Clasificacion_Imagenes_Redes_Profundas_CNN

November 30, 2023

Alejandra Velasco Zárate A01635453

José Antonio Juárez Pacheco A0057218

José Carlos Yamuni Contreras A01740285

Juan Manuel Hernández Solano A00572208

[1]:
```
# Importar librerías

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.utils import load_img
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout,␣
↪BatchNormalization,LeakyReLU
import os
from skimage import color
from skimage.io import imread, imshow
from skimage.transform import resize
```

```
C:\Users\Alejandra Velasco\anaconda3\Lib\site-
packages\paramiko\transport.py:219: CryptographyDeprecationWarning: Blowfish has
been deprecated
  "class": algorithms.Blowfish,
```

1 Función para preprocesamiento de imágenes

[2]:
```
def preprocesamiento(datadir,categories, scale):
    flat_data_arr=[]
    target_arr=[]
    for i in categories:
        path=os.path.join(datadir,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img))
            img_array = color.rgb2gray(img_array)
            rgb_resized = resize(img_array, (int(1080/scale), int(1920/scale)))
            rgb_resized.tolist()
            flat_data_arr.append(rgb_resized)
            target_arr.append(categories.index(i))
        print(f'Categoría {i} cargada exitosamente')
    return flat_data_arr,target_arr
```

2 Conjunto de imágenes Fashion-MNIST

[3]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.
↪load_data()

x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)

x_train, x_test = x_train / 255.0, x_test /255.0
```

[4]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu', input_shape=(28,␣
↪28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation = 'relu'),
    tf.keras.layers.Dense(10, activation = 'softmax')
])
```

[5]:
```
model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy',␣
↪metrics = ['accuracy'])
```

[6]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 max_pooling2d (MaxPooling2  (None, 13, 13, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 128)               204928

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 243786 (952.29 KB)
Trainable params: 243786 (952.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[7]:
```
history = model.fit(x_train, y_train, epochs = 5, validation_data = (x_test,␣
↪y_test))
```

```
Epoch 1/5
1875/1875 [==============================] - 49s 26ms/step - loss: 0.4392 -
accuracy: 0.8428 - val_loss: 0.3329 - val_accuracy: 0.8788
Epoch 2/5
1875/1875 [==============================] - 53s 28ms/step - loss: 0.2936 -
accuracy: 0.8932 - val_loss: 0.3008 - val_accuracy: 0.8918
Epoch 3/5
1875/1875 [==============================] - 52s 28ms/step - loss: 0.2465 -
accuracy: 0.9082 - val_loss: 0.2727 - val_accuracy: 0.8990
Epoch 4/5
1875/1875 [==============================] - 47s 25ms/step - loss: 0.2136 -
accuracy: 0.9205 - val_loss: 0.2652 - val_accuracy: 0.9037
Epoch 5/5
1875/1875 [==============================] - 47s 25ms/step - loss: 0.1875 -
accuracy: 0.9291 - val_loss: 0.2523 - val_accuracy: 0.9101
```

[8]:
```
from sklearn.metrics import accuracy_score, recall_score

y_pred = model.predict(x_test)

y_pred_classes = np.argmax(y_pred, axis=1)

accuracy = accuracy_score(y_test, y_pred_classes)
```

```
print(f'Accuracy: {accuracy}')

class_names = {
    0: 'T-shirt/top',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Dress',
    4: 'Coat',
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}

labels = list(class_names.keys())
recall_per_class = recall_score(y_test, y_pred_classes, labels=labels,␣
 ↪average=None)

for label, recall in zip(labels, recall_per_class):
    class_name = class_names[label]
    print(f'Recall para la clase {class_name}: {recall}')
```

```
313/313 [==============================] - 2s 7ms/step
Accuracy: 0.9101
Recall para la clase T-shirt/top: 0.837
Recall para la clase Trouser: 0.985
Recall para la clase Pullover: 0.86
Recall para la clase Dress: 0.949
Recall para la clase Coat: 0.846
Recall para la clase Sandal: 0.984
Recall para la clase Shirt: 0.749
Recall para la clase Sneaker: 0.977
Recall para la clase Bag: 0.967
Recall para la clase Ankle boot: 0.947
```

3 Conjunto de imágenes satelitales

[46]:
```
# Cargar imágenes satelitales
datadir = 'data'
classes = ['Agua', 'Bosque', 'Ciudad', 'Cultivo', 'Desierto', 'Montaña']

# Llamar a la función de preprocesamiento de imágenes
x, y = preprocesamiento(datadir, classes, 8)
```

```
Categoría Agua cargada exitosamente
```

```
Categoría Bosque cargada exitosamente
Categoría Ciudad cargada exitosamente
Categoría Cultivo cargada exitosamente
Categoría Desierto cargada exitosamente
Categoría Montaña cargada exitosamente
```

