MyApp Class Documentation

La classe MyApp représente l'application principale de l'interface.

Fonctionnalités

- Gestion des menus déroulants de l'application
- Gestion des actions des outils
- Gestion des onglets de la fenêtre principale
- Connexion des actions aux méthodes de la fenêtre principale

Méthodes

- _createMenuBar : Crée la barre de menu de l'application avec les menus déroulants et les actions associées.
- _createToolBars : Crée la barre d'outils de l'application avec les actions associées.
- _createAction : Crée les actions de l'application.
- createConnectActions : Connecte les actions aux méthodes correspondantes de la fenêtre principale.
- handle_tab_action : Gère les actions des onglets en appelant les méthodes correspondantes de l'interface active.
- add_new_tab : Ajoute un nouvel onglet à l'application.

Classe associée : CloseableTabWidget

Description : Une sous-classe de QTabWidget qui permet de fermer les onglets et de gérer dynamiquement l'ajout et la suppression d'onglets.

Méthodes

- close_tab : Ferme l'onglet à l'index spécifié.
- add_new_tab : Ajoute un nouvel onglet à CloseableTabWidget.

MainWindow Class Documentation

La classe MainWindow représente la fenêtre principale de l'application.

Fonctionnalités

- Création de la disposition de la fenêtre
- Gestion de la scène principale
- Gestion du terminal
- Gestion de la liste des objets détectés
- Gestion de la bibliothèque de fichiers

Méthodes

- _createLayouts : Crée les différents layouts pour organiser les widgets dans la fenêtre.
- setupUI : Met en place l'interface utilisateur en créant les différents widgets et en les organisant dans les layouts.
- openPicture : Ouvre le chemin de l'image mis en paramètre dans la scène principale.
- SavePicture : Sauvegarde l'image de la scène principale dans son disque
- load_list_widget : Charge les éléments d'un dictionnaire dans la liste des classes détectées.
- read_txt_file : Lit un fichier .txt contenant les résultats de détection et extrait les informations dans un dictionnaire.
- highlightRectangle : Met en surbrillance le rectangle correspondant à l'élément sélectionné dans la liste des classes détectées.

Classes associées :

TerminalTextEdit

Description : Cette classe représente la zone de texte du terminal.

Méthodes

- normalOutputWritten : Affiche le texte dans la zone de texte du terminal.
- openTerminalWindow : Gestion de l'affichage/cachage du terminal.

CustomTreeView

Description : Cette classe représente le QTreeView utilisé pour afficher l'arborescence des fichiers.

Méthodes

- loadDir : Charge le contenu d'un dossier dans l'arborescence.
- itemDoubleClicked : Gère le double click sur l'arbre
- openTree_view : Ouvre l'élement sélectionné dans la scène principale

CustomGroupBox

Description : Cette classe représente un QGroupBox personnalisé pour la scène secondaire.

Méthodes

- on_close_button_clicked : Gère le clic sur le bouton de fermeture du groupe.

CustomGroupBoxOverride

Documentation : Cette classe représente un QGroupBox personnalisé pour la scène principale.

CustomGraphicsView

Description : Cette classe représente la vue graphique utilisée pour afficher la scène.

Méthodes

- enterEvent : Gère l'événement de survol de la souris sur la vue.
- leaveEvent : Gère l'événement de sortie de la souris de la vue.
- zoom_in : Gére le zoom sur la scène
- zoom-out : Gére le dézoom sur la scène
- Crop : Gére le crop sur la scène
- CropZoom : Gére le cropzoom sur la scène

Classe associée : CropResultDialog

Description : Affiche l'image recadrée dans une vue graphique avec la possibilité de l'enregistrer.

Méthodes :

- saveImage : Ouvre une boîte de dialogue de sauvegarde de fichier pour enregistrer sur son disque

SecondaryWindow Class Documentation

La classe SecondaryWindow représente la fenêtre secondaire de l'interface, qui affiche les paramètres spécifiques en fonction du type d'analyse d'images sélectionné.

Fonctionnalités

- Affichage des options générales pour chaque type d'analyse.
- Affichage des paramètres spécifiques en fonction du type d'analyse choisi.
- Exécution du programme sélectionné avec les paramètres spécifiés.

