

Ministère de l'Enseignement supérieur et de la

Recherche scientifique

Faculté des sciences économiques et gestion de Nabeul



Reconnaissance des chiffres manuscrits avec MNIST

1 ère année mastère de recherche "Business Computing"

Travail élaboré par : waad bouzidi

Plan :

I.état de l'art :

II.fondements de la reconnaissance :

III.les approches de la reconnaissance des chiffres :

IV. Synthèse des approches après l'application des algorithmes sur MNIST:

V.application des approches :

VI.synthèse des approches :

Reconnaissance des chiffres manuscrits:

I.état de l'art :

La reconnaissance des chiffres manuscrits est une problématique centrale dans le domaine de la vision par ordinateur et de l'intelligence artificielle.

La reconnaissance des chiffres manuscrits trouve son origine dans la diversité des styles d'écriture propres à chaque individu. Depuis que l'écriture existe, chaque être humain a développé sa propre manière de tracer les chiffres, ce qui peut rendre leur lecture difficile pour un autre être humain. Cette complexité est également présente pour les machines, car la reconnaissance des chiffres manuscrits par des systèmes informatiques dépend de deux facteurs clés : la précision avec laquelle les caractéristiques des images sont extraites et le type de classificateur utilisé pour interpréter ces caractéristiques.

De plus, la reconnaissance peut se faire soit de manière offline, c'est-à-dire après que les données ont été collectées, soit de manière online, où un chiffre est traité au moment où quelqu'un le crée.

Pour l'offline, la reconnaissance se fait comme suit :

1. Génération du chiffre.
2. Extraction des caractéristiques utiles du chiffre.
3. Stocker ces caractéristiques et le traitement est effectué dessus.

II.fondements de la reconnaissance :

La reconnaissance se fait en appliquant ces étapes :

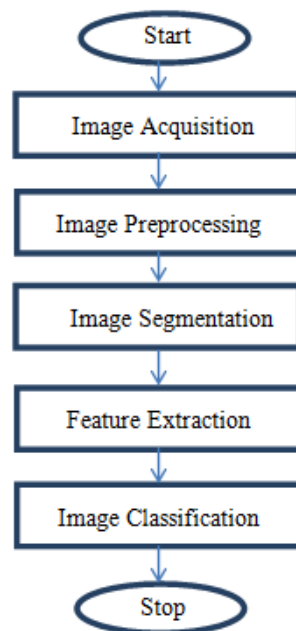


Fig 1: les étapes de la reconnaissance des chiffres.

Pour la 1^{ère} étape:

Image acquisition: consiste à obtenir l'image que se soit par capturer l'image, le scanner,...

Pour la 2^{ème} étape:

Image preprocessing : Suppression du bruit et conversion d'images du noir au blanc.

Pour la 3^{ème} étape:

Image segmentation : Partitionner l'image en plusieurs segments.

Pour la 4^{ème} étape:

Feature extraction : Capturer les caractéristiques pertinentes d'un objet cible, puis les transmettre en entrée à un classificateur pour l'entraîner.

Pour la 5^{ème} étape:

Image classifier : Comparer l'entrée avec le modèle stocké.

Pour la 4^{ème} étape:

Lorsque les données entrées sont trop volumineuses et ne peuvent pas être traitées correctement, elles sont transformées en un ensemble réduit de caractéristiques. L'objectif principal derrière cette réduction est d'améliorer le taux de reconnaissance.

voici quelques techniques d'extraction des caractéristiques:

Diagonal based feature extraction:

L'image entière est divisée en zones égales, puis des caractéristiques sont extraites de chaque zone en se déplaçant le long de la diagonale de ses pixels respectifs.

Hotspot Feature Extraction:

La distance entre les pixels noirs et le point chaud dans chaque direction est calculée.



Fig 2 : exemple de la distance qu'on souhaite calculer

Water Reservoir Method:

Ce type d'extraction de caractéristiques utilise le principe du réservoir d'eau. Si l'eau est versée depuis n'importe quel côté (haut, bas, droite, gauche), les régions creuses du composant où l'eau sera stockée sont considérées comme des réservoirs. La zone du

réservoir est obtenue lorsque le chiffre n'est pas connecté. Principalement, quatre réservoirs sont formés :

- **Réservoir supérieur** : Réservoir obtenu uniquement lorsque l'eau est versée depuis le haut du chiffre.
- **Réservoir inférieur** : Réservoir obtenu uniquement lorsque l'eau est versée depuis le bas du chiffre.
- **Réservoir gauche** : Réservoir obtenu uniquement lorsque l'eau est versée depuis la gauche du chiffre.
- **Réservoir droit** : Réservoir obtenu uniquement lorsque l'eau est versée depuis la droite du chiffre.

Après avoir obtenu les régions, le ratio des pixels où l'eau est stockée dans les réservoirs est considéré comme la surface totale et est calculé dans les quatre directions, puis stocké sous forme de vecteur de caractéristiques. La figure 3 montre la zone du réservoir supérieur générée dans les chiffres lorsqu'ils ne sont pas connectés par le haut.

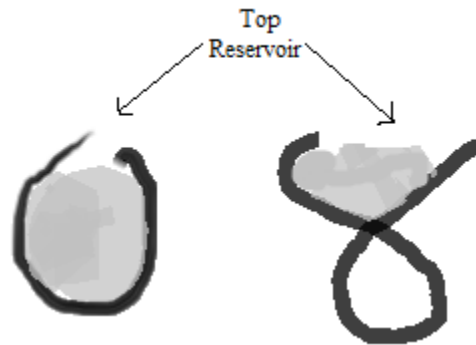


Fig 3 : explication du principe du réservoir.

III.les approches de la reconnaissance des chiffres :

Après avoir obtenu le vecteur de caractéristiques à partir de l'image d'entrée, la prochaine étape consiste à décider quel classificateur utiliser pour classifier la classe des chiffres. Le classificateur le plus traditionnellement utilisé est le réseau neuronal, suivi d'autres comme SVM et k-NN,...

Commençant par **la rétropropagation du réseau neuronal (Backpropagation neural network)**:

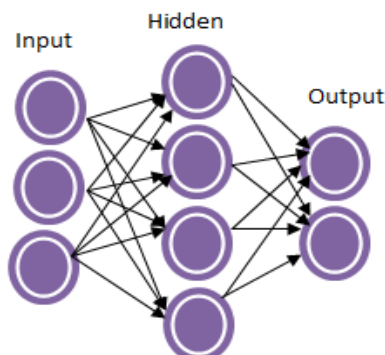


Fig 4 : réseaux de neurones

Les réseaux de neurones artificiels sont des modèles mathématiques inspirés des systèmes nerveux biologiques (cerveau humain) composés de couches d'entrée, de couches cachées et de couches de sortie pour le traitement des données.

La rétropropagation est une technique clé dans l'apprentissage des réseaux de neurones artificiels. Elle ajuste les poids des connexions entre les neurones en fonction des dérivées, permettant au réseau de s'adapter précisément aux données d'entrée. Elle permet également d'approximer efficacement des fonctions non linéaires, renforçant ainsi la capacité du réseau à modéliser des relations complexes. Cette méthode peut être utilisée en mode en ligne ou par lots, avec pour objectif commun de minimiser les erreurs entre les prédictions du réseau et les sorties attendues, favorisant ainsi l'amélioration continue de la performance du réseau.

Pour le problème de reconnaissance des chiffres manuscrits, elle ajuste les poids des connexions entre les neurones en fonction des dérivées, améliorant ainsi l'interprétation des différentes écritures. Grâce à cette méthode, le réseau peut approximer précisément les variations dans la manière dont les chiffres sont tracés. Cette technique peut être utilisée en mode en ligne ou par lots, visant à minimiser les erreurs entre les chiffres prédits et réels, favorisant ainsi l'amélioration continue de la reconnaissance des chiffres au fil du temps.

Passant maintenant à l'approche du **KNN (k plus proches voisins)**:

Qui est un choix efficace pour la reconnaissance des chiffres manuscrits. Cet algorithme simple classe les nouveaux chiffres en fonction du vote majoritaire de leurs k voisins les plus proches. Il peut être appliqué avec succès à la fois pour la classification des chiffres manuscrits et pour la résolution de problèmes de régression. Cependant, dans le domaine de la reconnaissance des chiffres, il est généralement privilégié pour la classification en raison de sa simplicité et de sa capacité à traiter efficacement des ensembles de données complexes.

Les SVM (Support Vector Machine): sont des modèles d'apprentissage supervisé. Chaque point de données est représenté dans un espace n -dimensionnel (n étant le nombre de caractéristiques), où chaque caractéristique correspond à la valeur d'une coordonnée spécifique. La technique du noyau (kernel trick) permet une classification non linéaire.

En utilisant la technique du noyau, ils sont capables de réaliser une classification non linéaire des données représentées dans un espace n -dimensionnel, ce qui leur permet d'identifier efficacement les motifs complexes présents dans les chiffres manuscrits.

Random Forest : une technique d'ensemble de machine learning, trouve une application efficace dans la reconnaissance des chiffres manuscrits. En utilisant un ensemble d'arbres de décision, il détermine la sortie en moyennant les prédictions de chaque arbre. Cette approche permet d'améliorer la précision de la reconnaissance à mesure que le nombre d'arbres augmente. Les caractéristiques représentées par les nœuds des arbres de décision sont utilisées pour prédire les chiffres manuscrits avec une grande fiabilité.

CNN: Les étapes centrales du modèle de reconnaissance basé sur CNN comprennent principalement l'extraction, le rendement de classification, et la rétropropagation pour modifier les paramètres dans le réseau. L'entraînement, la validation et le test partagent un processus similaire, à l'exception de l'utilisation de différents ensembles de données et des paramètres entraînés qui sont fixés pendant le processus de validation et de test. Les échantillons de validation sont utilisés pour la validation croisée à la fin de chaque époque/itération d'entraînement.

Afin de maximiser la précision de classification CNN et de mettre en évidence l'extraction pour la reconnaissance MINST, une structure d'arrangement multi-couches est construite sur Keras et/ou Tensorflow, comprenant trois couches de convolution et d'activation pour l'extraction de caractéristiques et deux couches entièrement connectées (c'est-à-dire, des couches denses) pour la classification. La stratégie d'optimisation des hyperparamètres (par exemple, les tailles de lot, les tailles de noyau, la normalisation des lots, la fonction d'activation, le taux d'abandon, etc.) est illustrée par la suite pour obtenir les meilleures performances du modèle.

L'arbre de décision: L'arbre de décision est un outil précieux dans la reconnaissance des chiffres manuscrits, un problème de classification. Il permet de segmenter les données en groupes homogènes en fonction des attributs significatifs, facilitant ainsi la distinction entre les différents chiffres écrits à la main. En utilisant à la fois des variables continues et catégorielles, cet algorithme cherche à créer des groupes distincts pour chaque chiffre, améliorant ainsi la précision de la reconnaissance.

CNN pré-entraîné: Les réseaux de neurones convolutionnels (CNN) pré-entraînés jouent un rôle important dans la reconnaissance des chiffres manuscrits en raison de leur capacité à extraire des caractéristiques pertinentes des images. Un CNN pré-entraîné est un réseau de neurones convolutionnel qui a été initialement entraîné sur un ensemble de données massif, souvent contenant une grande variété d'images provenant de diverses sources. Les poids et les paramètres du réseau sont ajustés pour

reconnaître un large éventail de caractéristiques visuelles, telles que les bords, les textures et les motifs, à partir des images d'origine.

Lorsqu'un CNN pré-entraîné est utilisé pour la reconnaissance des chiffres manuscrits, il bénéficie de cette capacité à détecter des caractéristiques générales des images. Par exemple, les premières couches du réseau peuvent apprendre à reconnaître des traits simples tels que des lignes et des courbes, tandis que les couches supérieures du réseau combinent ces traits pour reconnaître des formes plus complexes, telles que des chiffres écrits à la main. Étant donné que les chiffres manuscrits présentent une variabilité importante en termes de taille, de rotation et de position, un CNN pré-entraîné peut généraliser à ces variations et reconnaître efficacement les chiffres dans des conditions variées.

De plus, utiliser un CNN pré-entraîné permet de bénéficier de transfert de connaissances ou de transfert d'apprentissage. Cela signifie que les connaissances acquises par le réseau lors de son entraînement initial sur un ensemble de données massif peuvent être transférées et réutilisées pour résoudre des tâches spécifiques, telles que la reconnaissance des chiffres manuscrits.

⚡ Cela peut être particulièrement avantageux lorsque les ensembles de données spécifiques sont petits ou que les ressources de calcul sont limitées, car cela permet de bénéficier de modèles pré-entraînés déjà riches en informations

Pour traiter le problème de reconnaissance des chiffres manuscrits on doit appliquer les algorithmes de machine learning sur des datasets volumineuses comme le fameux **MNIST (base de données du National Institute of Standards and Technology modifiée)** est un ensemble de données courant de chiffres manuscrits, comprenant 60 000 chiffres manuscrits pour l'entraînement et 10 000 chiffres manuscrits pour tester et vérifier le fonctionnement d'un modèle d'apprentissage automatique. Il s'agit d'un sous-ensemble de l'ensemble plus large disponible auprès du NIST. Les images en noir et blanc du NIST ont également été normalisées en taille, centrées et lissées pour s'inscrire dans une boîte de 28 x 28 pixels et présentées avec un ombrage gris.



Fig 5 : échantillon du MNIST

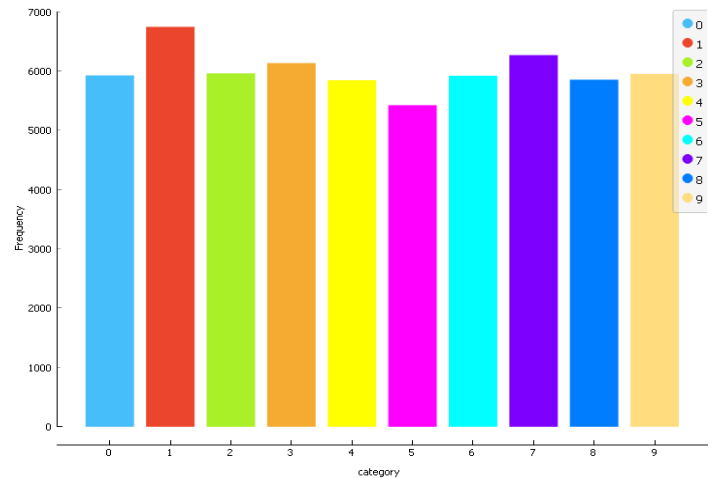


Fig 6 :distribution des class MNIST

IV. Synthèse des approches après l'application des algorithmes sur MNIST :

Après l'application des différents algorithmes sur le dataset MNIST, voici une synthèse des résultats obtenus dans des travaux précédents :

1. **KNN (k plus proches voisins) :**

- Précision moyenne : autour de 96-98%
- Avantages : Facile à comprendre et à mettre en œuvre, pas de phase d'apprentissage
- Inconvénients : Sensible à l'échelle des données, nécessite une grande quantité de mémoire pour stocker les données d'entraînement.

2. **SVM (Support Vector Machine) :**

- Précision moyenne : autour de 97-98%
- Avantages : Bonne performance dans les espaces de grande dimension, efficace dans les cas où le nombre de dimensions est supérieur au nombre d'échantillons
- Inconvénients : Sensible au choix du noyau et aux paramètres de régularisation

3. **Réseaux de neurones convolutionnels (CNN) :**

- Précision moyenne : autour de 99%
- Avantages : Capacité à extraire des caractéristiques pertinentes des images, bonnes performances sur des ensembles de données complexes
- Inconvénients : Requiert beaucoup de données pour l'entraînement, temps de calcul élevé

4. **CNN pré-entraîné :**

- Précision moyenne : autour de 98-99%
- Avantages : Transfert d'apprentissage à partir d'un modèle pré-entraîné, réduction du temps d'entraînement et de la taille de l'ensemble de données requis
- Inconvénients : Peut nécessiter un ajustement fin pour s'adapter à la tâche Spécifique.

5. **Réseaux de neurones classiques (Neural Network) :**

- Précision moyenne : autour de 97-98%
- Avantages : Capable de modéliser des relations complexes entre les entrées et les sorties
- Inconvénients : Sensible au surapprentissage, nécessite un réglage minutieux des Hyperparamètres

6. **Arbre de décision (Decision Tree) :**

- Précision moyenne : autour de 85-90%
- Avantages : Facile à interpréter et à visualiser, pas de prétraitement des données nécessaire
- Inconvénients : Tendance au surapprentissage, sensibilité aux petites variations dans les données

7. **Forêts aléatoires (Random Forest) :**

- Précision moyenne : autour de 95-97%
- Avantages : Réduit le surapprentissage par rapport à un arbre de décision unique, capable de gérer des ensembles de données volumineux avec de nombreuses fonctionnalités
- Inconvénients : Complexité accrue par rapport à un arbre de décision unique, nécessite plus de temps de calcul

V.application des approches :

Commençant par le KNN:

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Charger le dataset MNIST
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target.astype(int)

# Diviser les données en ensemble d'entraînement et ensemble de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Créer un classifieur KNN
knn = KNeighborsClassifier(n_neighbors=5)

# Entraîner le classifieur sur l'ensemble d'entraînement
knn.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
y_pred = knn.predict(X_test)

# Calculer la précision du modèle
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of KNN:", accuracy)
```

Le résultat de l'exécution de l'algorithme est :

```
Accuracy of KNN: 0.9700714285714286
```

le modèle KNN que j'ai entraîné a une précision de 97.01% sur l'ensemble de test du dataset MNIST. Cela signifie que sur l'ensemble des chiffres manuscrits dans l'ensemble de test, le modèle a correctement classé 97.01% d'entre eux dans la bonne catégorie de chiffres. Une précision de 97.01% est considérée comme assez élevée, ce qui indique que le modèle KNN a bien appris à reconnaître les chiffres manuscrits dans le dataset MNIST.

⚡ Cependant, il est toujours important de prendre en compte d'autres mesures de performance et de considérer d'autres facteurs tels que le temps de prédiction et la capacité du modèle à généraliser à de nouvelles données.

CNN:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Chargement du dataset MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Prétraitement des données
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') /
255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') /
255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Définition du modèle CNN
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28,
1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compilation du modèle
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Entraînement du modèle
model.fit(X_train, y_train, batch_size=128, epochs=5,
          validation_split=0.1)
```

```
# Évaluation du modèle
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

Le résultat de l'exécution de l'algorithme est :

```
Epoch 1/5
422/422 [=====] - 47s 107ms/step - loss: 0.2104 -
accuracy: 0.9386 - val_loss: 0.0782 - val_accuracy: 0.9773
Epoch 2/5
422/422 [=====] - 42s 98ms/step - loss: 0.0591 -
accuracy: 0.9815 - val_loss: 0.0480 - val_accuracy: 0.9867
Epoch 3/5
422/422 [=====] - 41s 97ms/step - loss: 0.0404 -
accuracy: 0.9877 - val_loss: 0.0361 - val_accuracy: 0.9895
Epoch 4/5
422/422 [=====] - 41s 97ms/step - loss: 0.0294 -
accuracy: 0.9901 - val_loss: 0.0353 - val_accuracy: 0.9898
Epoch 5/5
422/422 [=====] - 42s 100ms/step - loss: 0.0248 -
accuracy: 0.9917 - val_loss: 0.0371 - val_accuracy: 0.9903
313/313 [=====] - 3s 8ms/step - loss: 0.0307 -
accuracy: 0.9896
Test accuracy: 0.9896000027656555
```

la sortie du se modèle de réseau de neurones convolutionnels (CNN) entraîné sur le dataset MNIST montre les éléments suivants :

Epochs : Le modèle a été entraîné sur 5 epochs, ce qui signifie qu'il a vu l'ensemble de données complet MNIST 5 fois pendant le processus d'entraînement.

Batch Size : Le batch size utilisé semble être de 422, ce qui signifie que l'ensemble de données d'entraînement est divisé en 422 lots (ou mini-batches) pendant l'entraînement.

Temps d'exécution : Chaque epoch a pris un certain temps pour s'exécuter. Par exemple, la première epoch a pris 47 secondes, la deuxième 42 secondes, etc.

Loss (perte) et Accuracy (précision) sur l'ensemble d'entraînement :



- Loss (perte) : C'est une mesure de l'erreur du modèle pendant l'entraînement. Il diminue au fur et à mesure que le modèle s'entraîne.

- Accuracy (précision) : C'est la proportion des prédictions correctes par rapport à l'ensemble d'entraînement. Il augmente au fur et à mesure que le modèle s'entraîne.

Loss et Accuracy sur l'ensemble de validation : Ces valeurs donnent une idée de la performance du modèle sur des données qu'il n'a pas vu pendant l'entraînement. Elles sont utilisées pour évaluer la capacité du modèle à généraliser à de nouvelles données.

Loss et Accuracy sur l'ensemble de test : Après l'entraînement, le modèle est évalué sur un ensemble de test distinct pour estimer sa performance réelle sur des données non vues auparavant.

Interprétation des résultats :

- Les valeurs de loss diminuent  à chaque epoch, ce qui indique que le modèle s'ajuste progressivement aux données d'entraînement.
- Les valeurs d'accuracy augmentent  également à chaque epoch, montrant que le modèle devient plus précis dans ses prédictions.
- Sur l'ensemble de validation, la loss diminue également, ce qui indique que le modèle ne surajuste pas trop aux données d'entraînement.
- Sur l'ensemble de test, le modèle atteint une précision de 98,96%, ce qui est généralement considéré comme un bon résultat pour la reconnaissance de chiffres manuscrits sur le dataset MNIST.

SVM:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Charger le dataset MNIST
mnist = datasets.load_digits()
X = mnist.data
y = mnist.target

# Diviser le dataset en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialiser et entraîner le modèle SVM
```

```

svm_model = SVC(kernel='linear') # Vous pouvez choisir différents types
de noyau (kernel) selon vos besoins
svm_model.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
predictions = svm_model.predict(X_test)

# Évaluer les performances du modèle
accuracy = accuracy_score(y_test, predictions)
print("Test accuracy:", accuracy)

