CNN dataSet Chess Image

Enlace al cuaderno en github: [bigData/1CEIABDTA - 7RO/Programación de Inteligencia Artificial/TAREA 6 CNN (Chess Data Set) at main · Javiert54/bigData · GitHub]

Importamos las librerías que vamos a usar:

```
import cv2
import os
import numpy as np
import keras
import matplotlib.pyplot as plt
import glob
import shutil
import zipfile
import os
```

Creamos una función para eliminar carpetas y extraer el zip para cada vez que ejecutemos el cuaderno:

```
def eliminar_carpetas(carpeta):
    if os.path.exists(carpeta):
        shutil.rmtree(carpeta)
        print(f"Carpeta '{carpeta}' eliminada.")
    else:
        print(f"La carpeta '{carpeta}' no existe.")

def extraer_zip(archivo_zip, destino):
    with zipfile.ZipFile(archivo_zip, 'r') as zip_ref:
        zip_ref.extractall(destino)
        print(f"Archivo '{archivo_zip}' extraído en '{destino}'.")

# Ejemplo de uso
carpeta_a_eliminar = 'ruta/a/tu/carpeta'
archivo_zip = 'ruta/a/tu/archivo.zip'
destino_extraccion = 'ruta/a/tu/destino'

eliminar_carpetas("Chessman-image-dataset")
extraer_zip("archive.zip", ".")



Carpeta 'Chessman-image-dataset' eliminada.
Archivo 'archive.zip' extraído en '.'.
```

Establecemos las equivalencias entre las clases y cada figura de ajedrez y seleccionamos un tamaño de imagen de 150x150 píxeles:

```
#Hacemos una relación entre los nombres de las imágenes y las clases
MAP_CHARACTERS = { 0: 'Bishop', 1: 'King', 2: 'Knight', 3: 'Pawn', 4: 'Queen', 5: 'Rook' }
# Vamos a standarizar todas las imágenes a tamaño 64x64
IMG_SIZE = 150
```

Creamos la función para cargar los datos de entrenamiento:

```
def load_train_set(dirname, map_characters, verbose=True):
    """Esta función carga los datos de training en imágenes.

    Como las imágenes tienen tamaños distintas, utilizamos la librería opencv
    para hacer un resize y adaptarlas todas a tamaño IMG_SIZE x IMG_SIZE.

    Args:
        dirname: directorio completo del que leer los datos
        map_characters: variable de mapeo entre labels y personajes
        verbose: si es True, muestra información de las imágenes cargadas

    Returns:
        X, y: X es un array con todas las imágenes cargadas con tamaño
    IMG SIZE x IMG SIZE
                y es un array con las labels de correspondientes a cada imagen
    """
    X_train = []
    y_train = []
    for label, character in map_characters.items():
        files = os.listdir(os.path.join(dirname, character))
        images = [file for file in files if file.endswith("jpg")]
        if verbose:
            print("Leyendo {} imágenes encontradas de {}".format(len(images), character))
        for image_name in images:
            image = cv2.imread(os.path.join(dirname, character, image_name))
            X_train.append(cv2.resize(image,(IMG_SIZE, IMG_SIZE)))
            y_train.append(label)
    return np.array(X_train), np.array(y_train)

def load_test_set(dirname, map_characters, verbose=True):
    """Esta función funciona de manera equivalente a la función load_train_set
    pero cargando los datos de test."""
    X_test = []
    y_test = []
    reverse_dict = {v: k for k, v in map_characters.items()}
    for filename in glob.glob(dirname + '/*.*'):
        char_name = "_".join(filename.split('/')[-1].split('_')[:-1])
        if char_name in reverse_dict:
            image = cv2.imread(filename)
            image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
            X_test.append(image)
            y_test.append(reverse_dict[char_name])
    if verbose:
        print("Leídas {} imágenes de test".format(len(X_test)))
    return np.array(X_test), np.array(y_test)
```

