I. Préambule

Le code source de ce projet est disponible sur GitHub sur ce lien.
Ce projet porte sur la prédiction des kandjis (matrice carré) plus proches voisins qui résulte d'un d'entrainement de plusieurs milliers de données en partant d'une base de connaissance établie (des kandjis connus).
En chargeant le ficher de test du projet ou starting kit sur le challenge j'ai pu obtenir un score de 0.0579 ce qui correspond donc à une précision de 5,79 % dans la prédiction des véritable classes associé aux kandji ce qui largement faux soit 268 kandji ayant la bonne classification sur les 4630 kandji au total, donc ce taux de 5,79% laisse pensé que les prédictions des classes présentent dans le ficher ont été générées de façons aléatoire c'est-à-dire à travers une fonction qui génère des nombres aléatoires.

II. Algorithme des KNN

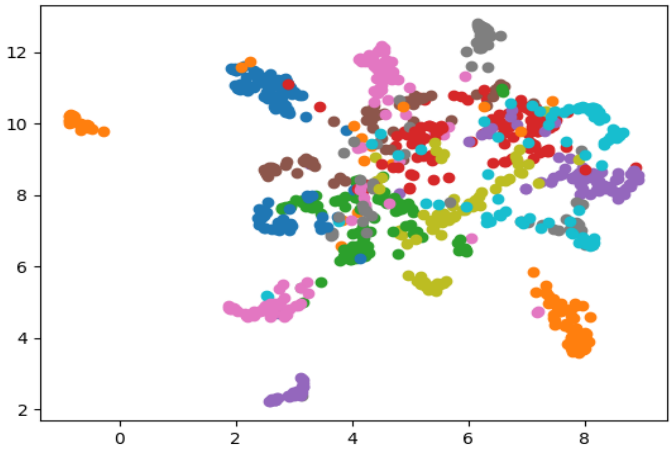
Concernant l'algorithme des kNN pour des besoins d'optimisation j'ai codé sur deux fichiers partant sur le même principe l'un utilisant la librairie Sklearn et l'autre n'utilisant aucune libraire du même type de sklearn (trop lent au vu des données) dans les deux cas j'ai obtenu les mêmes résultats. Parmi lesquels le meilleur k plus proches voisin des kandjis est 1 et la meilleure précision étant de 0.9600431965 soit un taux de 96 %.
Le code vise à classifier des caractères Kanji en utilisant l'algorithme des k plus proches voisins (k-NN), un algorithme de machine learning supervisé simple mais puissant. L'objectif est de prédire la catégorie d'un caractère Kanji inconnu en se basant sur les caractères les plus similaires (voisins) dans l'ensemble d'entraînement.

Grandes Étapes du Processus

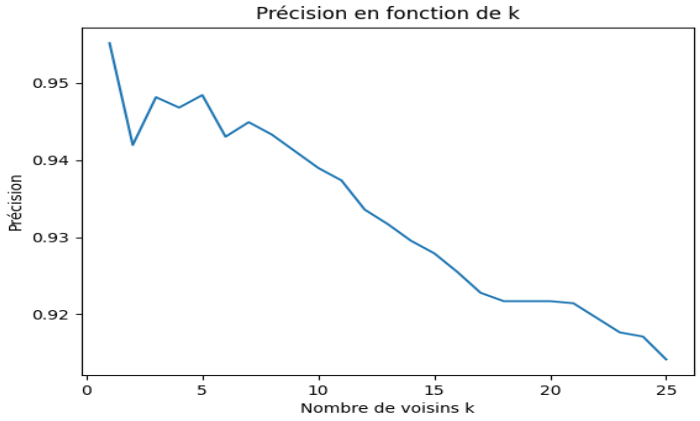
Chargement des Données : Les caractéristiques des caractères Kanji et leurs étiquettes sont extraites de fichiers CSV.
Prétraitement (Optionnel) : Un commentaire suggère la normalisation des données pour améliorer les performances, bien que cette étape ne soit pas active dans le script.
Division en Ensembles d'Entraînement et de Test : Les données sont séparées en deux parties, l'une pour l'entraînement du modèle et l'autre pour le tester.
Application de k-NN : L'algorithme est utilisé pour identifier les k caractères Kanji les plus proches et prédire la classe d'un nouveau caractère basé sur ces voisins.
Optimisation de k : Différentes valeurs de k sont testées pour trouver celle qui offre la meilleure précision.