[59]:
```
X = np.array(x)
y = np.array(y)

# Separar bases de datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0, shuffle = True)

# Reshape a los datos a una dimension mas para poder emplear el modelo de CNN.
X_train = X_train.reshape(-1, int(1080/8), int(1920/8), 1)
X_test = X_test.reshape(-1, int(1080/8), int(1920/8), 1)
```

[60]:
```
# Model configuration
num_classes = 6
input_shape = (int(1080/8), int(1920/8), 1)
y_categorical = to_categorical(y_train)
y_categorical_test = to_categorical(y_test)
batch_size = 100
no_epochs = 20

# CNN
clf = Sequential()
clf.add(Conv2D(64, kernel_size=(3, 3),activation='relu',input_shape =␣
 ↪input_shape))
clf.add(MaxPooling2D(2, 2))
clf.add(Dropout(0.10))
clf.add(Conv2D(128, (3, 3), activation='relu'))
clf.add(MaxPooling2D(pool_size=(2, 2)))
clf.add(Dropout(0.15))
clf.add(Conv2D(256, (3, 3), activation='relu'))
clf.add(MaxPooling2D(pool_size=(2, 2)))
clf.add(Dropout(0.15))
clf.add(Flatten())
clf.add(Dense(128, activation = 'relu'))
clf.add(Dense(num_classes, activation='softmax'))

# Compilación del modelo
clf.compile(loss='categorical_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])
```

[49]:
```
clf.summary()
```

```
Model: "sequential_29"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_114 (Conv2D)         (None, 135, 240, 64)      640

 max_pooling2d_114 (MaxPool  (None, 68, 120, 64)       0
 ing2D)

 dropout_112 (Dropout)       (None, 68, 120, 64)       0

 conv2d_115 (Conv2D)         (None, 68, 120, 128)      73856

 max_pooling2d_115 (MaxPool  (None, 34, 60, 128)       0
 ing2D)

 dropout_113 (Dropout)       (None, 34, 60, 128)       0

 conv2d_116 (Conv2D)         (None, 34, 60, 256)       295168

 max_pooling2d_116 (MaxPool  (None, 17, 30, 256)       0
 ing2D)

 dropout_114 (Dropout)       (None, 17, 30, 256)       0

 flatten_29 (Flatten)        (None, 130560)            0

 dense_36 (Dense)            (None, 128)               16711808

 dense_37 (Dense)            (None, 6)                 774

=================================================================
Total params: 17082246 (65.16 MB)
Trainable params: 17082246 (65.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[61]:
```python
# Evaluación con validación cruzada
n_splits = 5
kf = StratifiedKFold(n_splits=n_splits, shuffle = True)
k = 0
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(X, y):
    x_train = X[train_index, :]
    y_train = y[train_index]
    y_train_categorical = to_categorical(y_train, num_classes)
    x_test = X[test_index, :]
```

```
    y_test = y[test_index]
    y_test_categorical = to_categorical(y_test, num_classes)
    k = k + 1

    # Define CNN model
    clf_cv = Sequential()
    clf_cv.add(Conv2D(64, kernel_size=(3, 3),activation='relu',input_shape =␣
 ↪input_shape))
    clf_cv.add(MaxPooling2D(2, 2))
    clf_cv.add(Dropout(0.10))
    clf_cv.add(Conv2D(128, (3, 3), activation='relu'))
    clf_cv.add(MaxPooling2D(pool_size=(2, 2)))
    clf_cv.add(Dropout(0.15))
    clf_cv.add(Conv2D(256, (3, 3), activation='relu'))
    clf_cv.add(MaxPooling2D(pool_size=(2, 2)))
    clf_cv.add(Dropout(0.15))
    clf_cv.add(Flatten())
    clf_cv.add(Dense(128, activation = 'relu'))
    clf_cv.add(Dense(num_classes, activation='softmax'))

    clf_cv.compile(loss='categorical_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])
    clf_cv.fit(x_train, y_train_categorical,␣
 ↪batch_size=batch_size,epochs=no_epochs,verbose=0,validation_data=(x_test,␣
 ↪y_test_categorical))

    # Evaluate model using test data
    y_pred = np.argmax(clf_cv.predict(x_test), axis=-1)
    cv_y_test.append(y_test)
    cv_y_pred.append(y_pred)

    print('***** Pliegue ', k, ' terminado ****')

print(classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred)))
```

```
13/13 [==============================] - 7s 562ms/step
***** Pliegue  1  terminado ****
13/13 [==============================] - 8s 595ms/step
***** Pliegue  2  terminado ****
13/13 [==============================] - 6s 482ms/step
***** Pliegue  3  terminado ****
13/13 [==============================] - 6s 462ms/step
***** Pliegue  4  terminado ****
13/13 [==============================] - 6s 467ms/step
***** Pliegue  5  terminado ****
              precision    recall  f1-score   support
```

