Analyse complète des blocs de compétences et validation par le projet WildLens

Je vais analyser systématiquement chaque critère de chaque bloc et expliquer comment votre projet WildLens le valide, avec des preuves tirées de votre code.

BLOC E6.1: Créer un modèle de données d'une solution I.A en utilisant des méthodes de Data science

1. Définir les sources et les outils nécessaires pour permettre de collecter les données

Preuves de validation:

- Configuration des chemins de données dans backend/config.py: DATA_DIR, MAMMALS_DIR, CSV_PATH
- Structure de dossiers définie pour stocker les données d'empreintes (data/Mammiferes)
- Liste des espèces dans SPECIES_LIST = ["Castor", "Chat", "Chien", "Coyote", "Ecureuil", "Lapin", "Loup", "Ours", "Rat", "Renard"]
- Méthodologie d'analyse dans load_data() de backend/models/data_preparation.py

2. Recueillir de manière sécurisée les informations à partir de sources adaptées

Preuves de validation:

- Vérification d'existence des données dans load_data(): if not os.path.exists(MAMMALS_DIR): raise FileNotFoundError
- Collecte organisée par espèce: for species in species_list: species_dir = os.path.join(MAMMALS_DIR, species)
- Gestion des chemins d'images: img_path = os.path.join(species_dir, img_name)
- Sécurisation: stockage dans des chemins définis avec os.path.join() plutôt que concaténation directe

3. Paramétrer les outils afin d'importer les données de manière automatisée et sécurisée

Preuves de validation:

- Automatisation de l'importation dans path_to_input() et create_dataset_from_paths()
- Paramétrage des générateurs d'images: train_datagen = ImageDataGenerator(...)
- Sécurisation: normalisation des images avec rescale=1./255

- Paramètres complets dans create_image_generators() pour adapter automatiquement les images au modèle

4. Analyser, nettoyer, trier et s'assurer de la qualité des données

Preuves de validation:

- Prétraitement des images dans path_to_input()
- Augmentation de données dans train_datagen avec rotation, zoom, retournement
- Division en ensembles d'entraînement/validation/test avec stratification: train_test_split(..., stratify=labels)
- Visualisation de la distribution des données: visualize_data_distribution()
- Détection et traitement des données manquantes: vérification d'existence des espèces avec message d'avertissement

5. Construire la structure de stockage des données qui répond au besoin

Preuves de validation:

- Base de données SQLite dans DATABASE_PATH
- Création de la table observations dans init_db()
- Schéma adapté aux besoins: id, date, time, latitude, longitude, species, confidence, image_path
- Préparation de la connexion à la base: get_db_connection()

6. Représenter graphiquement les relations entre les données

Preuves de validation:

- Visualisation des distributions de données: visualize_data_distribution()
- Graphiques de performances dans evaluate_model() et plot_confusion_matrix()
- Interface frontend avec tableaux de bord dans observation.html
- Création de graphiques dans frontend/assets/js/app.js: createSpeciesChart() et createDateChart()

7. Exploiter et analyser les informations recueillies dans les structures de stockage

Preuves de validation:

- Requêtes SQL automatisées dans les routes API: conn.execute('SELECT * FROM observations...')

- Analyse des statistiques avec requêtes agrégées dans get_stats()
- Visualisation des observations dans displayObservations()
- Traitement automatisé des données dans identify_footprint()

BLOC E6.2: Développer un modèle prédictif d'une solution I.A

1. Générer des données d'entrée, récolter et adapter les types de données

Preuves de validation:

- Générateurs de données adaptés dans create_image_generators()
- Augmentation des données d'entraînement: rotation_range=40, zoom_range=0.2, horizontal_flip=True
- Adaptation du format des images: img = tf.keras.preprocessing.image.load_img(img_path, target_size=INPUT_SHAPE[:2])
- Normalisation des données: rescale=1./255

2. Paramétrer un environnement de codage adéquat pour développer le modèle d'apprentissage

Preuves de validation:

- Configuration des hyperparamètres dans backend/config.py: LEARNING_RATE_INITIAL, BATCH_SIZE, EPOCHS_INITIAL
- Configuration de l'environnement TensorFlow/Keras dans backend/train_model.py
- Configuration des GPU: gpus = tf.config.experimental.list_physical_devices('GPU')
- Définition de l'architecture dans backend/models/architecture.py

3. Coder le modèle d'apprentissage choisi

Preuves de validation:

- Architecture modulaire dans create_footprint_model()
- Flexibilité du choix du modèle: if base_model_type.lower() == "efficientnet": base_model = EfficientNetB0(...) else: base_model = MobileNetV2(...)
- Adaptation de l'architecture: ajout de couches GlobalAveragePooling2D, Dense, Dropout

- Configuration optimisée: model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

4. Réaliser et paramétrer une procédure d'entraînement adéquate

Preuves de validation:

- Procédure d'entraînement en deux phases dans train_model()
- Callbacks pour optimiser l'entraînement: EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
- Paramétrage du fine-tuning: dégel progressif des couches for layer in base_model.layers[-30:]
- Ajustement du taux d'apprentissage pour la phase de fine-tuning: LEARNING_RATE_FINE_TUNING = 5e-6

5. Réaliser une phase de test en choisissant une méthode appropriée

Preuves de validation:

- Évaluation complète dans evaluate_model()
- Calcul de métriques diverses: accuracy_score, precision_recall_fscore_support
- Matrice de confusion: confusion_matrix(y_true, y_pred)
- Top-K Accuracy: if true_label in pred_indices: top_k += 1

6. Ajuster l'apprentissage du modèle à partir des résultats obtenus

Preuves de validation:

- Ajustement du taux d'apprentissage avec ReduceLROnPlateau
- Fine-tuning après la phase initiale d'entraînement
- Analyse visuelle des erreurs: plot_error_examples()
- Analyse des paires d'espèces souvent confondues: confusion_pairs = []

BLOC E6.3: Produire et maintenir une solution I.A

1. Développer le back-end: API et programmes intégrés

Preuves de validation:

- API REST Flask complète dans backend/api/app.py
- Routes API bien définies: /api/status, /api/species, /api/identify, etc.
- Gestion des requêtes POST/GET: @app.route('/api/identify', methods=['POST'])

- Sécurisation CORS: CORS(app)

2. Développer le front-end: interface homme-machine ergonomique

Preuves de validation:

- UI responsive dans frontend/assets/css/styles.css
- Gestion de l'appareil photo dans frontend/assets/js/app.js
- Design adaptatif: @media (max-width: 768px)
- Structure HTML modulaire et sémantique dans les fichiers .html

3. Mettre en œuvre des plans de tests

Preuves de validation:

- Tests du modèle dans evaluate_model()
- Tests visuels dans visualize_model_predictions()
- Validation des données dans load_data()
- Gestion des erreurs dans les routes API: try/except avec retours d'erreurs formatés

4. Superviser le fonctionnement de la solution IA

Preuves de validation:

- Route de statut: /api/status retournant l'état du modèle
- Logging des actions: print(f"Modèle chargé: {MODEL_PATH}")
- Gestion des erreurs de chargement du modèle: except Exception as e: print(f"Erreur lors du chargement du modèle: {e}")
- Monitoring des prédictions avec niveau de confiance

5. Corriger les dysfonctionnements

Preuves de validation:

- Gestion d'erreurs robuste dans l'API: except Exception as e: return jsonify({'error': str(e)}), 500
- Validation des entrées: if 'image' not in request.json: return jsonify({'error': 'No image provided'}), 400
- Prétraitement d'images pour assurer la compatibilité
- Gestion des cas où le modèle n'est pas chargé: if model is None: return jsonify({'error': 'Model not loaded'}), 500

6. Réaliser les évolutions fonctionnelles

Preuves de validation:

- Architecture modulaire permettant l'extension
- Séparation des couches logiques (modèle, API, UI)
- Configuration centralisée dans config.py pour faciliter les modifications
- Possibilité d'ajouter de nouvelles espèces dans SPECIES_LIST

BLOC E6.4: Gérer les activités/tâches du développement d'une solution I.A

1. Mettre en œuvre une méthodologie adaptée de réalisation du projet

Preuves de validation:

- Structure de projet organisée: séparation backend/frontend/data
- Script d'automatisation run.sh avec étapes claires
- Vérification des prérequis dans check_prerequisites()
- Gestion des dépendances dans requirements.txt

2. Rendre compte de l'avancement du projet

Preuves de validation:

- Logs détaillés: print(f"✓ API démarrée avec succès (PID: $API_PID)")
- Statistiques d'utilisation accessibles via l'API: /api/stats
- Documentation dans le README.md expliquant la structure et l'utilisation
- Messages d'état lors de l'entraînement: print("\n6. Évaluation du modèle...")

3. Contribuer ou animer des réunions de travail

Preuves indirectes:

- Documentation complète du projet dans README.md permettant la collaboration
- Code commenté et organisé pour faciliter le travail d'équipe
- Interface utilisateur avec section "À propos" et "Contact" pour la communication

4. Auto-contrôler ses actions et productions

Preuves de validation:

- Tests d'évaluation du modèle: matrices de confusion, analyse des erreurs

- Validation des données: vérification des espèces manquantes
- Gestion des erreurs robuste dans l'API
- Création automatique des dossiers nécessaires: os.makedirs(UPLOAD_DIR, exist_ok=True)

5. Définir et mettre en place un système de veille

Preuves limitées mais présentes:

- Utilisation de technologies récentes: TensorFlow, Flask, responsive design
- Structure modulaire permettant l'intégration de nouveaux modèles (MobileNet, EfficientNet)
- Approche adaptative aux différentes architectures de réseaux de neurones
- Possibilité d'entraînement continu avec de nouvelles données

6. Améliorer le potentiel de développement des solutions IA

Preuves de validation:

- Collecte continue de données via l'interface utilisateur
- Stockage des observations pour enrichir la base de données
- Statistiques permettant d'identifier les axes d'amélioration
- Architecture permettant le fine-tuning et l'amélioration du modèle