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

Alumno: Luis Ángel Guzmán Iribe

Matricula: A01741757

Clase: TC3007C.501 Inteligencia Artificial Avanzada para la Ciencia de Datos II

En este documento se implementa un modelo de DL utilizando Tensorflow / Keras con la finalidad de reconocer ambientes de imagenes tomadas por satelites, clasificadas en las siguientes categorías:

- agricultural
- buildings
- freeway
- mediumresidential
- river
- tenniscourt
- airplane
- chaparral
- golfcourse
- mobilehomepark
- runway
- baseballdiamond
- denseresidential
- harbor
- overpass
- sparseresidential
- beach
- forest
- intersection
- parkinglot
- storagetanks

Se generan 2 modelos diferentes. El primero de ellos de 0, y luego se utiliza el modelo VGG16 entrenado sobre ImageNet para mejorar el resultado del modelo nuevo.

Configuración de Google Drive

Aquí podemos observar los directorios en los que se consentra la información. El dataset original puede ser encontrado aquí, en resumen se cuentan con 21 categorías en total que describen diferentes tipos de terreno, cada categoría cuenta con 100 imagenes de 256x256 px.

```
from google.colab import drive
drive.mount('/content/drive')

# data: http://weegee.vision.ucmerced.edu/datasets/landuse.html

%cd "/content/drive/MyDrive/Colab Notebooks/satelite data/UCMerced_LandUse/Images"
!pwd
!ls
```

```
Mounted at /content/drive
/content/drive/MyDrive/Colab Notebooks/satelite data/UCMerced_LandUse/Images
/content/drive/MyDrive/Colab Notebooks/satelite data/UCMerced_LandUse/Images
agricultural     buildings         freeway       mediumresidential  river
airplane         chaparral         golfcourse    mobilehomepark     runway
baseballdiamond  denseresidential  harbor        overpass           sparseres
beach            forest            intersection  parkinglot         storageta
```

Importación de librerías

```
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

Carga, aumento y separación de datos

Aquí se acceden a los datos mencionados anteriormente, en la misma carpeta de drive que se encuentra este archivo. Además de dividir los datos en subconjuntos de entrenamiento (80%) y validación (20%), se generan nuevos datos a partir de los existentes, en un proceso que se conoce como "data augmentation" en el cual se cambia el zoom, altura, longitud, rotación y demás caracteristicas de las imagenes con la finalidad de que el modelo se ajuste mejor a perturbaciones no presentes en el dataset original.

```
# Paso 1: Cargar los datos
data_dir = "/content/drive/MyDrive/Colab Notebooks/satelite data/UCMerced_LandUse
class_names = os.listdir(data_dir)
class_names.sort()
num_classes = len(class_names)

# Separador de datos
batch_size = 32
train_datagen = ImageDataGenerator(
    rescale=1./256,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2 # Split 80 / 20
    )

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(256, 256),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(256, 256),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 1680 images belonging to 21 classes.
Found 420 images belonging to 21 classes.
```

Creación del modelo

En esta sección, se crea la base del modelo que se usará para realizar predicciones sobre las categorías. Este modelo se trata de una red neurunal convolucional (o CNN por sus iniciales en inglés) con 3 capas que aplicas 32, 64 y 128 filtros respectivamente, los outputs de estas capas despues son "aplanados" a un vector unidimencional que contiene las probabildiades de que un terminado input corresponda a una terminada categoría. Esta arquitectura suele ser empelada en problemas de clasificación de imagenes, que es precisamente el tipo de problema que se trata de atacar en esta situación.

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(256, 256, 3)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

Entrenamiento del modelo

Se entrena el modelo creado previamente con los subconjuntos de entrenamiento y validación.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
model.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
Epoch 1/10
53/53 [==============================] - 368s 7s/step - loss: 2.9685 - accurac
Epoch 2/10
53/53 [==============================] - 39s 739ms/step - loss: 2.8006 - accu
Epoch 3/10
53/53 [==============================] - 39s 734ms/step - loss: 2.6178 - accu
Epoch 4/10
53/53 [==============================] - 39s 735ms/step - loss: 2.4478 - accu
Epoch 5/10
53/53 [==============================] - 39s 739ms/step - loss: 2.3728 - accu
Epoch 6/10
53/53 [==============================] - 42s 797ms/step - loss: 2.2379 - accu
Epoch 7/10
53/53 [==============================] - 40s 766ms/step - loss: 2.2004 - accu
Epoch 8/10
53/53 [==============================] - 40s 763ms/step - loss: 2.0769 - accu
Epoch 9/10
53/53 [==============================] - 43s 811ms/step - loss: 2.0048 - accu
Epoch 10/10
53/53 [==============================] - 41s 781ms/step - loss: 1.9060 - accu
<keras.src.callbacks.History at 0x7d12c9e11060>
```

