26/05/2024

Classification d'Images ASL

IA pour les systèmes cyber-physique

Christoph Samuel – Mouche Valentin

Table des Matières

I. Introduction

Ce projet vise à appliquer les connaissances acquises dans les travaux pratiques (TP) précédents sur la **classification d'images** pour développer un **système de classification d'images spécifique**. Nous avons choisi de travailler sur la **reconnaissance de la langue des signes américaine (ASL).** Le jeu de données utilisé pour l'entraînement et la validation est **le "Sign Language MNIST"** disponible sur **Kaggle.**

1. Objectif

L'objectif principal est de construire un système de classification d'images capable de **reconnaître des lettres de l'alphabet** en langue des signes américaine **à partir d'images de gestes de la main.** Nous allons mettre en place un modèle de réseau de **neurones convolutionnels (CNN)** pour cette tâche, en utilisant **TensorFlow et Keras.**

2. Jeu de Données

Le jeu de données **"Sign Language MNIST"** contient 27,455 images en niveaux de gris de 28x28 pixels. Ces images **représentent 24 lettres de l'alphabet** (A-Z sans J et Z étant des mouvements et non une image fixe). Le jeu de données est **divisé en deux répertoires** : "Train" **pour l'entraînement** et "Test" **pour le test.** Chaque répertoire contient des sous-dossiers nommés selon les classes (lettres).

3. Prétraitement des Données

Pour **améliorer les performances de notre modèle,** nous avons appliqué des techniques d'augmentation des données et de normalisation**.** Les transformations incluent le **redimensionnement des images,** la **normalisation des pixels** et des transformations aléatoires telles que des **translations horizontales...**

II. Analyse du Code

1. Chargement des Bibliothèques

Pour charger les bibliothèques nécessaires à notre projet, nous avons utilisé des outils tels **que TensorFlow et Keras** pour la **construction et l'entraînement du modèle CNN**. De plus, nous avons importé des modules comme **Matplotlib** pour la **visualisation des données** et **NumPy** pour la **manipulation des tableaux**, garantissant ainsi une analyse et **une préparation efficaces des données**.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import random
import tensorflow as tf

# Ignorer les avertissements de type FutureWarning
warnings.simplefilter(action='ignore', category=FutureWarning)

from sklearn.model_selection import train_test_split
from tensorflow import keras
# Pour l'augmentation de données
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Pour construire un modèle CNN couche par couche
from tensorflow.keras.models import Sequential
# Différents types de couches nécessaires pour le modèle CNN
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout
# Algorithme d'optimisation pour mettre à jour les poids du réseau pendant l'entraînement
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
```

2. Visualisation des Données

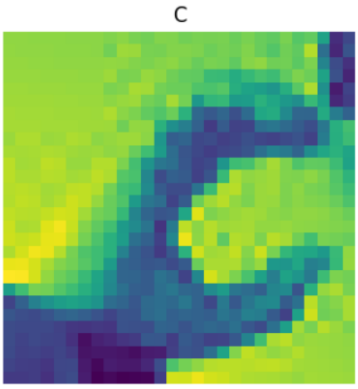
Nous avons exploré des **exemples d'images** représentatives de chaque classe **dans notre jeu de données,** offrant ainsi un aperçu de la **diversité des images et des catégories.** Cette analyse visuelle nous a aidés à mieux comprendre les caractéristiques et la répartition des données, ce qui facilite la **phase de modélisation.** À cette fin, nous avons utilisé des bibliothèques telles que Matplotlib pour visualiser les images et NumPy pour **manipuler les données,** générant ainsi des représentations graphiques informatives.

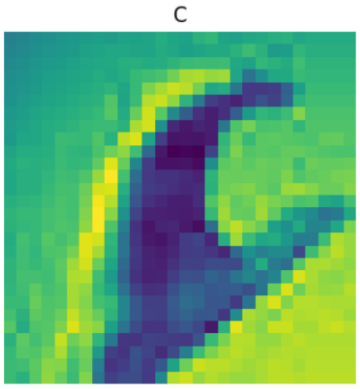
```
train_dir = 'Train' # Répertoire de formation
test_dir = 'Test' # Répertoire de test

# Obtenir la liste des répertoires de classes dans le répertoire de formation
class_directories = [os.path.join(train_dir, directory) for directory in sorted(os.listdir(train_dir))]

# Création de sous-graphiques pour afficher des exemples d'images
fig, axes = plt.subplots(3, 2, figsize=(10, 10))
```

```python
# Parcourir les répertoires de classes
for idx, class_dir in enumerate(class_directories):
    image_files = os.listdir(class_dir)

    # Vérifier s'il y a au moins 6 images à échantillonner
    if len(image_files) >= 6:
        random_images = random.sample(image_files, 6)
    else:
        # S'il y a moins de 6 images, utiliser toutes les images disponibles
        random_images = image_files

    for i, image_file in enumerate(random_images):
        image_path = os.path.join(class_dir, image_file)
        image = plt.imread(image_path)
        axes[int(i / 2), i % 2].imshow(image)
        axes[int(i / 2), i % 2].axis('off')
        axes[int(i / 2), i % 2].set_title(os.path.basename(class_dir))
    if idx == 2:
        break

plt.tight_layout()
plt.show()
```

3. Générateurs de Données

Nous avons employé ***ImageDataGenerator*** pour augmenter et **normaliser les données d'entraînement et de validation.** Cette approche nous a permis de générer de **nouvelles instances d'images** en temps réel, améliorant ainsi la robustesse et la **généralisation du modèle**. En utilisant ce générateur, nous avons appliqué des transformations telles que la rotation, le retournement horizontal, et le zoom aux images, enrichissant ainsi notre ensemble de données **pour une meilleure performance du modèle.**





4. Construction du Modèle

Nous avons construit un modèle de réseau de neurones convolutionnels (CNN) en **intégrant des couches spécifiques** conçues pour extraire des caractéristiques pertinentes à partir des données visuelles.

- Deux couches de **convolution** suivies **de max-pooling.**
- Une couche de **flatten** pour transformer **les matrices 2D en vecteurs.**
- Deux couches denses avec activation **ReLU**.
- Une couche dense finale avec activation **softmax** pour la classification des 24 classes.

```python
# Définition du modèle séquentiel CNN
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation ='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Conv2D(64,(3,3), activation ='relu'),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation = 'relu'),
    tf.keras.layers.Dense(256, activation = 'relu'),
    tf.keras.layers.Dense(24, activation='softmax')])

model.summary() # Afficher le résumé du modèle

metrics = ['accuracy'] # Définir la métrique pour évaluer le modèle

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=metrics)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 256)	409,856
dense_1 (Dense)	(None, 256)	65,792
dense_2 (Dense)	(None, 24)	6,168

```
Total params: 500,632 (1.91 MB)
Trainable params: 500,632 (1.91 MB)
Non-trainable params: 0 (0.00 B)
```

5. Compilation et Entraînement

Pour la compilation et **l'entraînement du modèle,** nous avons utilisé l'optimiseur **Adam**, un **algorithme d'optimisation** efficace pour les réseaux neuronaux, et la **fonction de perte *sparse_categorical_crossentropy***. L'entraînement s'est déroulé sur 10 époques, ce qui correspond à **10 passages complets du jeu de données à travers le réseau neuronal** pour **ajuster les poids** et minimiser la perte. Ce processus itératif a permis au modèle d'apprendre à **reconnaître les motifs** et **les caractéristiques des gestes** à partir des images d'entraînement, améliorant ainsi sa capacité à généraliser et à **effectuer des prédictions précises sur de nouvelles données.**

```
# Entraînement du modèle
history = model.fit(
    train_gen,
    epochs=10,
    validation_data = valid_gen)
```

6. Évaluation du Modèle

Pour évaluer le modèle, nous l'avons testé sur **un jeu de données distinct** appelé jeu de données de test, qui n'a **pas été utilisé pendant l'entraînement** ni la validation. Nous avons mesuré sa performance en termes de précision, qui représente **le pourcentage d'images correctement classées** par rapport au nombre total d'images dans le jeu de données de test.

```
# Évaluation du modèle sur l'ensemble de test
predics = model.evaluate(test_gen)
```

Les **résultats obtenus** indiquent une **précision de 99,71%** sur les données d'entraînement et de validation, ce qui montre que le modèle a très bien appris à partir de ces données. Cependant, **la précision sur les données de test** est **légèrement inférieure,** à 91,91%, ce qui peut indiquer que le modèle n'a pas généralisé parfaitement aux nouvelles données ou qu'il y a des différences subtiles entre les données **d'entraînement/validation** et les **données de test.**

7. Analyse et Interprétation

Les résultats obtenus sont satisfaisants avec une **précision élevée** sur les jeux de données **d'entraînement**, de **validation** et de **test**. Cependant, une légère diminution de la précision sur les données de test peut être due à une variété de facteurs, y compris la **complexité des gestes** et **les variations d'images.**

III. Conclusion

En conclusion, ce projet démontre **l'efficacité des réseaux de neurones convolutionnels** pour la tâche de **reconnaissance de la langue des signes** américaine. Les techniques de prétraitement et d'augmentation des données se sont avérées **cruciales** pour obtenir des **performances élevées.** Les résultats obtenus peuvent être **améliorés par des ajustements supplémentaires** du modèle et par **l'expérimentation avec d'autres architectures et hyperparamètres.**

Références

Sign Language MNIST, Kaggle: Sign Language MNIST Dataset

Documentation de TensorFlow et Keras.