Y otra para cargar los datos de test:

```python
def load_test_set(dirname, map_characters, verbose=True):
    """Esta función funciona de manera equivalente a la función load_train_set
    pero cargando los datos de test."""
    X_test = []
    y_test = []
    reverse_dict = {v: k for k, v in map_characters.items()}

    # Usa el patrón ** para incluir subcarpetas y recursive=True
    folder = glob.glob(dirname + '/**/*.*', recursive=True)

    for filename in folder:
        char_name = filename.split('\\')[-2]
        if char_name in reverse_dict:
            image = cv2.imread(filename)
            image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
            X_test.append(image)
            y_test.append(reverse_dict[char_name])

    if verbose:
        print("Leídas {} imágenes de test".format(len(X_test)))

    return np.array(X_test), np.array(y_test)
```

También creamos una función para dividir el dataset:

```python
import shutil
from sklearn.model_selection import train_test_split


def split_dataset(source_dir, dest_dir, split_ratio=0.3):
    """
    Esta función divide el dataset en dos partes, una para training y otra para test, y lo pone en directorios distintos.
    """
    if not os.path.exists(dest_dir):
        os.makedirs(dest_dir)

    for character in os.listdir(source_dir):
        character_path = os.path.join(source_dir, character)
        if os.path.isdir(character_path):
            images = [file for file in os.listdir(character_path) if file.endswith("jpg")]
            train_images, test_images = train_test_split(images, test_size=split_ratio, random_state=42)

            character_dest_path = os.path.join(dest_dir, character)
            if not os.path.exists(character_dest_path):
                os.makedirs(character_dest_path)

            for image in test_images:
                shutil.move(os.path.join(character_path, image), os.path.join(character_dest_path, image))

source_directory = ".\\Chessman-image-dataset\\Chess"
destination_directory = ".\\Chessman-image-dataset\\Chess_test"
split_dataset(source_directory, destination_directory)
```

Cargamos todas las imágenes:

```
DATASET_TRAIN_PATH_COLAB = ".\\Chessman-image-dataset\\Chess"
DATASET_TEST_PATH_COLAB = "Chessman-image-dataset\\Chess_test"
X, y = load_train_set(DATASET_TRAIN_PATH_COLAB, MAP_CHARACTERS)
X_t, y_t = load_test_set(DATASET_TEST_PATH_COLAB, MAP_CHARACTERS)
X = X / 255.0
X_t = X_t / 255.0
# Vamos a barajar aleatoriamente los datos. Esto es importante ya que si no
# lo hacemos y, por ejemplo, cogemos el 20% de los datos finales como validation
# set, estaremos utilizando solo un pequeño número de personajes, ya que
# las imágenes se leen secuencialmente personaje a personaje.
perm = np.random.permutation(len(X))
X, y = X[perm], y[perm]
```

✓ 3.2s

```
Leyendo 49 imágenes encontradas de Bishop
Leyendo 42 imágenes encontradas de King
Leyendo 65 imágenes encontradas de Knight
Leyendo 60 imágenes encontradas de Pawn
Leyendo 46 imágenes encontradas de Queen
Leyendo 60 imágenes encontradas de Rook
Leídas 143 imágenes de test
```

Definimos la capa convolucional:

```
filters = 32  # Valor de ejemplo para filtros
kernel_size = (3, 3)  # Valor de ejemplo para el tamaño del kernel

conv_layer = keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None
)
```

Compilamos el modelo:

```python
from keras.regularizers import l2

# Construcción del modelo secuencial
model = keras.Sequential([
    keras.layers.Conv2D(32, kernel_size=(3, 3), data_format="channels_last", activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3), kernel_regularizer=l2(0.01)),
    keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu', kernel_regularizer=l2(0.01)),
    keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu', kernel_regularizer=l2(0.01)),
    keras.layers.Flatten(),
    keras.layers.Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(6, activation='softmax')
])

# Resumen del modelo
model.summary()

# Compilación del modelo
from keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

1.3s

Separamos los datos de traning de los de validación:

```python
from sklearn.model_selection import train_test_split

# Separamos los datos en training y validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Training data shape: {X_train.shape}")
print(f"Validation data shape: {X_val.shape}")
```

✓ 0.0s