Évaluation : La précision des prédictions est calculée et visualisée.
Prédiction sur Nouvelles Données : Le meilleur modèle est appliqué à un nouvel ensemble de données, et les prédictions sont sauvegardées.

Af



Affichage de la coube de precision :



1. Ficher n'utilisant pas sklearn
   nous avons les fonction suivantes :
   split_data : Sépare les données en fonction du paramètre test_size, qui contrôle la proportion de l'ensemble de test.

   distance : Calcule la distance euclidienne entre les points de données pour identifier les voisins les plus proches.

   neighbors : Retrouve les k voisins les plus proches d'un point donné.

   prediction : Prédit la classe d'un point en se basant sur la classe majoritaire parmi ses k voisins.

accuracy : Mesure la précision du modèle en comparant les prédictions aux vraies étiquettes.

Paramètres Importants :

test_size=0.8 : Indique que 80% des données sont utilisées pour le test, laissant 20% pour l'entraînement.

random_state=90 : Assure la reproductibilité des résultats en fixant la graine du générateur aléatoire.

k : Le nombre de voisins considérés par l'algorithme, avec une recherche du meilleur k par essais successifs.

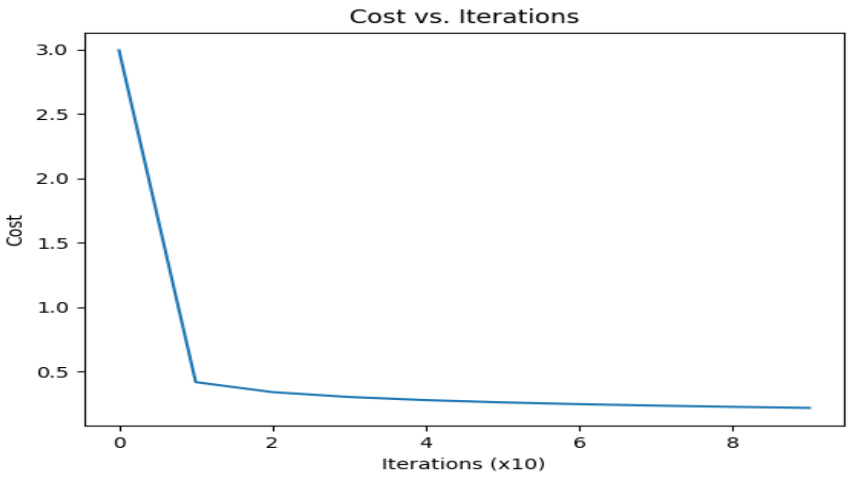
2. En utilisant sklearn :

   UMAP pour la Réduction de Dimensionnalité : UMAP est initialisé avec des paramètres spécifiques (n_neighbors=15, n_components=2, metric='euclidean') pour transformer les données dans un espace à 2D.
   Division en Ensembles d'Entraînement et de Test : Utilisation de train_test_split avec test_size=0.2 et random_state=42 pour assurer une division équilibrée et reproductible.
   k-NN pour la Classification : Le classificateur k-NN est appliqué avec différentes valeurs de k pour identifier le paramètre optimal qui maximise la précision.
   Visualisation de la Précision : La précision du classificateur est visualisée en fonction de différentes valeurs de k, permettant de sélectionner le meilleur k.
   Sauvegarde des Prédictions : Les prédictions finales sont enregistrées dans un fichier CSV pour une utilisation future.
   Paramètres Importants
   nb_kanji_plot=1000 : Spécifie le nombre de points Kanji à visualiser avec UMAP.
   test_size=0.2 : Définit la proportion de l'ensemble de données à utiliser comme ensemble de test.
   n_neighbors, n_components, et metric pour UMAP : Contrôlent la façon dont UMAP transforme les données.
   k dans k-NN : Le nombre de voisins les plus proches à considérer, avec une exploration de valeurs de 1 à 26 pour trouver le k optimal.