Precisión del modelo

Se genera un nuevo subconjunto de datos de prueba, con la finalidad de conocer algo más cercano a la precisión real del modelo. En este caso, la precisión del modelo es de 31.19, lo cual está muy por debajo de las expectativas de cualquier modelo. En pruebas anteriores, modificando el numero de epocas se logró una precisión de 50%, y aunque este valor es mejor, sigue habiendo posibilidad de mejorarlo.

```
test_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(256, 256),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Precisión de prueba: {test_accuracy*100:.2f}%")
```

```
Found 420 images belonging to 21 classes.
14/14 [==============================] - 8s 537ms/step - loss: 2.4071 - accura
Precisión de prueba: 31.19%
```

Transfer Learning

Dado que los resultados del modelo entrenado desde 0 pueden ser mejorados, se emplea la técnica de Transfer Learning (o entrenamiento transferido). Esta técnica consiste a grandes rasgos en hacer uso de un modelo general previamente entrenado al que se le agregan capas nuevas para cumplir con la tarea actual. En este caso, se emplea el modelo VGG16 entrenado sobre el dataset de ImageNet. Este modelo logró una precisión de XX%, mucho mejor que el máximo de 50% logrado anteriormente.

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(256, 256, 

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

model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
model.fit(train_generator, epochs=20, validation_data=validation_generator)

test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Precisión de prueba: {test_accuracy*100:.2f}%")
```

```
Epoch 1/20
53/53 [==============================] - 43s 798ms/step - loss: 2.7689 - accu
Epoch 2/20
53/53 [==============================] - 42s 791ms/step - loss: 1.9828 - accu
Epoch 3/20
53/53 [==============================] - 42s 787ms/step - loss: 1.7240 - accu
Epoch 4/20
53/53 [==============================] - 42s 788ms/step - loss: 1.5845 - accu
Epoch 5/20
53/53 [==============================] - 41s 784ms/step - loss: 1.5045 - accu
Epoch 6/20
53/53 [==============================] - 41s 783ms/step - loss: 1.3888 - accu
Epoch 7/20
53/53 [==============================] - 42s 788ms/step - loss: 1.3566 - accu
Epoch 8/20
53/53 [==============================] - 42s 788ms/step - loss: 1.3268 - accu
Epoch 9/20
53/53 [==============================] - 41s 783ms/step - loss: 1.2560 - accu
Epoch 10/20
53/53 [==============================] - 42s 795ms/step - loss: 1.2272 - accu
Epoch 11/20
53/53 [==============================] - 44s 835ms/step - loss: 1.2272 - accu
Epoch 12/20
53/53 [==============================] - 42s 783ms/step - loss: 1.1742 - accu
Epoch 13/20
53/53 [==============================] - 41s 779ms/step - loss: 1.2190 - accu
Epoch 14/20
53/53 [==============================] - 41s 768ms/step - loss: 1.1526 - accu
Epoch 15/20
53/53 [==============================] - 40s 762ms/step - loss: 1.1759 - accu
Epoch 16/20
53/53 [==============================] - 40s 757ms/step - loss: 1.1310 - accu
Epoch 17/20
53/53 [==============================] - 40s 756ms/step - loss: 1.1337 - accu
Epoch 18/20
53/53 [==============================] - 40s 759ms/step - loss: 1.2413 - accu
Epoch 19/20
53/53 [==============================] - 41s 768ms/step - loss: 1.1213 - accu
Epoch 20/20
53/53 [==============================] - 40s 764ms/step - loss: 1.1346 - accu
14/14 [==============================] - 9s 576ms/step - loss: 0.8589 - accura
Precisión de prueba: 70.24%
```