Campus Monterrey

Inteligencia artificial avanzada para la ciencia de datos II (Gpo 501)

Momento de Retroalimentación Individual: Implementación de un modelo de Deep Learning.

Juan Pablo Castañeda Serrano A01752030

Overview

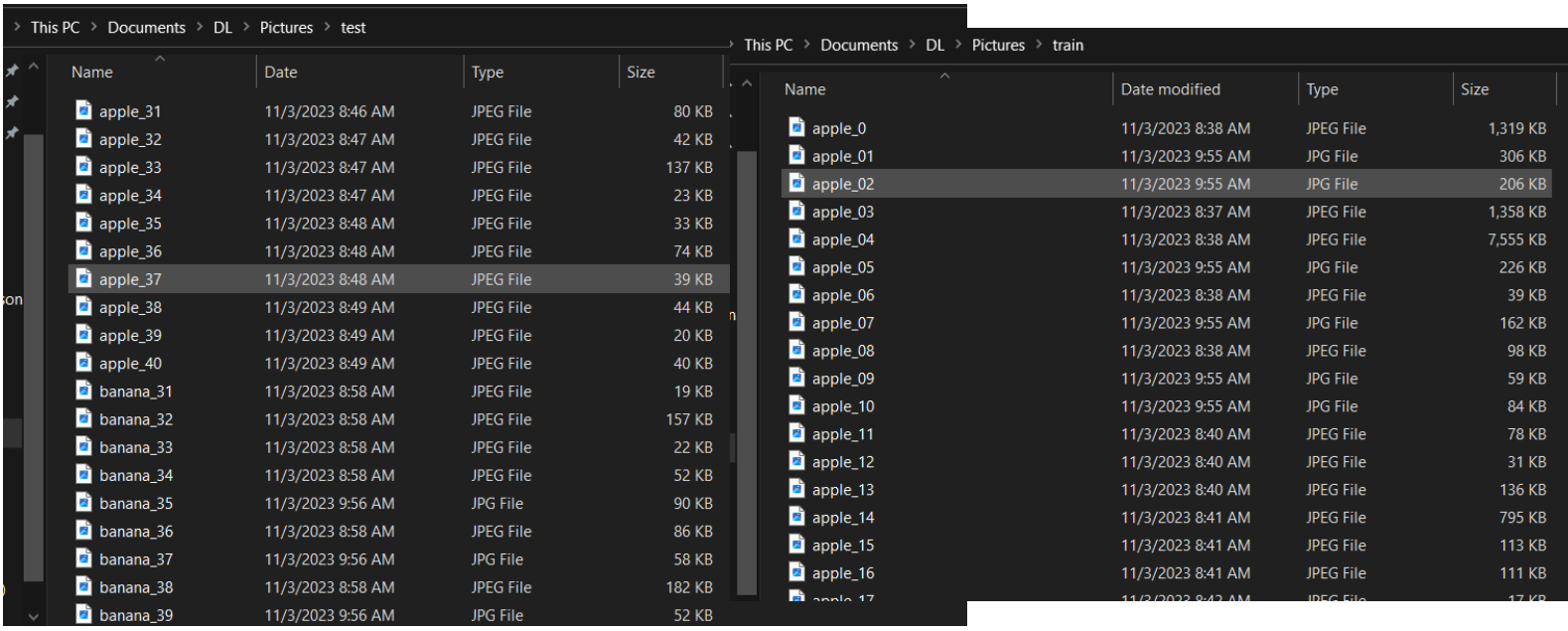
La interfaz, el modelo entrenado y el código del modelo se encuentran aquí:
https://github.com/a01752030/FruitRecognition

Este proyecto se divide en 3 partes, la codificación del modelo que se va usar, la codificación de la interfaz gráfica para el uso de usuarios y la pool de imágenes que se usó como base de datos de alimentación para el modelo. A continuación veremos cada una de las partes a detalle:

Base de datos

Para efectos prácticos, se descargaron varias imágenes de licencia de libre uso sobre frutas. En principio 3, estas son de naranjas, manzanas y de plátanos. Esta base de datos fue completamente realizada desde 0, esto para evitar conflictos de derechos de autor.

Esta base de datos está dividida en 2 carpetas. La carpeta con más imágenes "train" y la que tiene menos elementos "test", la primera será dividida en sí misma posteriormente para que entre esas 3 partes podamos tener conjuntos de train, validate y test respectivamente. La carpeta de "train" contiene 90 imágenes de las frutas, y la de "test" contiene 30 imágenes.



