00171             )
00172             thread.daemon = True
00173             thread.start()
00174
00175
00179     def _test_model_thread(self, model_name):
00180         try:
00181             self.logger.log("Loading dataset for testing...", "INFO")
00182             if self.data_handler is None:
00183                 self.data_handler = DataHandler(self.logger, self.dataset)
00184                 self.data_handler.load_data(normalize=True, flatten=True)
00185
00186             self.logger.log(f"Initializing {model_name} wrapper...", "INFO")
00187             model_class = self.model_classes[model_name]
00188             self.model = model_class(
00189                 self.logger, 0, 0
00190             )  # Pas besoin de n_train/n_val pour tester
00191
00192             self.logger.log(f"Testing {model_name} model...", "INFO")
00193             metrics_data = self.model.test(
00194                 self.data_handler.get_data_dict(), self.data_handler.get_class_names()
00195             )
00196
00197             self.logger.log("Testing completed successfully!", "SUCCESS")
00198             self.window.get_root().after(
00199                 0, self._display_test_results_display_test_results, model_name, metrics_data
00200             )
00201
00202         except Exception as e:
00203             self.logger.log(f"Testing error: {str(e)}", "ERROR")
00204             import traceback
00205
00206             self.logger.log(traceback.format_exc(), "ERROR")
00207         finally:
00208             self.window.get_root().after(0, self._reset_test_button_reset_test_button)
00209
00210
00214     def _display_test_results(self, model_name, metrics_data):
00215         try:
00216             confusion_matrix_fig = metrics_data.get("confusion_matrix_fig", None)
00217             tab_name = f"TEST: {model_name}"
00218             self.window.create_training_results_tab(
00219                 model_name=tab_name,
00220                 metrics_data=metrics_data,
00221                 confusion_matrix_fig=confusion_matrix_fig,
00222                 history=None,
00223             )
00224             self.logger.log(f"Test results tab created: {tab_name}", "INFO")
00225         except Exception as e:
00226             self.logger.log(f"Error creating results tab: {str(e)}", "ERROR")
00227
00228
00230     def _reset_test_button(self):
00231         self.window.get_test_button().config(state="normal", text="Test Model")
00232
00233
00236     def start_classify(self):
00237         self.logger.log("Start Classify...", "INFO")
00238         if not self.check_inputs("classify"):
00239             self.logger.log("Input error, aborting...", "ERROR")
00240         else:
00241             selected_model = self.window.get_selected_model()
```

```
00242             self.window.get_inference_button().config(
00243                 state="disabled", text="Classifying..."
00244             )
00245 
00246             thread = threading.Thread(
00247                 target=self._classify_model_thread, args=(selected_model,)
00248             )
00249             thread.daemon = True
00250             thread.start()
00251 
00252 
00256     def _classify_model_thread(self, model_name):
00257         try:
00258             self.logger.log(f"Loading {model_name} for inference...", "INFO")
00259             model_class = self.model_classes[model_name]
00260             self.model = model_class(self.logger, 0, 0)
00261             self.model.load_model()
00262 
00263             files_to_process = []
00264             if os.path.isfile(self.inference_path):
00265                 files_to_process.append(self.inference_path)
00266             elif os.path.isdir(self.inference_path):
00267                 valid_extensions = (".png", ".jpg", ".jpeg", ".bmp")
00268                 files_to_process = [
00269                     os.path.join(self.inference_path, f)
00270                     for f in os.listdir(self.inference_path)
00271                     if f.lower().endswith(valid_extensions)
00272                 ]
00273 
00274             if not files_to_process:
00275                 self.logger.log("No valid images found!", "ERROR")
00276                 return
00277 
00278             self.logger.log(f"Processing {len(files_to_process)} image(s)...", "INFO")
00279 
00280             if self.data_handler is not None:
00281                 class_names = self.data_handler.get_class_names()
00282             else:
00283                 class_names = [
00284                     "Airplane",
00285                     "Automobile",
00286                     "Bird",
00287                     "Cat",
00288                     "Deer",
00289                     "Dog",
00290                     "Frog",
00291                     "Horse",
00292                     "Ship",
00293                     "Truck",
00294                 ]
00295 
00296             results_data = []
00297 
00298             for file_path in files_to_process:
00299                 try:
00300                     img_array = ImageProcessor.load_and_preprocess(file_path)
00301 
00302                     prediction_index = self.model.classify(img_array)
00303 
00304                     if 0 <= prediction_index < len(class_names):
00305                         predicted_label = class_names[prediction_index]
00306                     else:
00307                         predicted_label = f"Unknown ({prediction_index})"
00308 
00309                     results_data.append(
00310                         (os.path.basename(file_path), predicted_label, img_array)
00311                     )
00312 
00313                     self.logger.log(
00314                         f"{os.path.basename(file_path)} -> {predicted_label}", "RESULT"
00315                     )
00316 
00317                 except Exception as img_err:
00318                     self.logger.log(
00319                         f"Error processing {os.path.basename(file_path)}: {str(img_err)}",
00320                         "ERROR",
00321                     )
00322 
00323             self.window.get_root().after(
00324                 0, self._display_classification_results_display_classification_results, results_data
00325             )
00326 
00327         except Exception as e:
00328             self.logger.log(f"Inference error: {str(e)}", "ERROR")
00329             import traceback
00330 
00331             self.logger.log(traceback.format_exc(), "ERROR")
```

```
00332         finally:
00333             self.window.get_root().after(0, self._reset_classify_button_reset_classify_button)
00334
00335
00338     def _display_classification_results(self, results_data):
00339         try:
00340             self.window.create_inference_results_tab(results_data)
00341             self.logger.log("Inference results tab displayed.", "INFO")
00342         except Exception as e:
00343             self.logger.log(f"UI Error: {str(e)}", "ERROR")
00344
00345
00347     def _reset_classify_button(self):
00348         self.window.get_inference_button().config(
00349             state="normal", text="Classify Image(s)"
00350         )
00351
00352
00356     def check_inputs(self, action):
00357         match action:
00358             case "train":
00359                 self.dataset = self.window.get_dataset_folder().get()
00360                 self.train_data = self.window.get_train_data().get()
00361                 self.val_data = self.window.get_val_data().get()
00362
00363                 if (not os.path.exists(self.dataset)) or self.dataset == "":
00364                     self.logger.log("Dataset folder not found", "ERROR")
00365                     return False
00366
00367                 if self.train_data == "" or not self.train_data.isdigit():
00368                     self.logger.log("Missing or invalid train data number", "ERROR")
00369                     return False
00370
00371                 if self.val_data == "" or not self.val_data.isdigit():
00372                     self.logger.log(
00373                         "Missing or invalid validation data number", "ERROR"
00374                     )
00375                     return False
00376
00377             case "test":
00378                 self.dataset = self.window.get_dataset_folder().get()
00379                 if (not os.path.exists(self.dataset)) or self.dataset == "":
00380                     self.logger.log(
00381                         "Dataset folder not found (needed for test data)", "ERROR"
00382                     )
00383                     return False
00384
00385             case "classify":
00386                 self.inference_path = self.window.get_inference_data().get()
00387
00388                 if not os.path.exists(self.inference_path) or self.inference_path == "":
00389                     self.logger.log("Inference file/folder not found", "ERROR")
00390                     return False
00391
00392         return True
```

7.3 main.py File Reference

Point d'entrée principal de l'application de classification.

Namespaces

- namespace main

Variables

- main.window = Window()
- main.Controller = Controller(window)

7.3.1 Detailed Description

Point d'entrée principal de l'application de classification.

Ce script orchestre le lancement de l'application en suivant le pattern MVC :

1. Instanciation de la vue (Window).
2. Instanciation du contrôleur (Controller) qui lie la logique à la vue.
3. Démarrage de la boucle d'événements principale (mainloop).

Author

Romain Brouard et Paul Henry

Definition in file main.py.

7.4 main.py

Go to the documentation of this file.

```
00001 from Window import Window
00002 from Controller import Controller
00003
00004
00012
00013 window = Window()
00014 Controller = Controller(window)
00015 window.run()
```

7.5 src/CNNClassifier.py File Reference

Classes

- class CNNClassifier.CNNArchitecture

  Architecture du réseau de neurones convolutif (CNN) définie avec PyTorch.
- class CNNClassifier.CNNClassifier

Namespaces

- namespace CNNClassifier

7.6 CNNClassifier.py

Go to the documentation of this file.