Méthodes

- setupOption : Construit les options disponibles dans la combobox algorithme.
- run_prog : Lance le programme choisi par l'utilisateur en fonction des paramètres sélectionnés.
- runYolo_detection : Lance le thread YOLO pour la détection.
- runYolo_segmentation : Lance le thread de segmentation de YOLO.
- runSegmentation : Lance le thread de segmentation.
- opentable_segmentation : Ouvre la classe TableWidget_Dialog qui affiche la table des paramètres de la segmentation.

- openHelp_segmentation : Ouvre la classe HelpDialog_segmentation qui affiche l'aide relative à la segmentation.
- openHelp_detection : Ouvre la classe HelpDialog_detection qui affiche l'aide relative à la détection.

Classes associées :

TableWidget_Dialog

Description : Cette fenêtre affiche une table permettant de gérer les paramètres sous forme de paires clé-valeur.

Méthodes

- addRow : Ajoute une nouvelle ligne à la table.
- deleteRow : Supprime les lignes sélectionnées de la table.
- loadData : Charge les données d'un fichier XML dans la table.
- saveData : Sauvegarde les données de la table dans un fichier XML.

Help

Description : Contient les méthodes pour la création des tableaux d'informations sur la configuration des algorithme.

ProgressDialog :

Description : Une boîte de dialogue qui affiche la progression d'une opération avec une étiquette de progression et une barre de progression.

Signaux :

- progress_updated : Signal émis lorsque la valeur de progression est mise à jour.
- progressbar_updated : Signal émis lorsque la valeur de la barre de progression est mise à jour

Méthodes :

- update_progress : Met à jour la valeur de progression et l'étiquette de progression.
- update_progressbar : Met à jour la valeur de la barre de progression.

Ces instructions vous permettront d'ajouter de nouveaux modèles et algorithmes à l'interface graphique en respectant la structure et les noms de dossiers nécessaires pour une intégration réussie.

1. Ajout d'un nouveau modèle YOLO à l'interface graphique

Placez le modèle dans le dossier « Model » correspondant à l'algorithme.

2. Ajout d'un nouveau programme à Segnet/Rednet

Ajoutez le fichier du programme dans le dossier « Program » correspondant à l'algorithme.

3. Ajout de nouveaux poids à Segnet/Rednet

Placez le fichier de poids dans le dossier « Weights » correspondant à l'algorithme.

4. Ajouter une nouvelle version de Yolo à l'interface graphique (Exemple avec la version v9 de Yolo)

1. Ajoutez un nouvel élément ("YoloV9") dans la méthode "setupOption" à l'index 0 (Object Detection) de la classe "SecondaryWindow".
2. Créez un dossier "Model_YoloV9" dans le dossier "Model". Le nom du dossier doit correspondre à l'élément ajouté ("YoloV9").
3. Ajoutez les modèles associés à l'algorithme dans le dossier "Model_YoloV9".

« Note : Le même principe s'applique pour l'ajout de la segmentation d'instance de Yolo. Ajouter un nouvel élément ("YoloV9_Seg") dans la méthode "setupOption" à l'index 2 (Instance segmentation) de la classe "SecondaryWindow", puis suivez les étapes 2 et 3 . »

5. Ajout d'un nouvel algorithme pour la segmentation d'instance à l'interface graphique (Exemple avec U-Net)

1. Ajoutez un nouvel élément ("U-Net") dans la méthode "setupOption" à l'index 2 (Semantic segmentation) de la classe "SecondaryWindow".
2. Créez un dossier "Program_U-Net" dans le dossier "Program". Le nom du dossier doit correspondre à l'élément ajouté ("U-Net").
3. Ajoutez les programmes associés à l'algorithme dans le dossier "Program_U-Net".
4. Créez un dossier "Weights_U-Net" dans le dossier "Weights". Le nom du dossier doit correspondre à l'élément ajouté ("U-Net").
5. Ajoutez les poids associés à l'algorithme dans le dossier "U-Net".
6. Modifiez la méthode "runSegmentation" de la classe "SecondaryWindow" pour définir le programme par défaut souhaité. Ajoutez une condition pour "U-Net" avec les commandes appropriées pour exécuter l'algorithme.