Modelo de predicción

Esta parte de igual manera está partida en 2, la primera será la explicación del código así como el flujo en cómo este funciona, y posteriormente la justificación de cada uno de los parámetros elegidos y como se llegó a ese ajuste.

Código

Este código en Python realiza varias tareas relacionadas con la clasificación de imágenes utilizando la biblioteca Keras y una red neuronal pre-entrenada VGG16. A continuación, se explicará cada parte del código:

1. **Importación de módulos**: Importa las bibliotecas y módulos necesarios para el proyecto, incluyendo NumPy para operaciones matemáticas, Pandas para manejo de datos, Matplotlib para visualización, OpenCV (cv2) para procesamiento de imágenes y Keras para construir y entrenar modelos de redes neuronales.

```python
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import cv2
import numpy as np
import cv2
from keras.layers import Dense, Conv2D, Flatten, MaxPool2D, Dropout, GlobalAveragePooling2D
from keras.models import Sequential, Model
from keras.applications import VGG16
from sklearn.model_selection import train_test_split
```

2. **Carga de imágenes de entrenamiento**: Carga imágenes de entrenamiento desde un directorio, redimensiona las imágenes a un tamaño específico y almacena las imágenes y etiquetas en listas separadas.

```python
np.random.seed(1)

train_images = []
train_labels = []
shape = (200, 200)
train_path = './Pictures/train'

for filename in os.listdir('./Pictures/train'):
    if filename.split('.')[1] == 'jpeg' or filename.split('.')[1] == 'jpg':
        img = cv2.imread(os.path.join(train_path, filename))

        train_labels.append(filename.split('_')[0])

        img = cv2.resize(img, shape)

        train_images.append(img)
```

3. **One-Hot Encoding de las etiquetas**: Convierte las etiquetas en una matriz dispersa codificada en caliente (one-hot encoding), donde cada etiqueta se representa como un vector binario.

```python
train_labels = pd.get_dummies(train_labels).values
```

4. **División de datos de entrenamiento y validación**: Divide los datos de entrenamiento en conjuntos de entrenamiento y validación para su posterior uso en el entrenamiento del modelo.

```
train_images = np.array(train_images)

x_train, x_val, y_train, y_val = train_test_split(train_images, train_labels, random_state=1)
```

5. **Carga de imágenes de prueba**: Realiza una tarea similar a la carga de imágenes de entrenamiento, pero para las imágenes de prueba. Sin embargo, las etiquetas de prueba no se utilizan para entrenar el modelo.

```
test_images = []
test_labels = []
shape = (200, 200)
test_path = './Pictures/test'

for filename in os.listdir('./Pictures/test'):
    if filename.split('.')[1] == 'jpeg' or filename.split('.')[1] == 'jpg':
        img = cv2.imread(os.path.join(test_path, filename))

        test_labels.append(filename.split('_')[0])
        img = cv2.resize(img, shape)

        test_images.append(img)

test_images = np.array(test_images)
```

6. **Visualización de imágenes de entrenamiento**: Muestra dos imágenes de entrenamiento en una ventana de Matplotlib.

```
plt.imshow(train_images[0])

plt.imshow(train_images[4])
```

7. **Carga de un modelo pre-entrenado VGG16**: Carga el modelo VGG16 pre-entrenado con pesos de 'imagenet' y excluye las capas superiores (top layers). Aquí empieza la secuencia de **transfer learning**

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(200, 200, 3))
```

8. **Congelación de las capas pre-entrenadas**: Congela los pesos de las capas pre-entrenadas para que no se modifiquen durante el entrenamiento.

```
for layer in base_model.layers:
    layer.trainable = False
```

9. **Agregación de capas adicionales para clasificación**: Agrega capas de Global Average Pooling y capas completamente conectadas para realizar la clasificación de las imágenes.

```
x = base_model.output
x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(train_labels.shape[1], activation='softmax')(x)
```

10. **Creación del modelo final**: Combina las capas pre-entrenadas con las capas de clasificación para formar el modelo final.
11. **Compilación del modelo**: Compila el modelo especificando la función de pérdida, el optimizador y las métricas a utilizar durante el entrenamiento.
12. **Entrenamiento del modelo**: Entrena el modelo utilizando los datos de entrenamiento y valida el rendimiento en los datos de validación. Guarda el historial de entrenamiento en la variable history.

