Module R508 – Compte-rendu Part.1

Compréhension du projet

Ce projet vise à développer un réseau de neurones convolutif capable de reconnaître des chiffres manuscrits à partir d'images. L'objectif est d'entraîner une intelligence artificielle sur un ensemble de données spécifiques, à savoir la base de données MNIST, qui contient des images de chiffres écrits à la main.

Le projet peut alors être décomposé en quelques grandes étapes, détaillées ci-dessous.

1. Analyse du Dataset

L'objectif ici est d'observer et de comprendre la structure de la base de données MNIST, contenant des images de 28x28 pixels et leurs labels correspondants. En la présence de données simplement interprétables, c'est une étape plutôt rapide (1 à 2 minutes).

2. Préparation des données

On commence alors par implémenter des fonctions pour charger et prétraiter les données (normalisation, division en ensembles d'entraînement et de test).

L'usage d'un langage de programmation adapté au traitement de données est donc essentiel. Notre choix se portera sur Python et ses bibliothèques. Les modules NumPy, TensorFlow ou PyTorch ne seront pas nécessaires ici, mais seront notamment incontournables pour réaliser le projet.

Le temps requis pour la complétion de cette étape est estimé à 4 à 6 heures.

3. Conception du modèle

Il s'agit ensuite d'entamer le développement de l'architecture du CNN. A moins d'être initié aux concepts, projets et pratiques concernant le développement de modèles d'intelligence artificielle, cette étape représente un premier grand saut vers l'inconnu.

Les tâtonnements et avancées faites à cette étape reposeront certainement sur le suivi de guides et tutoriels sur les architectures CNN dans un premier temps, avant de passer à la scrupuleuse consultation des documentations des outils recommandés par la communauté initiée au développement de modèles d'intelligence artificielle.

Ce temps de conception et d'auto-formation pourrait s'étendre sur une période de 3 à 5 jours.

4. Définition de la fonction de coût

Dans l'éventualité où le concept de fonction de coût n'aurait pas été suffisamment observé lors de l'étape précédente, cela constitue un deuxième saut vers l'inconnu. Il s'agit de proposer une fonction de coût appropriée pour évaluer les erreurs du modèle. Quelques retours sur cette étape seront sans doute nécessaires.

Les documentations portées sur les fonctions de coût en machine learning sont des ressources de choix pour compléter cette étape.

5. Choix de l'algorithme d'apprentissage

Vient alors le moment d'implémenter dans le projet un algorithme d'apprentissage, qui permettra de faire "apprendre" le modèle en optimisant ses paramètres.

Là encore, les documentations et recommandations de la communauté de l'intelligence artificielle et du machine learning sur les algorithmes d'apprentissage seront incontournables, et le temps dédié à cette étape pourrait atteindre 2 jours.

6. Entraînement, évaluation et optimisation du modèle

Cette dernière partie pratique et conceptuelle implique l'opérabilité et le fonctionnement effectif de l'algorithmique assemblée tout au long du projet.

Les principales activités de cette étape consistent à :

- Former le modèle sur l'ensemble d'entraînement, tout en surveillant le coût et la précision des progrès réalisés.
- Tester le modèle sur des ensembles de test.

Cet apprentissage se traduira certainement par une ou plusieurs séries d'ajustements successifs sur les paramètres du modèle, réalisées à la suite d'itérations sur les jeux de données fournis et évalués.

L'accès à des ressources contribuant aux opérations de calcul induites par l'entraînement du modèle (cartes graphiques, meilleurs processeurs, etc.) accélérera cette démarche d'apprentissage. Des outils d'analyse de performance seront requis pour quantifier dans une certaine mesure les capacités atteintes par le modèle entraîné.

Dans le cadre de ce projet universitaire, une semaine d'entraînement devrait convenir.

Estimation globale du temps

Le projet pourrait s'étendre sur environ 2 à 3 semaines, selon le niveau de connaissances déjà acquis et les ressources disponibles. En suivant les étapes décrites plus haut, le

projet devrait permettre de créer une IA performante capable de reconnaître des chiffres manuscrits avec un bon niveau de précision.