```
00001 import os
00002 import copy
00003 import numpy as np
00004 import torch
00005 import torch.nn as nn
00006 import torch.optim as optim
00007 from torch.utils.data import TensorDataset, DataLoader
00008 from tqdm import tqdm
00009
00010 from src.Model import Model
00011 from utils.Metrics import Metrics
00012 from utils.TqdmToLogger import TqdmToLogger
00013
00014
00015
00021 class CNNArchitecture(nn.Module):
00022
00023     def __init__(self):
00024         super(CNNArchitecture, self).__init__()
00025         self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
00026         self.bn1_1 = nn.BatchNorm2d(32)
00027         self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
00028         self.bn1_2 = nn.BatchNorm2d(32)
00029         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
00030         self.dropout1 = nn.Dropout(0.3)
00031
00032         self.conv2_1 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
00033         self.bn2_1 = nn.BatchNorm2d(64)
00034         self.conv2_2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
00035         self.bn2_2 = nn.BatchNorm2d(64)
00036         self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
00037         self.dropout2 = nn.Dropout(0.4)
00038
00039         self.conv3_1 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
00040         self.bn3_1 = nn.BatchNorm2d(128)
00041         self.conv3_2 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
00042         self.bn3_2 = nn.BatchNorm2d(128)
00043         self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
00044         self.dropout3 = nn.Dropout(0.5)
00045
00046         self.flatten = nn.Flatten()
00047         self.fc1 = nn.Linear(128 * 4 * 4, 128)
00048         self.bn_fc = nn.BatchNorm1d(128)
00049         self.dropout_fc = nn.Dropout(0.5)
00050         self.fc2 = nn.Linear(128, 10)
00051         self.relu = nn.ReLU()
00052
00053
00057     def forward(self, x):
00058         x = self.relu(self.bn1_1(self.conv1_1(x)))
00059         x = self.relu(self.bn1_2(self.conv1_2(x)))
00060         x = self.pool1(x)
00061         x = self.dropout1(x)
00062
00063         x = self.relu(self.bn2_1(self.conv2_1(x)))
00064         x = self.relu(self.bn2_2(self.conv2_2(x)))
00065         x = self.pool2(x)
00066         x = self.dropout2(x)
00067
00068         x = self.relu(self.bn3_1(self.conv3_1(x)))
00069         x = self.relu(self.bn3_2(self.conv3_2(x)))
00070         x = self.pool3(x)
00071         x = self.dropout3(x)
00072
00073         x = self.flatten(x)
00074         x = self.relu(self.bn_fc(self.fc1(x)))
00075         x = self.dropout_fc(x)
00076         x = self.fc2(x)
00077         return x
00078
00079
00080
00086 class CNNClassifier(Model):
00087     def __init__(self, logger, n_train, n_val):
00088         super().__init__(logger, n_train, n_val, model_name="CNN")
00089         self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
00090         self.model_path = "_models/CNN.pth"
00091         # Structure pour stocker l'historique (compatible avec votre snippet)
00092         self.history = {"loss": [], "accuracy": [], "val_loss": [], "val_accuracy": []}
00093         self.tqdm_out = TqdmToLogger(self.logger, "TQDM")
00094
00095     def _create_model(self, **kwargs):
```

```
00096         return CNNArchitecture().to(self.device)
00097
00098
00110     def train(self, data_dict: dict, class_names=None) -> tuple:
00111         self.logger.log(f"[{self.model_name}] Using device: {self.device}", "INFO")
00112
00113         X_train_raw, X_val_raw, y_train, y_val = self._prepare_data(data_dict)
00114
00115         if len(X_train_raw.shape) == 2:
00116             X_train_raw = X_train_raw.reshape(-1, 32, 32, 3)
00117             X_val_raw = X_val_raw.reshape(-1, 32, 32, 3)
00118
00119         X_train = torch.FloatTensor(X_train_raw).permute(0, 3, 1, 2)
00120         y_train_t = torch.LongTensor(y_train)
00121
00122         X_val = torch.FloatTensor(X_val_raw).permute(0, 3, 1, 2)
00123         y_val_t = torch.LongTensor(y_val)
00124
00125         train_loader = DataLoader(
00126             TensorDataset(X_train, y_train_t), batch_size=32, shuffle=True
00127         )
00128         val_loader = DataLoader(
00129             TensorDataset(X_val, y_val_t), batch_size=32, shuffle=False
00130         )
00131
00132         # Initialisation Modèle, Optimiseur, Scheduler, EarlyStopping
00133         self.model = self._create_model()
00134         criterion = nn.CrossEntropyLoss()
00135         optimizer = optim.Adam(self.model.parameters(), lr=0.001)
00136         scheduler = optim.lr_scheduler.ReduceLROnPlateau(
00137             optimizer, mode="min", factor=0.1, patience=3
00138         )
00139
00140         # Paramètres Early Stopping
00141         patience = 5
00142         best_val_loss = float("inf")
00143         patience_counter = 0
00144         best_model_state = None
00145
00146         self.logger.log(f"[{self.model_name}] Training started (Epochs=50)...", "INFO")
00147
00148         for epoch in range(50):
00149             self.model.train()
00150             train_loss = 0.0
00151             train_correct = 0
00152             train_total = 0
00153
00154             train_pbar = tqdm(
00155                 train_loader,
00156                 desc=f"Epoch {epoch + 1:2d}/50 [Train]",
00157                 position=0,
00158                 leave=True,
00159                 ncols=100,
00160                 file=self.tqdm_out,
00161                 mininterval=1.0,
00162             )
00163
00164             for batch_X, batch_y in train_pbar:
00165                 batch_X, batch_y = batch_X.to(self.device), batch_y.to(self.device)
00166
00167                 optimizer.zero_grad()
00168                 outputs = self.model(batch_X)
00169                 loss = criterion(outputs, batch_y)
00170                 loss.backward()
00171                 optimizer.step()
00172
00173                 train_loss += loss.item() * batch_X.size(0)
00174                 _, predicted = outputs.max(1)
00175                 train_total += batch_y.size(0)
00176                 train_correct += predicted.eq(batch_y).sum().item()
00177
00178                 train_pbar.set_postfix(
00179                     {
00180                         "loss": f"{loss.item():.3f}",
00181                         "acc": f"{train_correct / train_total:.3f}",
00182                     }
00183                 )
00184
00185             train_pbar.close()
00186
00187             train_loss = train_loss / train_total
00188             train_acc = train_correct / train_total
00189
00190             # Phase de Validation
00191             self.model.eval()
00192             val_loss = 0.0
00193             val_correct = 0
```

```
00194             val_total = 0
00195
00196             val_pbar = tqdm(
00197                 val_loader,
00198                 desc=f"Epoch {epoch + 1:2d}/50 [Val]  ",
00199                 position=0,
00200                 leave=True,
00201                 ncols=100,
00202                 file=self.tqdm_out,
00203                 mininterval=1.0,
00204             )
00205
00206             with torch.no_grad():
00207                 for batch_X, batch_y in val_pbar:
00208                     batch_X, batch_y = batch_X.to(self.device), batch_y.to(self.device)
00209                     outputs = self.model(batch_X)
00210                     loss = criterion(outputs, batch_y)
00211
00212                     val_loss += loss.item() * batch_X.size(0)
00213                     _, predicted = outputs.max(1)
00214                     val_total += batch_y.size(0)
00215                     val_correct += predicted.eq(batch_y).sum().item()
00216
00217                     val_pbar.set_postfix(
00218                         {
00219                             "loss": f"{loss.item():.3f}",
00220                             "acc": f"{val_correct / val_total:.3f}",
00221                         }
00222                     )
00223
00224             val_pbar.close()
00225
00226             val_loss = val_loss / val_total
00227             val_acc = val_correct / val_total
00228
00229             # Mise à jour historique
00230             self.history["loss"].append(train_loss)
00231             self.history["accuracy"].append(train_acc)
00232             self.history["val_loss"].append(val_loss)
00233             self.history["val_accuracy"].append(val_acc)
00234
00235             # Scheduler step
00236             current_lr = optimizer.param_groups[0]["lr"]
00237             scheduler.step(val_loss)
00238
00239             self.logger.log(
00240                 f"Epoch {epoch + 1:2d}/50 Summary - Train Loss: {train_loss:.4f}, Train Acc:
    {train_acc:.4f} | Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f} | LR: {current_lr:.2e}",
00241                 "RESULT",
00242             )
00243
00244             # Early Stopping
00245             if val_loss < best_val_loss:
00246                 best_val_loss = val_loss
00247                 best_model_state = copy.deepcopy(self.model.state_dict())
00248                 patience_counter = 0
00249             else:
00250                 patience_counter += 1
00251                 if patience_counter >= patience:
00252                     print(f"\nEarly stopping triggered after epoch {epoch + 1}")
00253                     break
00254
00255         if best_model_state is not None:
00256             self.model.load_state_dict(best_model_state)
00257             self.logger.log("Restored best model based on validation loss.", "INFO")
00258
00259         # 4. Évaluation Finale pour le retour (Compatible Controller)
00260         self.logger.log(f"[{self.model_name}] Generating final metrics...", "INFO")
00261         val_preds = self._predict_batch(X_val, y_val_t)
00262
00263         metrics = Metrics.calculate_metrics(
00264             self.logger, y_val, val_preds, model_name=self.model_name
00265         )
00266
00267         metrics["history_fig"] = Metrics.plot_training_history(self.history, "cnn")
00268
00269         metrics["loss"] = best_val_loss
00270
00271         if "confusion_matrix" in metrics and class_names is not None:
00272             metrics["confusion_matrix_fig"] = self._generate_cm_figure(
00273                 metrics["confusion_matrix"], class_names
00274             )
00275         else:
00276             metrics["confusion_matrix_fig"] = None
00277
00278         self.save_model()
00279
```

```
00280         return self.model, metrics
00281
00282
00289     def test(self, data_dict: dict, class_names=None) -> dict:
00290         self.load_model()
00291
00292         X_test = np.array(data_dict["test_data"])
00293         y_test = np.array(data_dict["test_labels"])
00294
00295         # Reshape si nécessaire
00296         if len(X_test.shape) == 2:
00297             X_test = X_test.reshape(-1, 32, 32, 3)
00298
00299         # Conversion Tenseurs
00300         X_test_t = torch.FloatTensor(X_test).permute(0, 3, 1, 2)
00301         y_test_t = torch.LongTensor(y_test)
00302
00303         self.logger.log(
00304             f"[{self.model_name}] Testing on {len(X_test)} samples...", "INFO"
00305         )
00306
00307         # Inférence par batch
00308         test_preds = self._predict_batch(X_test_t, y_test_t)
00309
00310         # Calcul métriques
00311         metrics = Metrics.calculate_metrics(
00312             self.logger, y_test, test_preds, model_name=self.model_name
00313         )
00314
00315         if "confusion_matrix" in metrics and class_names is not None:
00316             metrics["confusion_matrix_fig"] = self._generate_cm_figure(
00317                 metrics["confusion_matrix"], class_names
00318             )
00319
00320         return metrics
00321
00322
00327     def _predict_batch(self, X_tensor, y_tensor):
00328         self.model.eval()
00329         loader = DataLoader(
00330             TensorDataset(X_tensor, y_tensor), batch_size=32, shuffle=False
00331         )
00332         all_preds = []
00333
00334         with torch.no_grad():
00335             # Une simple barre de progression pour l'inférence
00336             for batch_X, _ in tqdm(
00337                 loader,
00338                 desc="Inference",
00339                 unit="batch",
00340                 leave=False,
00341                 file=self.tqdm_out,
00342                 mininterval=1.0,
00343             ):
00344                 batch_X = batch_X.to(self.device)
00345                 outputs = self.model(batch_X)
00346                 preds = outputs.argmax(1).cpu().numpy()
00347                 all_preds.extend(preds)
00348
00349         return np.array(all_preds)
00350
00351
00356     def classify(self, image: np.ndarray) -> int:
00357         self.load_model()
00358         self.model.eval()
00359
00360         image = np.array(image)
00361         if len(image.shape) == 1:
00362             image = image.reshape(32, 32, 3)
00363
00364         image_t = torch.FloatTensor(image).permute(2, 0, 1).unsqueeze(0).to(self.device)
00365
00366         with torch.no_grad():
00367             outputs = self.model(image_t)
00368             predicted_class = outputs.argmax(1).item()
00369
00370         return predicted_class
00371
00372
00374     def save_model(self) -> None:
00375         os.makedirs("_models", exist_ok=True)
00376         torch.save(self.model.state_dict(), self.model_path)
00377         self.logger.log(f"[{self.model_name}] Model saved to {self.model_path}", "INFO")
00378
00379
00382     def load_model(self) -> None:
00383         if not os.path.exists(self.model_path):
```

```
00384             raise FileNotFoundError(f"{self.model_path} not found")
00385
00386         if self.model is None:
00387             self.model = self._create_model()
00388
00389         self.model.load_state_dict(
00390             torch.load(self.model_path, map_location=self.device)
00391         )
```

7.7 src/DecisionTreeClassifier.py File Reference

Classes

- class DecisionTreeClassifier.DecisionTreeClassifier

Namespaces

- namespace DecisionTreeClassifier

7.8 DecisionTreeClassifier.py

Go to the documentation of this file.

```
00001 from sklearn.tree import DecisionTreeClassifier as SKLearnDT
00002 from src.Model import Model
00003
00004
00005
00011 class DecisionTreeClassifier(Model):
00012
00013     def __init__(self, logger, n_train, n_val):
00014         super().__init__(logger, n_train, n_val, model_name="Decision Tree")
00015
00016
00020     def _create_model(self, **kwargs):
00021         return SKLearnDT(
00022             criterion="gini",
00023             max_depth=30,
00024             min_samples_split=20,
00025             min_samples_leaf=10,
00026             max_features="sqrt",
00027             random_state=42,
00028             **kwargs
00029         )
```

7.9 src/GBMClassifier.py File Reference

Classes

- class GBMClassifier.GBMClassifier

Namespaces

- namespace GBMClassifier

7.10 GBMClassifier.py

Go to the documentation of this file.

```
00001 from sklearn.ensemble import HistGradientBoostingClassifier
00002 from sklearn.pipeline import make_pipeline
00003 from sklearn.preprocessing import StandardScaler
00004 from src.Model import Model
00005
00006
00007
00013 class GBMClassifier(Model):
00014
00015     def __init__(self, logger, n_train, n_val):
00016         super().__init__(logger, n_train, n_val, model_name="Gradient Boosting")
00017
00018
00022     def _create_model(self, **kwargs):
00023         return make_pipeline(
00024             StandardScaler(),
00025             HistGradientBoostingClassifier(
00026                 learning_rate=0.1,
00027                 max_iter=100,
00028                 max_leaf_nodes=31,
00029                 random_state=42,
00030                 verbose=0,
00031                 **kwargs
00032             ),
00033         )
```

7.11 src/LogisticRegressionClassifier.py File Reference

Classes

- class LogisticRegressionClassifier.LogisticRegressionClassifier

Namespaces

- namespace LogisticRegressionClassifier

7.12 LogisticRegressionClassifier.py

Go to the documentation of this file.

```
00001 from sklearn.linear_model import LogisticRegression
00002 from sklearn.pipeline import make_pipeline
00003 from sklearn.preprocessing import StandardScaler
00004 from sklearn.decomposition import PCA
00005 from src.Model import Model
00006
00007
00008
00014 class LogisticRegressionClassifier(Model):
00015
00016     def __init__(self, logger, n_train, n_val):
00017         super().__init__(logger, n_train, n_val, model_name="Logistic Regression")
00018
00019
00023     def _create_model(self, **kwargs):
00024         return make_pipeline(
00025             StandardScaler(),
00026             PCA(n_components=0.95),
00027             LogisticRegression(
00028                 solver="lbfgs",
00029                 max_iter=1000,
00030                 C=1.0,
00031                 random_state=42,
00032                 n_jobs=-1,
00033                 **kwargs
00034             ),
00035         )
```

7.13 src/Model.py File Reference

Classes

- class Model.Model

Namespaces

- namespace Model

7.14 Model.py

Go to the documentation of this file.

```
00001 import os
00002 import pickle
00003 import numpy as np
00004 import matplotlib
00005 import matplotlib.pyplot as plt
00006 import seaborn as sns
00007 from abc import ABC, abstractmethod
00008 from sklearn.model_selection import train_test_split
00009
00010 matplotlib.use("Agg")
00011
00012 try:
00013     from utils.Metrics import Metrics
00014 except ImportError:
00015     from utils.Metrics import Metrics
00016
00017
00018
00024 class Model(ABC):
00025
00032     def __init__(self, logger, n_train, n_val, model_name="BaseModel"):
00033         self.logger = logger
00034         self.n_train = n_train
00035         self.n_val = n_val
00036         self.model_name = model_name
00037         self.model_path = f"_models/{model_name}.pkl"
00038         self.model = None
00039
00040
00045     @abstractmethod
00046     def _create_model(self, **kwargs):
00047         pass
00048
00049
00053     def _prepare_data(self, data_dict):
00054         X = np.array(data_dict["train_data"])
00055         y = np.array(data_dict["train_labels"])
00056
00057         limit = min(len(X), self.n_train + self.n_val)
00058         X_sub = X[:limit]
00059         y_sub = y[:limit]
00060
00061         self.logger.log(
00062             f"[{self.model_name}] Data split: {self.n_train} Train, {self.n_val} Val",
00063             "RESULT",
00064         )
00065
00066         return train_test_split(
00067             X_sub,
00068             y_sub,
00069             train_size=self.n_train,
00070             test_size=self.n_val,
00071             random_state=42,
00072             stratify=y_sub,
00073         )
00074
00075     def _generate_cm_figure(self, cm, class_names):
00076         fig = plt.figure(figsize=(10, 8))
00077         sns.heatmap(
00078             cm,
00079             annot=True,
```

```
00080             fmt="d",
00081             cmap="Blues",
00082             xticklabels=class_names,
00083             yticklabels=class_names,
00084         )
00085         plt.title(f"Confusion Matrix - {self.model_name}")
00086         plt.tight_layout()
00087         return fig
00088
00089
00096     def train(self, data_dict: dict, class_names=None) -> tuple:
00097         self.logger.log(f"[{self.model_name}] Preparing data...")
00098         X_train, X_val, y_train, y_val = self._prepare_data(data_dict)
00099
00100         # Flatten automatique sauf pour le CNN
00101         if len(X_train.shape) > 2 and self.model_name != "CNN":
00102             X_train = X_train.reshape(X_train.shape[0], -1)
00103             X_val = X_val.reshape(X_val.shape[0], -1)
00104
00105         self.model = self._create_model()
00106         self.logger.log(f"[{self.model_name}] Training started...")
00107
00108         self.model.fit(X_train, y_train)
00109
00110         self.logger.log(f"[{self.model_name}] Evaluating on validation set...")
00111         y_pred = self.model.predict(X_val)
00112
00113         metrics = Metrics.calculate_metrics(
00114             self.logger, y_val, y_pred, model_name=self.model_name
00115         )
00116
00117         if "confusion_matrix" in metrics and class_names is not None:
00118             metrics["confusion_matrix_fig"] = self._generate_cm_figure(
00119                 metrics["confusion_matrix"], class_names
00120             )
00121         else:
00122             metrics["confusion_matrix_fig"] = None
00123
00124         self.logger.log(
00125             f"[{self.model_name}] Validation Accuracy: {metrics.get('accuracy', 0):.4f}",
00126             "RESULT",
00127         )
00128         self.save_model()
00129
00130         return self.model, metrics
00131
00132
00138     def test(self, data_dict: dict, class_names=None) -> dict:
00139         self.load_model()
00140         X_test = np.array(data_dict["test_data"])
00141         y_test = np.array(data_dict["test_labels"])
00142
00143         if len(X_test.shape) > 2 and self.model_name != "CNN":
00144             X_test = X_test.reshape(X_test.shape[0], -1)
00145
00146         y_pred = self.model.predict(X_test)
00147         metrics = Metrics.calculate_metrics(self.logger, y_test, y_pred, model_name=self.model_name)
00148
00149         if "confusion_matrix" in metrics and class_names is not None:
00150             metrics["confusion_matrix_fig"] = self._generate_cm_figure(
00151                 metrics["confusion_matrix"], class_names
00152             )
00153
00154         return metrics
00155
00156
00160     def classify(self, image: np.ndarray):
00161         if self.model is None:
00162             self.load_model()
00163         image = np.array(image)
00164
00165         # Gestion du format d'entrée
00166         if len(image.shape) > 1 and self.model_name != "CNN":
00167             image = image.reshape(1, -1)
00168         elif len(image.shape) == 1 and self.model_name != "CNN":
00169             image = image.reshape(1, -1)
00170
00171         return self.model.predict(image)[0]
00172
00173
00176     def save_model(self) -> None:
00177         os.makedirs("_models", exist_ok=True)
00178         with open(self.model_path, "wb") as f:
00179             pickle.dump(self.model, f)
00180         self.logger.log(f"[{self.model_name}] Model saved to {self.model_path}")
00181
00182
```

```
00185     def load_model(self) -> None:
00186         if not os.path.exists(self.model_path):
00187             raise FileNotFoundError(f"{self.model_path} not found")
00188         with open(self.model_path, "rb") as f:
00189             self.model = pickle.load(f)
```

7.15 src/RandomForestClassifier.py File Reference

Classes

- class RandomForestClassifier.RandomForestClassifier

Namespaces

- namespace RandomForestClassifier

7.16 RandomForestClassifier.py

Go to the documentation of this file.

```
00001 from sklearn.ensemble import RandomForestClassifier as SKLearnRF
00002 from src.Model import Model
00003
00004
00005
00010 class RandomForestClassifier(Model):
00011     def __init__(self, logger, n_train, n_val):
00012         super().__init__(logger, n_train, n_val, model_name="Random Forest")
00013
00014
00018     def _create_model(self, **kwargs):
00019         return SKLearnRF(
00020             n_estimators=100,
00021             max_features="sqrt",
00022             min_samples_leaf=1,
00023             n_jobs=-1,
00024             random_state=42,
00025             **kwargs
00026         )
```

7.17 src/SVMClassifier.py File Reference

Classes

- class SVMClassifier.SVMClassifier

Namespaces

- namespace SVMClassifier

7.18 SVMClassifier.py

Go to the documentation of this file.

```
00001 from sklearn.svm import SVC
00002 from sklearn.pipeline import make_pipeline
00003 from sklearn.preprocessing import StandardScaler
00004 from sklearn.decomposition import PCA
00005 from src.Model import Model
00006
00007
00008
00014 class SVMClassifier(Model):
00015
00016     def __init__(self, logger, n_train, n_val):
00017         super().__init__(logger, n_train, n_val, model_name="SVM")
00018
00019
00023     def _create_model(self, **kwargs):
00024         return make_pipeline(
00025             StandardScaler(),
00026             PCA(n_components=0.95),
00027             SVC(
00028                 kernel="rbf",
00029                 C=1.0,
00030                 gamma="scale",
00031                 random_state=42,
00032                 probability=True,
00033                 verbose=0,
00034                 **kwargs
00035             ),
00036         )
```

7.19 utils/DataHandler.py File Reference

Classes

- class DataHandler.DataHandler

Namespaces

- namespace DataHandler

7.20 DataHandler.py

Go to the documentation of this file.

```
00001 import os
00002 import pickle
00003 import numpy as np
00004
00005 from utils.SingletonMeta import SingletonMeta
00006
00007
00008
00013 class DataHandler(metaclass=SingletonMeta):
00014
00015
00020     def __init__(self, logger, base_path: str):
00021         self.logger = logger
00022         self.base_path = base_path
00023         self.data = None
00024         self.train_data = None
00025         self.train_labels = None
00026         self.test_data = None
00027         self.test_labels = None
00028
00029         self.class_names = [
00030             "Airplane",
00031             "Automobile",
```

```
00032             "Bird",
00033             "Cat",
00034             "Deer",
00035             "Dog",
00036             "Frog",
00037             "Horse",
00038             "Ship",
00039             "Truck",
00040         ]
00041
00042         if not os.path.exists(base_path):
00043             raise FileNotFoundError(f"Directory {base_path} does not exist!")
00044
00045
00050     def unpickle(self, filename: str) -> dict:
00051         filepath = os.path.join(self.base_path, filename)
00052
00053         if not os.path.exists(filepath):
00054             raise FileNotFoundError(f"File {filepath} does not exist!")
00055
00056         with open(filepath, "rb") as fo:
00057             raw_data = pickle.load(fo, encoding="bytes")
00058
00059         return raw_data
00060
00061
00067     def load_data(self, normalize: bool = True, flatten: bool = True) -> dict:
00068         train_files = [f"data_batch_{i}" for i in range(1, 6)]
00069         test_file = "test_batch"
00070
00071         train_data = []
00072         train_labels = []
00073
00074         self.logger.log("Loading training data...", "INFO")
00075         for filename in train_files:
00076             batch = self.unpickle(filename)
00077             train_data.append(batch[b"data"])
00078             train_labels.extend(batch[b"labels"])
00079
00080         # Empiler dans un seul tableau
00081         train_data = np.vstack(train_data)  # shape (50000, 3072)
00082         train_labels = np.array(train_labels, dtype=np.int64)
00083
00084         self.logger.log("Loading test data...", "INFO")
00085         test_batch = self.unpickle(test_file)
00086
00087         test_data = np.array(test_batch[b"data"])
00088         test_labels = np.array(test_batch[b"labels"], dtype=np.int64)
00089
00090         # Redimensionner les images à (N, 3, 32, 32) si nécessaire
00091         if not flatten:
00092             train_data = train_data.reshape(-1, 3, 32, 32)
00093             test_data = test_data.reshape(-1, 3, 32, 32)
00094
00095         # Normaliser les valeurs de pixels si demandé
00096         if normalize:
00097             train_data = train_data.astype(np.float32) / 255.0
00098             test_data = test_data.astype(np.float32) / 255.0
00099         else:
00100             train_data = train_data.astype(np.float32)
00101             test_data = test_data.astype(np.float32)
00102
00103         self.train_data = train_data
00104         self.train_labels = train_labels
00105         self.test_data = test_data
00106         self.test_labels = test_labels
00107
00108         self.data = {
00109             "train_data": train_data,
00110             "train_labels": train_labels,
00111             "test_data": test_data,
00112             "test_labels": test_labels,
00113         }
00114
00115         self.logger.log(
00116             f"Data loaded: {len(train_data)} train samples, {len(test_data)} test samples",
00117             "INFO",
00118         )
00119
00120
00124     def get_train_data(self) -> tuple:
00125         if self.train_data is None:
00126             raise ValueError("Data not loaded yet. Call load_data() first.")
00127         return self.train_data, self.train_labels
00128
00129
00133     def get_test_data(self) -> tuple:
```

```
00134         if self.test_data is None:
00135             raise ValueError("Data not loaded yet. Call load_data() first.")
00136         return self.test_data, self.test_labels
00137 
00138 
00142     def get_data_dict(self) -> dict:
00143         if self.data is None:
00144             raise ValueError("Data not loaded yet. Call load_data() first.")
00145         return self.data
00146 
00147 
00153     def get_subset(self, n_train: int = None, n_test: int = None) -> dict:
00154         if self.data is None:
00155             raise ValueError("Data not loaded yet. Call load_data() first.")
00156 
00157         train_data = self.train_data[:n_train] if n_train else self.train_data
00158         train_labels = self.train_labels[:n_train] if n_train else self.train_labels
00159         test_data = self.test_data[:n_test] if n_test else self.test_data
00160         test_labels = self.test_labels[:n_test] if n_test else self.test_labels
00161 
00162         return {
00163             "train_data": train_data,
00164             "train_labels": train_labels,
00165             "test_data": test_data,
00166             "test_labels": test_labels,
00167         }
00168 
00169 
00174     def get_class_distribution(self, dataset: str = "train") -> dict:
00175         if self.data is None:
00176             raise ValueError("Data not loaded yet. Call load_data() first.")
00177 
00178         labels = self.train_labels if dataset == "train" else self.test_labels
00179         unique, counts = np.unique(labels, return_counts=True)
00180 
00181         distribution = {self.class_names[i]: count for i, count in zip(unique, counts)}
00182 
00183         self.logger.log(f"\nClass distribution ({dataset}):", "RESULT")
00184         for class_name, count in distribution.items():
00185             self.logger.log(
00186                 f"  {class_name:12s}: {count:5d} ({count / len(labels) * 100:.1f}%)",
00187                 "RESULT",
00188             )
00189 
00190         return distribution
00191 
00192 
00197     def get_sample(self, index: int, dataset: str = "train") -> tuple:
00198         if self.data is None:
00199             raise ValueError("Data not loaded yet. Call load_data() first.")
00200 
00201         if dataset == "train":
00202             return self.train_data[index], self.train_labels[index]
00203         else:
00204             return self.test_data[index], self.test_labels[index]
00205 
00206 
00211     def get_batch(self, indices: list, dataset: str = "train") -> tuple:
00212         if self.data is None:
00213             raise ValueError("Data not loaded yet. Call load_data() first.")
00214 
00215         if dataset == "train":
00216             return self.train_data[indices], self.train_labels[indices]
00217         else:
00218             return self.test_data[indices], self.test_labels[indices]
00219 
00220 
00223     def get_class_names(self):
00224         return self.class_names
00225 
00226 
00229     def get_data(self):
00230         return self.data
```

7.21 utils/functions.py File Reference

Namespaces

- namespace functions

Functions

- functions.convert_time (secondes)

7.22 functions.py

Go to the documentation of this file.

```
00001 def convert_time(secondes):
00002     heures = int(secondes // 3600)
00003     reste = secondes % 3600
00004     minutes = int(reste // 60)
00005     secondes_restantes = int(reste % 60)
00006 
00007     parts = []
00008     if heures > 0:
00009         parts.append(f"{heures}h")
00010     if minutes > 0:
00011         parts.append(f"{minutes}min")
00012     parts.append(f"{secondes_restantes}s")
00013 
00014     return " ".join(parts)
```

7.23 utils/ImageProcessor.py File Reference

Classes

- class ImageProcessor.ImageProcessor

Namespaces

- namespace ImageProcessor

7.24 ImageProcessor.py

Go to the documentation of this file.

```
00001 import numpy as np
00002 from PIL import Image
00003 
00004 
00005 
00008 class ImageProcessor:
00009 
00010 
00017     @staticmethod
00018     def load_and_preprocess(file_path: str) -> np.ndarray:
00019         try:
00020             img = Image.open(file_path).convert("RGB")
00021             img = img.resize((32, 32), Image.Resampling.LANCZOS)
00022             img_array = np.array(img)
00023             img_array = img_array.astype(np.float32) / 255.0
00024 
00025             return img_array
00026 
00027         except Exception as e:
00028             raise ValueError(f"Impossible de traiter l'image {file_path}: {str(e)}")
```

7.25 utils/Logger.py File Reference

Classes

- class Logger.Logger

Namespaces

- namespace Logger

Functions

- Logger.get_log_color (tag)

  Retourne la couleur Tkinter associée à un tag de log donné.

7.26 Logger.py

Go to the documentation of this file.

```
00001 from tkinter import *
00002 import time
00003
00004 from utils.SingletonMeta import SingletonMeta
00005
00006
00007
00011 def get_log_color(tag):
00012     match tag:
00013         case "WARNING":
00014             return "orange"
00015         case "ERROR":
00016             return "red"
00017         case "SUCCESS":
00018             return "green"
00019         case "RESULT":
00020             return "blue"
00021         case "TQDM":
00022             return "purple"
00023         case _:
00024             return "black"
00025
00026
00027
00033 class Logger(metaclass=SingletonMeta):
00034
00035
00039     def __init__(self, root, console):
00040         self.console = console
00041         self.root = root
00042
00043
00050     def log(self, message, tag="INFO", buffered=False):
00051         self.console.config(state=NORMAL)
00052         if buffered:
00053             self.console.delete("end-2l", "end-1c")
00054         self.console.tag_configure(tag, foreground=get_log_color(tag))
00055         m = time.strftime("[%H:%M:%S]", time.localtime()) + " " + message + "\n"
00056         self.console.insert(END, m, tag)
00057         self.console.see(END)
00058         self.console.config(state=DISABLED)
00059         self.root.update()
```

7.27 utils/Metrics.py File Reference

Classes

- class Metrics.Metrics

Namespaces

- namespace Metrics

7.28 Metrics.py

Go to the documentation of this file.

```
00001 import os.path
00002
00003 import numpy as np
00004 import matplotlib.pyplot as plt
00005 from matplotlib.figure import Figure
00006 import seaborn as sns
00007 from sklearn.metrics import (
00008     f1_score,
00009     confusion_matrix,
00010     classification_report,
00011     accuracy_score,
00012     precision_score,
00013     recall_score,
00014 )
00015
00016 class_names = [
00017     "airplane",
00018     "automobile",
00019     "bird",
00020     "cat",
00021     "deer",
00022     "dog",
00023     "frog",
00024     "horse",
00025     "ship",
00026     "truck",
00027 ]
00028
00029
00030
00036 class Metrics:
00037
00038
00048     @staticmethod
00049     def calculate_metrics(logger, y_true, y_pred, model_name="Model") -> dict:
00050         y_true = np.array(y_true).flatten()
00051         y_pred = np.array(y_pred).flatten()
00052
00053         print(f"METRICS FOR: {model_name}")
00054
00055         accuracy = accuracy_score(y_true, y_pred)
00056         f1_macro = f1_score(y_true, y_pred, average="macro")
00057         f1_weighted = f1_score(y_true, y_pred, average="weighted")
00058         f1_per_class = f1_score(y_true, y_pred, average=None)
00059         precision_macro = precision_score(y_true, y_pred, average="macro")
00060         recall_macro = recall_score(y_true, y_pred, average="macro")
00061
00062         logger.log(f"\nOVERALL METRICS:", "RESULT")
00063         logger.log(f"  Accuracy:           {accuracy:.4f}", "RESULT")
00064         logger.log(f"  F1 Score (Macro):   {f1_macro:.4f}", "RESULT")
00065         logger.log(f"  F1 Score (Weighted): {f1_weighted:.4f}", "RESULT")
00066         logger.log(f"  Precision (Macro):  {precision_macro:.4f}", "RESULT")
00067         logger.log(f"  Recall (Macro):     {recall_macro:.4f}", "RESULT")
00068
00069         logger.log(f"\nF1 SCORE PER CLASS:", "RESULT")
00070         for i, class_name in enumerate(class_names):
00071             logger.log(f"  {class_name:12s}: {f1_per_class[i]:.4f}", "RESULT")
00072
00073         print(f"\nCLASSIFICATION REPORT:")
00074         print(classification_report(y_true, y_pred, target_names=class_names, digits=4))
00075
00076         metrics = {
00077             "accuracy": accuracy,
00078             "f1_macro": f1_macro,
00079             "f1_weighted": f1_weighted,
00080             "f1_per_class": f1_per_class,
00081             "precision_macro": precision_macro,
00082             "recall_macro": recall_macro,
00083             "confusion_matrix": confusion_matrix(y_true, y_pred),
00084         }
00085
00086         return metrics
00087
00088
00096     @staticmethod
00097     def plot_confusion_matrix(
00098         y_true, y_pred, model_name="Model", save_path=None
00099     ) -> np.ndarray:
00100         y_true = np.array(y_true).flatten()
00101         y_pred = np.array(y_pred).flatten()
00102
00103         cm = confusion_matrix(y_true, y_pred)
```

```
00104
00105         plt.figure(figsize=(10, 8))
00106         sns.heatmap(
00107             cm,
00108             annot=True,
00109             fmt="d",
00110             cmap="Blues",
00111             xticklabels=class_names,
00112             yticklabels=class_names,
00113             cbar_kws={"label": "Count"},
00114         )
00115         plt.title(f"Confusion Matrix - {model_name}", fontsize=16, fontweight="bold")
00116         plt.ylabel("True Label", fontsize=12)
00117         plt.xlabel("Predicted Label", fontsize=12)
00118         plt.xticks(rotation=45, ha="right")
00119         plt.yticks(rotation=0)
00120         plt.tight_layout()
00121
00122         if save_path:
00123             plt.savefig(save_path, dpi=300, bbox_inches="tight")
00124             print(f"Confusion matrix saved to: {save_path}")
00125
00126         return cm
00127
00128
00135     @staticmethod
00136     def plot_training_history(
00137         history, model_name="Model", save_path="_results"
00138     ) -> Figure:
00139         if hasattr(history, "history"):
00140             history = history.history
00141
00142         fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
00143
00144         ax1.plot(history["accuracy"], label="Train Accuracy")
00145         ax1.plot(history["val_accuracy"], label="Val Accuracy")
00146         ax1.set_xlabel("Epoch")
00147         ax1.set_ylabel("Accuracy")
00148         ax1.set_title(f"{model_name} - Accuracy")
00149         ax1.legend()
00150         ax1.grid(True)
00151
00152         ax2.plot(history["loss"], label="Train Loss")
00153         ax2.plot(history["val_loss"], label="Val Loss")
00154         ax2.set_xlabel("Epoch")
00155         ax2.set_ylabel("Loss")
00156         ax2.set_title(f"{model_name} - Loss")
00157         ax2.legend()
00158         ax2.grid(True)
00159
00160         plt.tight_layout()
00161
00162         if save_path:
00163             save_name = os.path.join(save_path, model_name + ".png")
00164             plt.savefig(save_name, dpi=300, bbox_inches="tight")
00165             print(f"Training history saved to: {save_name}")
00166         return fig
00167
00168
00174     @staticmethod
00175     def compare_models(metrics_dict: dict, save_path="model_comparison.png") -> None:
00176         print("MODEL COMPARISON")
00177
00178         print(
00179             f"\n{'Model':<20} {'Accuracy':<12} {'F1 (Macro)':<12} {'F1 (Weighted)':<12}
      {'Precision':<12} {'Recall':<12}"
00180         )
00181         print("-" * 90)
00182
00183         for model_name, metrics in metrics_dict.items():
00184             print(
00185                 f"{model_name:<20} {metrics['accuracy']:<12.4f} "
00186                 f"{metrics['f1_macro']:<12.4f} {metrics['f1_weighted']:<12.4f} "
00187                 f"{metrics['precision_macro']:<12.4f} {metrics['recall_macro']:<12.4f}"
00188             )
00189
00190         fig, axes = plt.subplots(1, 3, figsize=(15, 5))
00191
00192         models = list(metrics_dict.keys())
00193         accuracies = [metrics_dict[m]["accuracy"] for m in models]
00194         f1_macros = [metrics_dict[m]["f1_macro"] for m in models]
00195         f1_weighteds = [metrics_dict[m]["f1_weighted"] for m in models]
00196
00197         axes[0].bar(models, accuracies, color="steelblue")
00198         axes[0].set_ylabel("Accuracy")
00199         axes[0].set_title("Accuracy Comparison")
00200         axes[0].tick_params(axis="x", rotation=45)
```

```
00201         axes[0].grid(axis="y", alpha=0.3)
00202
00203         axes[1].bar(models, f1_macros, color="coral")
00204         axes[1].set_ylabel("F1 Score (Macro)")
00205         axes[1].set_title("F1 Macro Comparison")
00206         axes[1].tick_params(axis="x", rotation=45)
00207         axes[1].grid(axis="y", alpha=0.3)
00208
00209         axes[2].bar(models, f1_weighteds, color="seagreen")
00210         axes[2].set_ylabel("F1 Score (Weighted)")
00211         axes[2].set_title("F1 Weighted Comparison")
00212         axes[2].tick_params(axis="x", rotation=45)
00213         axes[2].grid(axis="y", alpha=0.3)
00214
00215         plt.tight_layout()
00216         plt.savefig(save_path, dpi=300, bbox_inches="tight")
00217         print(f"\nComparison chart saved to: {save_path}")
00218
00219
00226     @staticmethod
00227     def get_precision_recall_f1(
00228         y_true, y_pred, model_name="Model", average="macro"
00229     ) -> dict:
00230         y_true = np.array(y_true).flatten()
00231         y_pred = np.array(y_pred).flatten()
00232
00233         if average is None:
00234             precision = precision_score(y_true, y_pred, average=None)
00235             recall = recall_score(y_true, y_pred, average=None)
00236             f1 = f1_score(y_true, y_pred, average=None)
00237
00238             print(f"\n{model_name} - Per-Class Metrics:")
00239             print(f"{'Class':<12} {'Precision':<12} {'Recall':<12} {'F1 Score':<12}")
00240             print("-" * 50)
00241             for i, class_name in enumerate(class_names):
00242                 print(
00243                     f"{class_name:<12} {precision[i]:<12.4f} {recall[i]:<12.4f} {f1[i]:<12.4f}"
00244                 )
00245
00246             return {
00247                 "precision_per_class": precision,
00248                 "recall_per_class": recall,
00249                 "f1_per_class": f1,
00250             }
00251         else:
00252             precision = precision_score(y_true, y_pred, average=average)
00253             recall = recall_score(y_true, y_pred, average=average)
00254             f1 = f1_score(y_true, y_pred, average=average)
00255
00256             print(f"\n{model_name} - Overall Metrics ({average}):")
00257             print(f"  Precision: {precision:.4f}")
00258             print(f"  Recall:    {recall:.4f}")
00259             print(f"  F1 Score:  {f1:.4f}")
00260
00261             return {"precision": precision, "recall": recall, "f1_score": f1}
00262
00263
00271     @staticmethod
00272     def get_predictions(model, data, model_type="sklearn") -> np.ndarray:
00273         if model_type == "pytorch":
00274             import torch
00275
00276             device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
00277             model.eval()
00278
00279             if len(data.shape) == 2:
00280                 data = data.reshape(-1, 32, 32, 3)
00281             data_tensor = torch.FloatTensor(data).permute(0, 3, 1, 2).to(device)
00282
00283             with torch.no_grad():
00284                 outputs = model(data_tensor)
00285                 predictions = outputs.argmax(1).cpu().numpy()
00286
00287             return predictions
00288
00289         elif model_type == "keras":
00290             if len(data.shape) == 2:
00291                 data = data.reshape(-1, 32, 32, 3)
00292             predictions = model.predict(data, verbose=0)
00293             predictions = np.argmax(predictions, axis=1)
00294         else:
00295             if len(data.shape) > 2:
00296                 data = data.reshape(data.shape[0], -1)
00297             predictions = model.predict(data)
00298
00299         return predictions
```

7.29 utils/SingletonMeta.py File Reference

Classes

- class SingletonMeta.SingletonMeta

Namespaces

- namespace SingletonMeta

7.30 SingletonMeta.py

Go to the documentation of this file.

```
00001
00007 class SingletonMeta(type):
00008     _instances = {}
00009
00010
00019     def __call__(cls, *args, **kwargs):
00020         if cls not in cls._instances:
00021             instance = super().__call__(*args, **kwargs)
00022             cls._instances[cls] = instance
00023         return cls._instances[cls]
```

7.31 utils/TqdmToLogger.py File Reference

Classes

- class TqdmToLogger.TqdmToLogger

Namespaces

- namespace TqdmToLogger

7.32 TqdmToLogger.py

Go to the documentation of this file.

```
00001
00006 class TqdmToLogger:
00007
00011     def __init__(self, logger, tag="INFO"):
00012         self.__logger = logger
00013         self.__tag = tag
00014         self.__first_line = True
00015
00016
00022     def write(self, buf):
00023         if buf.strip():
00024             self.__logger.log(buf.strip(), self.__tag, buffered=not self.__first_line)
00025             self.__first_line = False
00026
00027
00030     def flush(self):
00031         pass
```

7.33 Window.py File Reference

Classes

- class Window.Window

Namespaces

- namespace Window

7.34 Window.py

Go to the documentation of this file.

```
00001 import os
00002 import tkinter as tk
00003 from tkinter import *
00004 from tkinter import ttk
00005 from tkinter import filedialog as fd
00006 from PIL import Image, ImageTk
00007 import numpy as np
00008
00009 TEST = True
00010
00011
00012
00018 class Window:
00019
00023     def __init__(self):
00024         os.makedirs("_models", exist_ok=True)
00025
00026         self.__root = Tk()
00027         self.__root.title("Classifier interface")
00028         self.__root.geometry("1920x1080")
00029         self.__root.resizable(width=True, height=True)
00030         self.__root.grid_columnconfigure(0, weight=1)
00031         self.__root.grid_columnconfigure(1, weight=6)
00032         self.__root.grid_rowconfigure(0, weight=1)
00033
00034         style = ttk.Style()
00035
00036         # Forçage du thème "clam" pour forcer le mode clair sur macOS
00037         style.theme_use('clam')
00038
00039         self.__root.option_add("*foreground", "black")
00040         self.__root.option_add("*background", "#f0f0f0")
00041
00042         self.__root.option_add("*Entry.background", "white")
00043         self.__root.option_add("*Entry.foreground", "black")
00044         self.__root.option_add("*Text.background", "white")
00045         self.__root.option_add("*Text.foreground", "black")
00046         self.__root.option_add("*Button.foreground", "black")
00047
00048         style.configure(".", foreground="black", background="#f0f0f0")
00049         style.configure("TLabel", foreground="black", background="#f0f0f0")
00050         style.configure("TNotebook", background="#f0f0f0")
00051         style.configure("TNotebook.Tab", foreground="black")
00052
00053         self.__models = "_models"
00054
00055         self.__form = Frame(self.__root)
00056         self.__form.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)
00057
00058         self.__tab_system = ttk.Notebook(self.__root)
00059         self.__tab_system.grid(row=0, column=1, sticky="nsew", padx=5, pady=5)
00060
00061         self.__logs_tab = Frame(self.__tab_system)
00062         self.__logs_tab.grid_columnconfigure(1, weight=1)
00063         self.__tab_system.add(self.__logs_tab, text="logs")
00064
00065         self.__console = tk.Text(self.__logs_tab, wrap=WORD)
00066         self.__scrollbar = tk.Scrollbar(self.__logs_tab, command=self.__console.yview)
00067         self.__console.config(yscrollcommand=self.__scrollbar.set)
00068         self.__console.grid(row=0, column=1, sticky="nsew", padx=5, pady=5)
00069         self.__scrollbar.grid(row=0, column=1, sticky="nse", padx=5, pady=5)
```

```
00070
00071         self.__select_model_area = LabelFrame(
00072             self.__form,
00073             text="Select model",
00074             padx=15,
00075             pady=15,
00076             font=("Arial", 10, "bold"),
00077         )
00078         self.__select_model_area.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)
00079         self.__select_model_area.grid_columnconfigure(1, weight=1)
00080
00081         self.__model_label = Label(
00082             self.__select_model_area,
00083             text="Select model:",
00084             font=("Arial", 9),
00085             width=18,
00086             anchor="w",
00087         )
00088         self.__model_label.grid(row=0, column=0, padx=5, pady=8, sticky="w")
00089         self.__model_information = Label(
00090             self.__select_model_area,
00091             text="Model selected: CNN",
00092             font=("Arial", 10),
00093             bg="#e3f2fd",
00094             relief="solid",
00095             borderwidth=1,
00096             padx=10,
00097             pady=5,
00098         )
00099         self.__model_information.grid(
00100             row=4, column=0, columnspan=2, padx=5, pady=(0, 10), sticky="ew"
00101         )
00102
00103         self.__model_var = tk.StringVar()
00104         self.__model_dropdown = ttk.Combobox(
00105             self.__select_model_area, textvariable=self.__model_var, font=("Arial", 9)
00106         )
00107         self.__model_dropdown["values"] = [
00108             "CNN",
00109             "Decision Tree",
00110             "Random Forest",
00111             "Logistic Regression",
00112             "SVM",
00113             "Gradient Boosting",
00114         ]
00115         self.__model_dropdown.current(0)
00116         self.__model_dropdown.grid(row=0, column=1, padx=5, pady=8, sticky="ew")
00117         self.__model_dropdown.bind("«ComboboxSelected»", self.update_model_infoupdate_model_info)
00118
00119         # ===== TRAINING AREA =====
00120         self.__train_area = LabelFrame(
00121             self.__form, text="Train Data", padx=15, pady=15, font=("Arial", 10, "bold")
00122         )
00123         self.__train_area.grid(row=1, column=0, sticky="nsew", padx=10, pady=10)
00124         self.__train_area.grid_columnconfigure(1, weight=1)
00125
00126         # Dataset folder
00127         self.__dataset_folder_label = Label(
00128             self.__train_area,
00129             text="Dataset folder:",
00130             font=("Arial", 9),
00131             width=18,
00132             anchor="w",
00133         )
00134         self.__dataset_folder = tk.StringVar()
00135         self.__dataset_folder_entry = Entry(
00136             self.__train_area, textvariable=self.__dataset_folder, font=("Arial", 9)
00137         )
00138         self.__dataset_folder_button = Button(
00139             self.__train_area, text="Browse", width=10, command=self.select_dataset
00140         )
00141         self.__dataset_folder_label.grid(row=0, column=0, padx=5, pady=8, sticky="w")
00142         self.__dataset_folder_entry.grid(row=0, column=1, padx=5, pady=8, sticky="ew")
00143         self.__dataset_folder_button.grid(row=0, column=2, padx=5, pady=8)
00144
00145         # Training data size
00146         self.__train_data_label = Label(
00147             self.__train_area,
00148             text="Training data size:",
00149             font=("Arial", 9),
00150             width=18,
00151             anchor="w",
00152         )
00153         self.__train_data = tk.StringVar()
00154         self.__train_data_entry = Entry(
00155             self.__train_area,
00156             textvariable=self.__train_data,
```

```
00157             font=("Arial", 9),
00158             width=15,
00159         )
00160         self.__train_data_label.grid(row=1, column=0, padx=5, pady=8, sticky="w")
00161         self.__train_data_entry.grid(row=1, column=1, padx=5, pady=8, sticky="w")
00162 
00163         # Validation data size
00164         self.__val_data_label = Label(
00165             self.__train_area,
00166             text="Validation data size:",
00167             font=("Arial", 9),
00168             width=18,
00169             anchor="w",
00170         )
00171         self.__val_data = tk.StringVar()
00172         self.__val_data_entry = Entry(
00173             self.__train_area, textvariable=self.__val_data, font=("Arial", 9), width=15
00174         )
00175         self.__val_data_label.grid(row=2, column=0, padx=5, pady=8, sticky="w")
00176         self.__val_data_entry.grid(row=2, column=1, padx=5, pady=8, sticky="w")
00177 
00178         # Buttons frame for training
00179         self.__train_buttons_frame = Frame(self.__train_area)
00180         self.__train_buttons_frame.grid(
00181             row=3, column=0, columnspan=2, pady=(15, 5), sticky="ew"
00182         )
00183         self.__train_buttons_frame.grid_columnconfigure(0, weight=1)
00184         self.__train_buttons_frame.grid_columnconfigure(1, weight=1)
00185         self.__train_button = Button(
00186             self.__train_buttons_frame,
00187             text="Train Model",
00188             font=("Arial", 10, "bold"),
00189             bg="#4CAF50",
00190             fg="white",
00191             height=2,
00192         )
00193         self.__train_button.grid(row=0, column=0, padx=5, sticky="ew")
00194 
00195         # ===== TEST AREA =====
00196         self.__test_area = LabelFrame(
00197             self.__form, text="Test Data", padx=15, pady=15, font=("Arial", 10, "bold")
00198         )
00199         self.__test_area.grid(row=2, column=0, sticky="nsew", padx=10, pady=10)
00200         self.__test_area.grid_columnconfigure(1, weight=1)
00201 
00202         self.__test_buttons_frame = Frame(self.__test_area)
00203         self.__test_buttons_frame.grid(
00204             row=0, column=0, columnspan=2, pady=(15, 5), sticky="ew"
00205         )
00206         self.__test_buttons_frame.grid_columnconfigure(0, weight=1)
00207         self.__test_buttons_frame.grid_columnconfigure(1, weight=1)
00208         self.__test_button = Button(
00209             self.__test_buttons_frame,
00210             text="Test Model",
00211             font=("Arial", 10, "bold"),
00212             bg="#4CAF50",
00213             fg="white",
00214             height=2,
00215         )
00216         self.__test_button.grid(row=0, column=0, padx=5, sticky="ew")
00217 
00218         # ===== INFERENCE AREA =====
00219         self.__inference_area = LabelFrame(
00220             self.__form, text="Inference", padx=15, pady=15, font=("Arial", 10, "bold")
00221         )
00222         self.__inference_area.grid(row=3, column=0, sticky="nsew", padx=10, pady=10)
00223         self.__inference_area.grid_columnconfigure(1, weight=1)
00224 
00225         # Inference data type
00226         self.__inference_data_type_label = Label(
00227             self.__inference_area,
00228             text="Inference data type:",
00229             font=("Arial", 9),
00230             width=18,
00231             anchor="w",
00232         )
00233         self.__is_file = tk.BooleanVar()
00234         self.__inference_type_frame = Frame(self.__inference_area)
00235 
00236         self.__inference_data_type_file = ttk.Radiobutton(
00237             self.__inference_type_frame,
00238             text="File",
00239             variable=self.__is_file,
00240             value=True,
00241         )
00242         self.__inference_data_type_folder = ttk.Radiobutton(
00243             self.__inference_type_frame,
```

```
00244             text="Folder",
00245             variable=self.__is_file,
00246             value=False,
00247         )
00248
00249         self.__inference_data_type_file.pack(side="left", padx=5)
00250         self.__inference_data_type_folder.pack(side="left", padx=5)
00251
00252         self.__inference_data_type_label.grid(
00253             row=0, column=0, padx=5, pady=8, sticky="w"
00254         )
00255         self.__inference_type_frame.grid(row=0, column=1, padx=5, pady=8, sticky="w")
00256
00257         # Inference data path
00258         self.__inference_data_label = Label(
00259             self.__inference_area,
00260             text="Inference data:",
00261             font=("Arial", 9),
00262             width=18,
00263             anchor="w",
00264         )
00265         self.__inference_data = tk.StringVar()
00266         self.__inference_data_entry = Entry(
00267             self.__inference_area, textvariable=self.__inference_data, font=("Arial", 9)
00268         )
00269         self.__inference_data_button = Button(
00270             self.__inference_area,
00271             text="Browse",
00272             width=10,
00273             command=self.select_inference_data,
00274         )
00275
00276         self.__inference_data_label.grid(row=1, column=0, padx=5, pady=8, sticky="w")
00277         self.__inference_data_entry.grid(row=1, column=1, padx=5, pady=8, sticky="ew")
00278         self.__inference_data_button.grid(row=1, column=2, padx=5, pady=8)
00279
00280         self.__inference_buttons_frame = Frame(self.__inference_area)
00281         self.__inference_buttons_frame.grid(
00282             row=2, column=0, columnspan=2, pady=(15, 5), sticky="ew"
00283         )
00284         self.__inference_buttons_frame.grid_columnconfigure(0, weight=1)
00285         self.__inference_buttons_frame.grid_columnconfigure(1, weight=1)
00286         self.__inference_button = Button(
00287             self.__inference_buttons_frame,
00288             text="Classify Image(s)",
00289             font=("Arial", 10, "bold"),
00290             bg="#4CAF50",
00291             fg="white",
00292             height=2,
00293         )
00294         self.__inference_button.grid(row=0, column=0, padx=5, sticky="ew")
00295
00296         self.__error_label = Label(self.__form, text="", font=("Arial", 10, "bold"))
00297         self.__error_label.grid(row=4, column=0, padx=5, pady=5, sticky="ew")
00298         if not os.path.exists(os.path.join(self.__models, "CNN.pth")):
00299             self.__error_label.config(
00300                 text="Selected model (CNN) does not exist, please train it first !",
00301                 fg="red",
00302             )
00303
00304         if TEST:
00305             self.__dataset_folder.set(
00306                 "data/cifar-10-batches-py"
00307             )
00308             self.__train_data.set("42000")
00309             self.__val_data.set("8000")
00310             self.__inference_data.set(
00311                 "test_classify"
00312             )
00313
00314
00317     def get_root(self):
00318         return self.__root
00319
00320
00323     def get_console(self):
00324         return self.__console
00325
00326
00329     def get_train_button(self):
00330         return self.__train_button
00331
00332
00335     def get_dataset_folder(self):
00336         return self.__dataset_folder
00337
00338
```

```
00341     def get_train_data(self):
00342         return self.__train_data
00343
00344
00347     def get_val_data(self):
00348         return self.__val_data
00349
00350
00353     def get_test_button(self):
00354         return self.__test_button
00355
00356
00359     def get_inference_button(self):
00360         return self.__inference_button
00361
00362
00366     def update_model_info(self, event):
00367         selected_model = self.__model_var.get()
00368         self.__model_information.config(text=f"Model selected: {selected_model}")
00369         self.__error_label.config(text="", fg="red")
00370         if selected_model == "CNN":
00371             if not os.path.exists(os.path.join(self.__models, selected_model + ".pth")):
00372                 self.__error_label.config(
00373                     text="Selected model ("
00374                     + selected_model
00375                     + ") does not exist, please train it first !",
00376                     fg="red",
00377                 )
00378         else:
00379             if not os.path.exists(os.path.join(self.__models, selected_model + ".pkl")):
00380                 self.__error_label.config(
00381                     text="Selected model ("
00382                     + selected_model
00383                     + ") does not exist, please train it first !",
00384                     fg="red",
00385                 )
00386
00387
00395     def create_training_results_tab(
00396         self, model_name, metrics_data, confusion_matrix_fig=None, history=None
00397     ):
00398         results_tab = Frame(self.__tab_system)
00399         results_tab.grid_columnconfigure(0, weight=1)
00400         results_tab.grid_rowconfigure(1, weight=1)
00401
00402         title_label = Label(
00403             results_tab,
00404             text=f"Résultats d'entraînement - {model_name}",
00405             font=("Arial", 14, "bold"),
00406             bg="#e8f5e9",
00407             pady=10,
00408         )
00409         title_label.grid(row=0, column=0, sticky="ew", padx=10, pady=10)
00410
00411         canvas = tk.Canvas(results_tab, bg="white")
00412         scrollbar = tk.Scrollbar(results_tab, orient="vertical", command=canvas.yview)
00413         scrollable_frame = Frame(canvas, bg="white")
00414
00415         scrollable_frame.bind(
00416             "<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
00417         )
00418
00419         canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
00420         canvas.configure(yscrollcommand=scrollbar.set)
00421
00422         metrics_frame = LabelFrame(
00423             scrollable_frame,
00424             text="Métriques principales",
00425             font=("Arial", 11, "bold"),
00426             padx=20,
00427             pady=15,
00428             bg="white",
00429         )
00430         metrics_frame.pack(fill="x", padx=20, pady=10)
00431
00432         metrics_to_display = [
00433             ("Accuracy", metrics_data.get("accuracy", 0)),
00434             ("Precision", metrics_data.get("precision_macro", 0)),
00435             ("Recall", metrics_data.get("recall_macro", 0)),
00436             ("F1-Score", metrics_data.get("f1_macro", 0)),
00437         ]
00438
00439         for i, (metric_name, value) in enumerate(metrics_to_display):
00440             metric_container = Frame(
00441                 metrics_frame,
00442                 bg="#e3f2fd",
00443                 relief="solid",
```

```
00444                 borderwidth=1,
00445                 padx=15,
00446                 pady=10,
00447             )
00448             metric_container.grid(
00449                 row=i // 2, column=i % 2, padx=10, pady=10, sticky="ew"
00450             )
00451
00452             Label(
00453                 metric_container,
00454                 text=metric_name,
00455                 font=("Arial", 10),
00456                 bg="#e3f2fd",
00457                 anchor="w",
00458             ).pack(anchor="w")
00459             Label(
00460                 metric_container,
00461                 text=f"{value:.4f}",
00462                 font=("Arial", 12, "bold"),
00463                 bg="#e3f2fd",
00464                 fg="#1976d2",
00465             ).pack(anchor="w")
00466
00467         metrics_frame.grid_columnconfigure(0, weight=1)
00468         metrics_frame.grid_columnconfigure(1, weight=1)
00469
00470         if "classification_report" in metrics_data:
00471             report_frame = LabelFrame(
00472                 scrollable_frame,
00473                 text="Rapport de classification détaillé",
00474                 font=("Arial", 11, "bold"),
00475                 padx=20,
00476                 pady=15,
00477                 bg="white",
00478             )
00479             report_frame.pack(fill="both", expand=True, padx=20, pady=10)
00480
00481             report_text = tk.Text(
00482                 report_frame, wrap=WORD, height=15, font=("Courier", 9)
00483             )
00484             report_scrollbar = tk.Scrollbar(report_frame, command=report_text.yview)
00485             report_text.config(yscrollcommand=report_scrollbar.set)
00486
00487             report_text.insert("1.0", metrics_data["classification_report"])
00488             report_text.config(state="disabled")
00489
00490             report_text.pack(side="left", fill="both", expand=True)
00491             report_scrollbar.pack(side="right", fill="y")
00492
00493         if confusion_matrix_fig:
00494             cm_frame = LabelFrame(
00495                 scrollable_frame,
00496                 text="Matrice de confusion",
00497                 font=("Arial", 11, "bold"),
00498                 padx=20,
00499                 pady=15,
00500                 bg="white",
00501             )
00502             cm_frame.pack(fill="both", expand=True, padx=20, pady=10)
00503
00504             try:
00505                 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
00506
00507                 canvas_plot = FigureCanvasTkAgg(confusion_matrix_fig, master=cm_frame)
00508                 canvas_plot.draw()
00509                 canvas_plot.get_tk_widget().pack(fill="both", expand=True)
00510             except ImportError:
00511                 Label(
00512                     cm_frame,
00513                     text="Matplotlib backend non disponible",
00514                     font=("Arial", 10),
00515                     fg="red",
00516                 ).pack()
00517
00518         if history:
00519             hist_frame = LabelFrame(
00520                 scrollable_frame,
00521                 text="CNN history",
00522                 font=("Arial", 11, "bold"),
00523                 padx=20,
00524                 pady=15,
00525                 bg="white",
00526             )
00527             hist_frame.pack(fill="both", expand=True, padx=20, pady=10)
00528
00529             try:
00530                 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

```

                canvas_plot = FigureCanvasTkAgg(history, master=hist_frame)
                canvas_plot.draw()
                canvas_plot.get_tk_widget().pack(fill="both", expand=True)
            except ImportError:
                Label(
                    hist_frame,
                    text="Matplotlib backend non disponible",
                    font=("Arial", 10),
                    fg="red",
                ).pack()

        button_frame = Frame(scrollable_frame, bg="white")
        button_frame.pack(fill="x", padx=20, pady=20)

        close_button = Button(
            button_frame,
            text="Fermer cet onglet",
            font=("Arial", 10, "bold"),
            bg="#f44336",
            fg="white",
            command=lambda: self.close_tab(results_tab),
        )
        close_button.pack(side="right")

        canvas.grid(row=1, column=0, sticky="nsew", padx=(10, 0), pady=(0, 10))
        scrollbar.grid(row=1, column=1, sticky="ns", pady=(0, 10))

        tab_name = f"Training: {model_name}"
        self.__tab_system.add(results_tab, text=tab_name)
        self.__tab_system.select(results_tab)

        self.get_console().insert(
            tk.END, f"\n[INFO] Onglet de résultats créé: {tab_name}"
        )
        self.get_console().see(tk.END)


    def create_inference_results_tab(self, results):
        results_tab = Frame(self.__tab_system)
        results_tab.grid_columnconfigure(0, weight=1)
        results_tab.grid_rowconfigure(1, weight=1)

        title_label = Label(
            results_tab,
            text=f"Résultats d'inférence ({len(results)} images)",
            font=("Arial", 14, "bold"),
            bg="#e1f5fe",
            pady=10,
        )
        title_label.grid(row=0, column=0, sticky="ew", padx=10, pady=10)

        canvas = tk.Canvas(results_tab, bg="white")
        scrollbar = tk.Scrollbar(results_tab, orient="vertical", command=canvas.yview)
        scrollable_frame = Frame(canvas, bg="white")

        frame_window_id = canvas.create_window(
            (0, 0), window=scrollable_frame, anchor="nw"
        )

        def on_canvas_configure(event):
            canvas.itemconfig(frame_window_id, width=event.width)

        def on_frame_configure(event):
            canvas.configure(scrollregion=canvas.bbox("all"))

        canvas.bind("<Configure>", on_canvas_configure)
        scrollable_frame.bind("<Configure>", on_frame_configure)
        canvas.configure(yscrollcommand=scrollbar.set)

        scrollable_frame.image_refs = []

        headers_frame = Frame(scrollable_frame, bg="#eeeeee", pady=5)
        headers_frame.pack(fill="x", padx=0, pady=0)

        headers_frame.grid_columnconfigure(
            1, weight=1
        )  # La colonne "Fichier" prendra l'espace

        Label(
            headers_frame,
            text="Image",
            width=10,
            font=("Arial", 10, "bold"),
            bg="#eeeeee",
            anchor="center",
        ).grid(row=0, column=0, padx=10)
```

```
        Label(
            headers_frame,
            text="Fichier",
            font=("Arial", 10, "bold"),
            bg="#eeeeee",
            anchor="w",
        ).grid(row=0, column=1, padx=10, sticky="w")
        Label(
            headers_frame,
            text="Prédiction",
            width=15,
            font=("Arial", 10, "bold"),
            bg="#eeeeee",
            anchor="center",
        ).grid(row=0, column=2, padx=10)

        for filename, prediction, img_array in results:
            row_frame = Frame(
                scrollable_frame, bg="white", pady=10, borderwidth=1, relief="solid"
            )
            row_frame.pack(fill="x", padx=10, pady=5)

            row_frame.grid_columnconfigure(1, weight=1)

            img_data = (img_array * 255).astype(np.uint8)
            pil_img = Image.fromarray(img_data)
            pil_img = pil_img.resize((64, 64), Image.Resampling.NEAREST)
            tk_img = ImageTk.PhotoImage(pil_img)

            scrollable_frame.image_refs.append(tk_img)

            img_label = Label(row_frame, image=tk_img, bg="white")
            img_label.grid(row=0, column=0, padx=10)

            name_label = Label(
                row_frame, text=filename, anchor="w", font=("Arial", 10), bg="white"
            )
            name_label.grid(row=0, column=1, padx=10, sticky="ew")

            pred_label = Label(
                row_frame,
                text=prediction,
                width=15,
                font=("Arial", 11, "bold"),
                fg="#2e7d32",
                bg="white",
            )
            pred_label.grid(row=0, column=2, padx=10)

        button_frame = Frame(results_tab)
        button_frame.grid(row=2, column=0, pady=10, sticky="e", padx=20)
        close_button = Button(
            button_frame, text="Fermer", command=lambda: self.close_tab(results_tab)
        )
        close_button.pack()

        canvas.grid(row=1, column=0, sticky="nsew", padx=(10, 0), pady=(0, 10))
        scrollbar.grid(row=1, column=1, sticky="ns", pady=(0, 10))

        self.__tab_system.add(results_tab, text="Inférence")
        self.__tab_system.select(results_tab)


    def select_dataset(self):
        self.__dataset_folder.set(fd.askdirectory(title="Select dataset folder"))


    def close_tab(self, tab):
        self.__tab_system.forget(tab)


    def get_tab_system(self):
        return self.__tab_system


    def get_selected_model(self):
        return self.__model_var.get()


    def select_inference_data(self):
        if self.__is_file:
            self.__inference_data.set(
                fd.askopenfilename(
                    filetypes=[("png files", "*.png"), ("jpg files", "*.jpg")]
                )
            )
        else:
```

```
00716             self.__inference_data.set(fd.askdirectory(title="Select dataset folder"))
00717
00718
00721     def get_inference_data(self):
00722         return self.__inference_data
00723
00724
00727     def run(self):
00728         self.__root.mainloop()
```