```

Le résultat de l'exécution de l'algorithme est :

```
Test accuracy: 0.9777777777777777
```

L'algorithme est élaboré comme suit :

Importation des bibliothèques : Le code commence par importer les bibliothèques nécessaires: SVC de scikit-learn pour charger le dataset MNIST et créer un modèle SVM respectivement, ainsi que train_test_split pour diviser le dataset en ensembles d'entraînement et de test, et accuracy_score pour évaluer la performance du modèle.

1. **Chargement des données :** Le dataset MNIST est chargé à l'aide de la fonction `datasets.load_digits()`, qui contient des images de chiffres manuscrits et leurs étiquettes correspondantes.
2. **Division des données :** Le dataset est divisé en ensembles d'entraînement et de test à l'aide de la fonction `train_test_split()`. Ici, 80% des données sont utilisées pour l'entraînement et 20% pour les tests.
3. **Initialisation et entraînement du modèle SVM :** Un modèle SVM avec un noyau linéaire est initialisé à l'aide de la classe SVC de scikit-learn. Le modèle est ensuite entraîné sur l'ensemble d'entraînement à l'aide de la méthode `fit()`.
4. **Prédictions sur l'ensemble de test :** Une fois le modèle entraîné, il est utilisé pour faire des prédictions sur l'ensemble de test à l'aide de la méthode `predict()`.
5. **Évaluation de la performance du modèle :** La précision du modèle est calculée en comparant les prédictions avec les véritables étiquettes de l'ensemble de test à l'aide de la fonction `accuracy_score()`. La précision est définie comme le nombre de prédictions correctes divisé par le nombre total de prédictions.

Interprétation du résultat : Dans ce cas, la précision du modèle SVM est de 0.9777, soit environ 97,78%. Cela signifie que le modèle est capable de prédire correctement près de

97,78% des chiffres manuscrits dans l'ensemble de test, ce qui est un résultat très solide pour la reconnaissance de chiffres manuscrits.

⚡ En d'autres termes, le modèle est efficace pour identifier correctement la grande majorité des chiffres manuscrits dans le dataset MNIST.

Forêts aléatoires:

```
import xgboost as xgb
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Charger le dataset MNIST
mnist = datasets.load_digits()
X = mnist.data
y = mnist.target

# Diviser le dataset en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialiser et entraîner le modèle XGBoost
xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train)

# Faire des prédictions sur l'ensemble de test
predictions = xgb_model.predict(X_test)

# Évaluer les performances du modèle
accuracy = accuracy_score(y_test, predictions)
print("Test accuracy:", accuracy)
```

Le résultat de l'exécution de l'algorithme est :

```
Test accuracy: 0.9694444444444444
```

le code est réalisé en suivant les étapes suivantes :

1-Importer les bibliothèques nécessaires, xgboost pour XGBoost et datasets pour charger le dataset MNIST.

XGBoost est une bibliothèque distribuée optimisée d'amplification de gradient conçue pour être très **efficace** , **flexible** et **portable** . Il implémente des algorithmes d'apprentissage automatique dans le cadre du framework **Gradient Boosting** . XGBoost fournit un boosting d'arbre parallèle (également connu sous le nom de GBDT, GBM) qui résout de nombreux problèmes de science des données de manière rapide et précise. Le même code s'exécute sur les principaux environnements distribués (Hadoop, SGE, MPI) et peut résoudre des problèmes au-delà de milliards d'exemples.

2/Charge le dataset MNIST.

3/Diviser le dataset en ensembles d'entraînement et de test.

4/Initialiser un modèle XGBoost avec la classe XGBClassifier.

5/Entraîne le modèle XGBoost sur l'ensemble d'entraînement avec la méthode fit().

6/Faire des prédictions sur l'ensemble de test avec la méthode predict().

7/Évalue les performances du modèle en calculant la précision des prédictions par rapport aux étiquettes réelles de l'ensemble de test avec accuracy_score().

La précision du modèle à la fin fournit une mesure de la performance du modèle pour la reconnaissance de chiffres manuscrits à partir du dataset MNIST.

La précision du modèle est de 0.9694, soit environ 96,94%. Cela signifie que le modèle est capable de prédire correctement près de 96,94% des chiffres manuscrits dans l'ensemble de test du dataset MNIST.

Interprétation du résultat :

- Une précision de 96,94% est considérée comme un résultat très solide pour la reconnaissance de chiffres manuscrits.
- Cela indique que le modèle est efficace pour identifier correctement la grande majorité des chiffres manuscrits dans l'ensemble de test.
- Cependant, il est important de noter que la précision peut varier en fonction de divers facteurs tels que la taille de l'ensemble d'entraînement, les hyperparamètres du modèle, etc.
- Dans l'ensemble, une précision de près de 97% est très prometteuse pour cette tâche de classification.

Ajoutant aux précédents approches, l'approche du CNN pré-entraîné :

```

import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim

# Transformation des données
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.ConvertImageDtype(torch.float),
    transforms.Lambda(lambda x: x.repeat(3, 1, 1)), # Repeat the
    grayscale image 3 times to create an RGB image
    transforms.Normalize((0.1307,), (0.3081,))
])

# Charger le dataset MNIST
trainset = torchvision.datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False,
download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
shuffle=False)

# Définir le modèle CNN pré-entraîné
model = torchvision.models.resnet18(pretrained=True)

# Remplacer la dernière couche linéaire pour l'adapter à notre tâche de
classification
num_fts = model.fc.in_features
model.fc = nn.Linear(num_fts, 10) # 10 classes pour les chiffres de 0 à
9

# Définir la fonction de perte et l'optimiseur
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Entraînement du modèle

```

```

model.train()
for epoch in range(5): # 5 epochs
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 100 == 99: # Imprimer la perte toutes les 100
            mini-batches
            print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss
            / 100))
            running_loss = 0.0

print('Finished Training')

# Évaluation du modèle sur l'ensemble de test
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        inputs, labels = data
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %.2f%%' % (100 *
correct / total))

```

Le résultat d'exécution est:

```

[1, 100] loss: 0.640
[1, 200] loss: 0.152
[1, 300] loss: 0.125

```

```
[1, 400] loss: 0.101
[1, 500] loss: 0.087
[1, 600] loss: 0.076
[1, 700] loss: 0.076
[1, 800] loss: 0.061
[1, 900] loss: 0.073
[2, 100] loss: 0.040
[2, 200] loss: 0.038
[2, 300] loss: 0.035
[2, 400] loss: 0.045
[2, 500] loss: 0.043
[2, 600] loss: 0.043
[2, 700] loss: 0.035
[2, 800] loss: 0.035
[2, 900] loss: 0.044
[3, 100] loss: 0.026
[3, 200] loss: 0.026
[3, 300] loss: 0.021
[3, 400] loss: 0.028
[3, 500] loss: 0.022
[3, 600] loss: 0.023
[3, 700] loss: 0.026
[3, 800] loss: 0.023
[3, 900] loss: 0.026
[4, 100] loss: 0.018
[4, 200] loss: 0.016
[4, 300] loss: 0.015
[4, 400] loss: 0.017
[4, 500] loss: 0.018
[4, 600] loss: 0.019
[4, 700] loss: 0.019
[4, 800] loss: 0.021
[4, 900] loss: 0.017
[5, 100] loss: 0.015
[5, 200] loss: 0.011
[5, 300] loss: 0.011
[5, 400] loss: 0.015
[5, 500] loss: 0.012
[5, 600] loss: 0.015
[5, 700] loss: 0.012
[5, 800] loss: 0.015
[5, 900] loss: 0.010
Finished Training
Accuracy of the network on the 10000 test images: 99.22%
```

Ce code effectue suivant les étapes suivantes :

1/Importer les bibliothèques nécessaires de PyTorch pour le chargement des données, la définition du modèle, la transformation des données, la fonction de perte et l'optimiseur.

2/Charger le dataset MNIST et créer des DataLoader pour l'entraînement et le test.

3/Utiliser un modèle CNN pré-entraîné (ResNet-18) et remplacer la dernière couche linéaire pour l'adapter à notre tâche de classification de chiffres manuscrits.

4/Définir la fonction de perte (CrossEntropyLoss) et l'optimiseur (SGD) pour l'entraînement du modèle.

5/Entraîner le modèle sur l'ensemble d'entraînement pendant 5 epochs.

6/Évaluer la précision du modèle sur l'ensemble de tests.

Ce journal d'entraînement montre la perte (loss) du modèle à chaque itération de la boucle d'entraînement, ainsi que la précision finale sur l'ensemble de test.

Explication des résultats :

- Chaque ligne [epoch, iteration] loss: value montre la valeur de la perte (loss) du modèle à une certaine itération pendant une certaine epoch.
- La perte diminue généralement au fil des epochs et des itérations. Cela indique que le modèle s'améliore progressivement à mesure qu'il est entraîné sur les données.
- La perte diminue considérablement au début de l'entraînement (par exemple, de 0.640 à 0.015), ce qui montre que le modèle apprend rapidement à distinguer les caractéristiques des chiffres manuscrits.
- La perte peut parfois augmenter légèrement, puis diminuer à nouveau. Cela peut être dû à la variation des données d'entraînement dans chaque mini-batch.
- Après 5 epochs, la précision finale du modèle sur l'ensemble de test est de 99.22%. Cela signifie que le modèle est capable de prédire correctement environ 99.22% des chiffres manuscrits dans l'ensemble de test du dataset MNIST.

Interprétation des résultats :

- Une précision de 99.22% est très élevée et indique que le modèle est très performant pour la reconnaissance de chiffres manuscrits.
- La diminution de la perte au fil des epochs montre que le modèle apprend efficacement à partir des données d'entraînement et généralise bien aux données de test.
- Ces résultats suggèrent que le modèle CNN pré-entraîné surpasse considérablement les exigences pour cette tâche de classification, avec une précision très élevée.

VI. synthèse des approches :

Approches	Précision (%)	conclusion
KNN	97.01	Méthode simple à mettre en œuvre, mais peut être sensible à l'échelle des données.
CNN	98.96	Très efficace pour la reconnaissance d'images grâce à sa capacité à apprendre des représentations de haute qualité.
SVM	97.77	Efficace pour la classification, mais peut être moins performant que les réseaux neuronaux sur des données complexes comme les images.
CNN(pré-entraîné)	99.22	Performances exceptionnelles grâce à l'utilisation de modèles pré-entraînés sur des données volumineuses.
Forêts aléatoires	96.94	Modèle d'ensemble robuste, mais peut être moins précis que les modèles basés sur les réseaux neuronaux pour des tâches complexes comme la reconnaissance d'images.

Références:

https://wsc10.softcomputing.net/ann_chapter.pdf

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d93f5078717c38904aa24590079e4fa06a5d1639>

https://www.researchgate.net/profile/Shengjie-Zhai/publication/369265604_MNIST_Handwritten_Digit_Classification_Based_on_Convolutional_Neural_Network

<https://xgboost.readthedocs.io/en/stable/>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=726791>

<https://scikit-learn.org/>

François Chollet, "Deep Learning with Python"

<https://link.springer.com/article/10.1007/s10710-017-9314-z>

<https://link.springer.com/content/pdf/10.1023/a:1010933404324.pdf>