```
Training data shape: (257, 150, 150, 3)
Validation data shape: (65, 150, 150, 3)
```

Definimos algunas variables que usaremos más adelante:

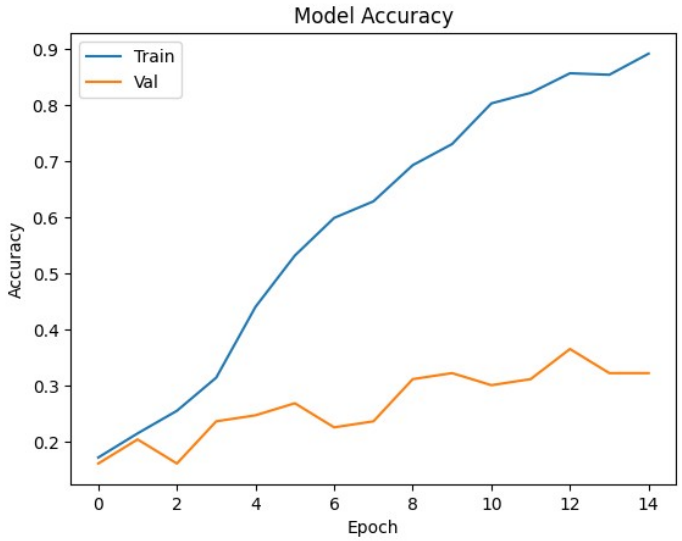
```python
#batch_size=128
num_classes=6
epochs=15
img_rows,img_cols=IMG_SIZE,IMG_SIZE
input_shape=(img_rows,img_cols,3) #(64,64,3)
# convert class vectors to binary class matrices
y=keras.utils.to_categorical(y,num_classes)
y_t=keras.utils.to_categorical(y_t,num_classes)
```

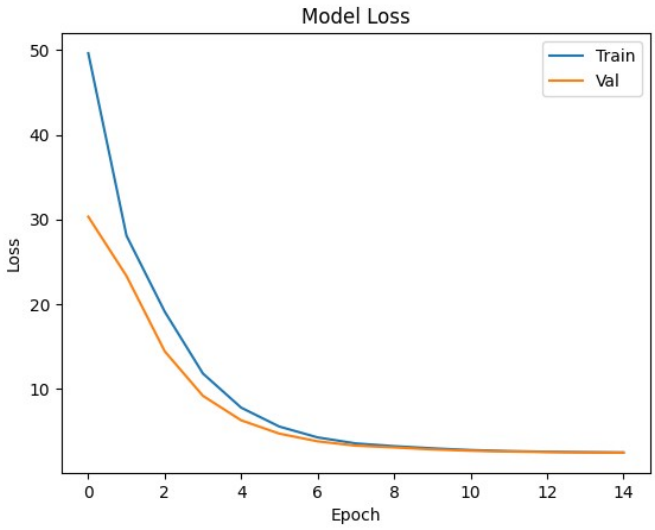
Entrenamos el modelo:

```python
from keras.utils import to_categorical

# Aseguramos que y_train y y_val estén codificados en one-hot
y_train = to_categorical(y_train, num_classes)
y_val = to_categorical(y_val, num_classes)

# Entrenamiento del modelo
history = model.fit(X_train, y_train, epochs=15, validation_data=(X_val, y_val))
```

Y creamos las funciones para las gráficas de pérdidas y precisión:

```python
# Función para graficar la precisión del modelo
def plot_acc(history, title="Model Accuracy"):
    """Imprime una gráfica mostrando la accuracy por epoch obtenida en un entrenamiento"""
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

# Función para graficar la pérdida del modelo
def plot_loss(history, title="Model Loss"):
    """Imprime una gráfica mostrando la pérdida por epoch obtenida en un entrenamiento"""
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    plt.show()

# Graficar la precisión y la pérdida del modelo
plot_acc(history)
plot_loss(history)
```





Creo que la baja precisión que he obtenido se debe al reducido dataset que he usado.