III. <u>Algorithme de régression logistique multivariée</u> :
J’ai obtenue un score de 0.9203563714902808 soit 92% .

Visualisation et Performance

La courbe de coût affichée en dessous montre comment la fonction de coût diminue au fil des itérations, indiquant l'amélioration du modèle



Fonctions et Paramètres Clés

softmax : Applique la fonction softmax pour convertir les logits en probabilités de classe.

compute_loss_and_gradients : Calcule la fonction de coût et les gradients par rapport aux poids et aux biais, en utilisant la perte logistique.

train : Entraîne le modèle en utilisant la descente de gradient, ajustant les poids et les biais pour minimiser la fonction de coût.

predict : Effectue des prédictions sur de nouvelles données en utilisant les poids et les biais appris.

calculate_accuracy : Calcule la précision du modèle en comparant les prédictions aux étiquettes réelles.

Paramètres Importants

learning_rate=0.1 : Détermine la taille des pas dans la descente de gradient.

num_iterations=2000 : Spécifie le nombre d'itérations pour l'entraînement du modèle (réduit à 100 pour l'exemple de code).

num_classes=20 : Indique le nombre de classes distinctes dans l'ensemble de données.

test_size=0.2 : La proportion de l'ensemble d'entraînement à utiliser comme ensemble de validation.

Préparation et Normalisation des Données

Calcul de la moyenne et de l'écart type : Se fait sur l'ensemble d'entraînement pour normaliser les ensembles d'entraînement et de test.

Éviter la division par zéro : L'écart type est ajusté pour les caractéristiques constantes en remplaçant 0 par 1.

IV. CNN

Grandes Étapes du Processus

Chargement et Prétraitement des Données : Les données sont chargées, normalisées, et restructurées pour correspondre à l'entrée attendue d'un CNN.

Préparation des Datasets et Dataloaders : Les données sont converties en tenseurs PyTorch et encapsulées dans des DataLoader pour l'entraînement et la validation.

Définition du Modèle CNN : Un modèle CNN est défini avec deux couches convolutives, des couches de pooling, et des couches fully connected pour la classification finale.

Entraînement du Modèle : Le modèle est entraîné sur l'ensemble d'entraînement à l'aide d'une fonction de perte de type CrossEntropy et d'un optimiseur Adam.

Évaluation de la Performance : La précision du modèle est calculée sur l'ensemble de validation.

Prédictions sur l'Ensemble de Test : Le modèle final est utilisé pour faire des prédictions sur l'ensemble de données de test.

Visualisation des Pertes : Les pertes d'entraînement sont tracées pour visualiser l'apprentissage du modèle au fil des époques.

Fonctions et Paramètres Clés

ConvNet : Classe définissant l'architecture du modèle CNN.

criterion : Fonction de perte CrossEntropy pour évaluer l'erreur du modèle.

optimizer : Optimiseur Adam avec un taux d'apprentissage de 0.001 pour ajuster les poids du modèle.

num_epochs : Nombre d'époques pour l'entraînement du modèle, défini à 100.

Préparation et Normalisation des Données

Les données d'entraînement et de test sont normalisées en utilisant la moyenne et l'écart type des données d'entraînement.

Les données sont restructurées en images 64x64 pour correspondre à l'entrée du CNN.

Appareil de Calcul

device : Utilise CUDA si disponible, sinon recourt au CPU pour les calculs, permettant une exécution plus rapide sur des appareils compatibles GPU.

Résultats

Le modèle atteint une précision spécifique sur l'ensemble de validation, illustrant sa capacité à généraliser à partir des données d'entraînement.

Les prédictions pour l'ensemble de test sont sauvegardées dans un fichier CSV pour une utilisation ultérieure.

resultat :

j’ai obtenu une précision de 0.9825053996 soit 98%