```
model = Model(inputs=base_model.input, outputs=predictions)


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

history = model.fit(x_train, y_train, epochs=50, batch_size=50, validation_data=(x_val, y_val))
```

```
Epoch 1/50
2/2 [==============================] - 11s 5s/step - loss: 5.6535 - accuracy: 0.2647 - val_loss: 1.4310 - val_accuracy: 0.6522
Epoch 2/50
2/2 [==============================] - 10s 5s/step - loss: 0.9130 - accuracy: 0.6471 - val_loss: 0.2251 - val_accuracy: 0.9130
Epoch 3/50
2/2 [==============================] - 9s 4s/step - loss: 0.0545 - accuracy: 0.9853 - val_loss: 0.0123 - val_accuracy: 1.0000
Epoch 4/50
2/2 [==============================] - 9s 4s/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 0.0032 - val_accuracy: 1.0000
Epoch 5/50
```

13. **Visualización de métricas de entrenamiento**: Muestra gráficos de la precisión (accuracy) y la pérdida (loss) en función del número de épocas durante el entrenamiento.
14. **Guardado del modelo entrenado**: Guarda el modelo entrenado en un archivo con nombre 'fruit_prediction_transfer.h5'.

```
model.save('fruit_prediction_transfer.h5')
```

15. **Evaluación del modelo en datos de validación**: Evalúa el rendimiento del modelo en los datos de validación y muestra la precisión y la pérdida en la consola.

```
[0.00031999778002500534, 1.0]
```

Los resultados de la evaluación muestran que el modelo está funcionando excepcionalmente bien en el conjunto de datos de validación, con una pérdida muy baja y una precisión perfecta. Esto sugiere que el modelo ha aprendido a hacer predicciones precisas sobre los datos de validación.

Este código completo se utiliza para entrenar un modelo de clasificación de imágenes de frutas y evaluar su rendimiento en un conjunto de validación. También incluye la capacidad de hacer predicciones en imágenes de prueba individuales.

Justificación de parámetros

1. **Tamaño de imagen (shape)**: La elección del tamaño de imagen (200x200 píxeles) es un compromiso entre la calidad de la información visual y los recursos computacionales requeridos. Imágenes más grandes pueden contener más detalles, pero también requieren más memoria y tiempo de cálculo. El tamaño de 200x200 píxeles es lo suficientemente grande para retener detalles importantes en las imágenes de frutas.
2. **Modelo base pre-entrenado (VGG16)**: Utilizar un modelo pre-entrenado como VGG16 tiene sentido cuando se trabaja con conjuntos de datos relativamente pequeños, ya que permite aprovechar el conocimiento previo aprendido en un conjunto de datos grande (ImageNet). Además, VGG16 es conocido por su capacidad de extracción de características, lo que lo hace adecuado para tareas de clasificación de imágenes.
3. **Global Average Pooling (GAP)**: La adición de una capa de GAP después de las capas pre-entrenadas es una técnica común para reducir la dimensionalidad y extraer características importantes. GAP reduce el número de parámetros en la red y ayuda a evitar el sobreajuste.
4. **Capas densamente conectadas**: La elección de capas densamente conectadas (Fully Connected Layers) después de la capa de GAP depende de la complejidad de la tarea. En este caso, se han utilizado dos capas densamente conectadas con activación ReLU, que son comunes en las arquitecturas de redes neuronales para clasificación.
5. **Función de pérdida (categorical_crossentropy)**: La función de pérdida categorical_crossentropy es adecuada para problemas de clasificación multiclase donde cada imagen puede pertenecer a una de varias clases.
6. **Optimizador (Adam)**: El optimizador Adam es una elección común debido a su eficiencia en el entrenamiento de redes neuronales. A menudo, funciona bien en una variedad de tareas y puede converger más rápido que otros optimizadores.

7. **Número de épocas y tamaño de lote (epochs, batch_size)**: El número de épocas (50) y el tamaño de lote (50) fueron elegidos según el analizar la accuracy en cada una de las épocas y llegar a un número redondo que diera resultados satisfactorios

Interfaz Gráfica

Esta sección es más corta debido a que la interfaz es bastante simple. Simplemente corra el código "UI.py" y una ventana emergente le pedirá que suba una imagen. Después de esto puede seleccionar una imagen de una fruta y en la interfaz dirá cual es la predicción del modelo: