**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