Etude de la base de données

La base de données fournie se décompose de la manière suivante :

- Un répertoire appelé *dataset* contient deux autres répertoires *test* et *train*, supposément respectivement dédiés au test et à l'entraînement des algorithmes que nous aurons l'occasion d'élaborer par la suite.
- 10000 images PNG (de 0.png à 9999.png) sont enregistrées dans le répertoire *test*.
- 60000 images PNG (de 0.png à 59999.png) sont enregistrées dans le répertoire *train*.

Avec cela, deux fichiers CSV (Comma-Separated Values) d'annotation sont fournis, *test_data.csv* et *train_data.csv*. Chacun de ces fichiers d'annotation associe chaque image de la base de données à un label, ce dernier permettant alors de catégoriser et de classer l'image.

Etude de fonctions OpenCV

Les fonctions cv2.imread(), cv2.imshow(), et cv2.imwrite() du module OpenCV en Python sont essentielles pour le traitement d'images :

- cv2.imread() permet de charger une image depuis un fichier. Cette fonction renvoie un tableau NumPy représentant l'image chargée. Si le fichier n'est pas trouvé ou si le chargement échoue, elle retourne None.
- cv2.imshow() permet d'afficher une image dans une fenêtre. Cette fonction ouvre une fenêtre graphique et affiche l'image. La fenêtre reste ouverte jusqu'à ce qu'une touche soit pressée (et il sera souvent nécessaire d'appeler cv2.waitKey() pour gérer cette attente).
- cv2.imwrite() permet d'enregistrer une image. Cette fonction renvoie True si l'image a été enregistrée avec succès, sinon False.

Module R508 – Compte-rendu Part.2

Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs (CNN, pour Convolutional Neural Networks) sont un type d'architecture de réseau de neurones particulièrement efficace pour le traitement et la reconnaissance d'images.

Un réseau de neurones convolutif est composé de plusieurs couches organisées en une séquence qui transforme progressivement les données d'entrée pour en extraire les caractéristiques les plus pertinentes. Les principales couches d'un CNN sont les couches de convolution, les couches de pooling et les couches entièrement connectées.

1. Notion de neurone

Un neurone représente une unité logique qui effectue une opération. Chaque couche d'un CNN contient un certain nombre de neurones. Chaque type de couche jouant un rôle spécifique dans l'extraction et la transformation des données d'entrée, le comportement des neurones varie alors en fonction du type de couche :

- Les neurones des couches convolutionnelles lles détectent des motifs locaux dans une région de l'entrée en appliquant un filtre (avec fonction d'activation).
- Les neurones des couches de pooling réduisent la taille des cartes de caractéristiques en résumant l'information (en sélectionnant les valeurs max ou moyennes).
- Les neurones des couches entièrement connectées intègrent toutes les caractéristiques pour produire une décision.
- Etc.

2. Couche de convolution

La couche de convolution est l'élément central du CNN. Elle utilise des filtres, appliqués sur l'image d'entrée pour détecter des motifs spécifiques.

Chaque filtre exploité par une couche de convolution est :

- Optimisé pour détecter un motif particulier (contours, textures, formes simples, etc.). A mesure que l'on progresse dans les couches, les motifs détectés deviennent alors de plus en plus complexes, allant de simples contours à des parties d'objets plus sophistiquées.

- Glissé (convolué) sur l'image d'entrée. À chaque position du filtre, une opération de produit scalaire entre les valeurs du filtre et les pixels de l'image est effectuée. Ce produit scalaire est stocké dans une nouvelle matrice appelée carte de caractéristiques (feature map).

2.1. Carte de caractéristiques

La carte de caractéristiques est une représentation intermédiaire de l'image dans un réseau de neurones convolutif. Elle est le résultat de l'application d'un filtre sur l'image d'entrée.

Concrètement, chaque filtre utilisé dans une couche de convolution génère une carte de caractéristiques distincte :

1. Lorsqu'un filtre de convolution est appliqué à une image d'entrée, il effectue une série d'opérations sur les pixels de l'image pour extraire une caractéristique spécifique, comme un contour horizontal ou vertical.
2. Cette application est répétée sur toute l'image, selon un certain nombre de pixels entre chaque application du filtre (stride). Le résultat est une carte de valeurs qui reflète la "présence" ou "intensité" de cette caractéristique dans différentes parties de l'image.

3. Couche entièrement connectée

La couche entièrement connectée sert à interpréter les caractéristiques extraites par les précédentes couches pour effectuer une classification ou une régression.

Chaque neurone de cette couche :

- Est connecté à tous les neurones de la couche précédente.
- Utilise des fonctions d'activation non linéaires pour produire une sortie interprétable. Les sorties interprétables peuvent alors être des probabilités d'appartenance à chaque classe dans une tâche de classification.

Chargeurs de données

Les chargeurs de données sont des composants qui permettent de préparer, charger et organiser les données d'entrée pour être traitées par le modèle de manière efficace.

La fonction **image_dataset_from_directory()** de TensorFlow (module **tensorflow.keras.utils**) convertit un répertoire structuré d'images en un objet de dataset

TensorFlow (**tf.data.Dataset**). Elle permet de charger et prétraiter des ensembles de données d'images stockées dans une structure de répertoires.

Paramètre	Type	Description
directory	str	Chemin du répertoire où les images sont stockées. Ce répertoire doit être organisé de manière hiérarchique: chaque sous-répertoire représentera une classe d'image analysée, contenant les images des objets correspondant.
labels	str, list, None	Détermine la manière dont les étiquettes sont gérées. Par défaut, les noms des sous-répertoires sont utilisés comme étiquettes (valeur **"inferred"**). Autrement, une liste explicite d'étiquettes correspondant aux images peut être utilisée.
label_mode	str	Spécifie le format des étiquettes. Par défaut, les classes sont codées comme entiers (valeur **"int"**).
class_names	list[str]	Liste explicite des noms de classes. Si non fourni, les classes sont inférées des sous-dossiers, triées alphabétiquement.
color_mode	str	Charge les images en couleur (par défaut) ou en niveaux de gris.
batch_size	int	Nombre d'images à inclure dans chaque batch (32 par défaut).
image_size	tuple[int, int]	Taille des images retournées. Les images sont redimensionnées si nécessaire. **(256, 256)** par défaut.
shuffle	bool	Si **True** (par défaut), les données retournées dans le dataset produit sont mélangées.
seed	int	Graine pour le mélange pseudo-aléatoire des données. Utile pour reproduire les résultats.
validation_split	float	Dans le cadre de l'apprentissage d'un modèle, détermine la fraction des données à réserver pour le dataset de validation.
subset	str	Spécifie si le dataset correspond au dataset d'entraînement ou au dataset de validation. Nécessite **validation_split** et **shuffle=True**.
interpolation	str	Méthode à utiliser pour redimensionner les images.
follow_links	bool	Si **True**, suit les liens symboliques dans le répertoire.

La fonction retourne un objet de type **tf.data.Dataset**, qui :

- Est un pipeline optimisé pour charger et prétraiter les données en mémoire ou sur disque.
- Contient des batches d'images sous forme de tenseurs et, si applicable, leurs étiquettes correspondantes.

Création de modèles séquentiels

Les fonctions **Sequential()** et **model.add()** font partie du module **tensorflow.keras** et sont couramment utilisées pour créer et configurer des modèles de réseaux de neurones dans TensorFlow.

Sequential() est une classe de TensorFlow qui sert à créer un modèle séquentiel, une structure linéaire dans laquelle les couches sont empilées les unes après les autres. Chaque couche a une entrée provenant de la couche précédente et une sortie allant vers la couche suivante.

Le modèle Sequential est idéal pour les architectures simples et linéaires (pas de bifurcation ou de fusion de flux de données). Les couches sont alors ajoutées dans l'ordre où elles sont déclarées, soit via le constructeur, soit via la méthode **add()**.

La méthode **model.add()** est utilisée pour ajouter une couche à un modèle séquentiel déjà créé avec **Sequential()** et permet une construction progressive du modèle, ce qui est utile pour des configurations dynamiques ou expérimentales. Elle prend une couche comme argument, qui est une instance d'une classe dérivée de Layer (comme Dense, Conv2D, etc.).

Pour correctement construire notre modèle séquentiel, il devient alors important que chaque couche ajoutée ait une entrée compatible avec la sortie de la couche précédente.

Module R508 – Compte-rendu Part.3

Evaluation d'un modèle de classification

Dans le cas de l'évaluation d'un modèle de classification sur des données de test, la procédure typique à suivre se décline selon les grands principes suivants:

1. Préparer les données de test. Il s'agit ici de vérifier que les données de test sont prétraitées de manière cohérente avec l'ensemble d'entraînement (normalisation, encodage, gestion des valeurs manquantes, etc.). Il faudra également confirmer que les données de test n'ont pas été utilisées pendant l'entraînement ou la validation. Dans le cas contraire, les performances démontrées par le modèle en situation d'évaluation face à de nouvelles données seraient faussées.
2. Générer des prédictions à l'aide du modèle. Tenter de prédire les étiquettes réelles sur les données de test tout en conservant les étiquettes réelles pour évaluer les prédictions réalisées.
3. Calculer des métriques de performance du modèle. On se basera notamment souvent sur la précision, correspondant à la proportion de prédictions correctes. D'autres métriques globales (F1-score, ROC-AUC, Log-Loss, etc.) et des métriques spécifiques à l'étude de chaque classe (capacités à éviter les faux positifs et à capturer les vrais positifs pour chaque classe) pourront être utilisées.
4. Approfondir les analyses menées:
    - Identifier les erreurs (faux positifs, faux négatifs) et voir quelles classes sont confondues à l'aide de matrices de confusion.
    - Visualiser les performances sur chaque classe par des courbes ROC et PR.
    - Analyser la répartition des probabilités prédites pour vérifier si le modèle est bien calibré.

Les étapes qui pourraient suivre dépassent le cadre de la simple évaluation statistique et impliquent d'introduire des critères qualitatifs.

Matrice de confusion pour une classification

Une matrice de confusion est un tableau qui permet de résumer les performances d'un modèle de classification en comparant les prédictions du modèle avec les valeurs réelles. Elle fournit une vue détaillée sur les erreurs commises par le modèle et permet d'évaluer sa capacité à classifier correctement les données.

Pour une classification impliquant plusieurs classes, la matrice est une table $n \times n$ où n est le nombre de classes. Chaque case (i,j) indique alors le nombre d'exemples appartenant à la classe réelle i prédits comme j.

Détection d'objets par fenêtre glissante

La détection d'objets avec l'approche de fenêtre glissante est une méthode classique en vision par ordinateur pour localiser et identifier des objets dans une image. Cela consiste à faire glisser une fenêtre de taille fixe sur l'image, à différentes positions et échelles, et en appliquant un classificateur à chaque sous-région pour déterminer si elle contient l'objet recherché.

Cette approche se décompose de la manière suivante:

1. Une fenêtre rectangulaire de dimensions fixes est d'abord définie pour extraire des sous-régions de l'image traitée. Cette fenêtre est ensuite déplacée (ou glissée) sur l'image avec un certain pas, horizontalement et verticalement, pour couvrir l'ensemble de l'image.
2. Comme les objets peuvent avoir différentes tailles, l'image traitée sera souvent redimensionnée à plusieurs niveaux. La fenêtre glissante est alors appliquée sur chaque version redimensionnée de l'image.
3. Pour chaque position de la fenêtre, un modèle de classification sera utilisé pour prédire si la région contient ou non l'objet d'intérêt. Si la prédiction est positive, la région est finalement marquée comme contenant un objet.