```
              0       0.88      0.85      0.87       335
              1       0.75      0.89      0.82       335
              2       0.78      0.96      0.86       336
              3       0.83      0.73      0.78       334
              4       0.92      0.88      0.90       334
              5       0.90      0.71      0.79       342

    accuracy                            0.84      2016
   macro avg       0.84      0.84      0.84      2016
weighted avg       0.84      0.84      0.84      2016
```

4 Conjunto de imágenes de verduras

[34]:
```
# Cargar imágenes satelitales
datadir = 'data_verduras'
classes = ['Cebolla', 'Chayote', 'Jitomate', 'Pepino', 'Zanahoria']

# Llamar a la función de preprocesamiento de imágenes
x, y = preprocesamiento(datadir, classes, 8)
```

```
Categoría Cebolla cargada exitosamente
Categoría Chayote cargada exitosamente
Categoría Jitomate cargada exitosamente
Categoría Pepino cargada exitosamente
Categoría Zanahoria cargada exitosamente
```

[35]:
```
X = np.array(x)
y = np.array(y)

# Separar bases de datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0, shuffle = True)
```

[37]:
```
# Reshape a los datos a una dimension mas para poder emplear el modelo de CNN.
X_train = X_train.reshape(-1, int(1080/8), int(1920/8), 1)
X_test = X_test.reshape(-1, int(1080/8), int(1920/8), 1)
```

[43]:
```
# Model configuration
num_classes = 5
input_shape = (int(1080/8), int(1920/8), 1)
y_categorical = to_categorical(y_train)
y_categorical_test = to_categorical(y_test)
batch_size = 100
no_epochs = 5
```

```
# CNN
clf = Sequential()
clf.add(Conv2D(64, kernel_size=(3, 3),activation='relu',input_shape =␣
↪input_shape))
clf.add(MaxPooling2D(2, 2))
clf.add(Dropout(0.10))
clf.add(Conv2D(128, (3, 3), activation='relu'))
clf.add(MaxPooling2D(pool_size=(2, 2)))
clf.add(Dropout(0.15))
clf.add(Conv2D(256, (3, 3), activation='relu'))
clf.add(MaxPooling2D(pool_size=(2, 2)))
clf.add(Conv2D(512, kernel_size=3, activation='relu'))
clf.add(MaxPooling2D(2, 2))
clf.add(Dropout(0.15))
clf.add(Flatten())
clf.add(Dropout(0.25))
clf.add(Dense(128, activation = 'relu'))
clf.add(Dense(num_classes, activation='softmax'))

# Compilación del modelo
clf.compile(loss='categorical_crossentropy', optimizer='Adam',␣
↪metrics=['accuracy'])
```

[45]:
```
clf.summary()
```

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_90 (Conv2D)          (None, 135, 240, 64)      640

 max_pooling2d_90 (MaxPooli  (None, 68, 120, 64)       0
 ng2D)

 dropout_88 (Dropout)        (None, 68, 120, 64)       0

 conv2d_91 (Conv2D)          (None, 68, 120, 128)      73856

 max_pooling2d_91 (MaxPooli  (None, 34, 60, 128)       0
 ng2D)

 dropout_89 (Dropout)        (None, 34, 60, 128)       0

 conv2d_92 (Conv2D)          (None, 34, 60, 256)       295168

 max_pooling2d_92 (MaxPooli  (None, 17, 30, 256)       0
 ng2D)
```

```
conv2d_93 (Conv2D)          (None, 17, 30, 512)       1180160

max_pooling2d_93 (MaxPooli  (None, 9, 15, 512)        0
ng2D)

dropout_90 (Dropout)        (None, 9, 15, 512)        0

flatten_23 (Flatten)        (None, 69120)             0

dropout_91 (Dropout)        (None, 69120)             0

dense_24 (Dense)            (None, 128)               8847488

dense_25 (Dense)            (None, 5)                 645

=================================================================
Total params: 10397957 (39.67 MB)
Trainable params: 10397957 (39.67 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[44]:
```python
# Evaluación con validación cruzada
n_splits = 5
kf = StratifiedKFold(n_splits=n_splits, shuffle = True)
k = 0
cv_y_test = []
cv_y_pred = []
for train_index, test_index in kf.split(X, y):
    x_train = X[train_index, :]
    y_train = y[train_index]
    y_train_categorical = to_categorical(y_train, num_classes)
    x_test = X[test_index, :]
    y_test = y[test_index]
    y_test_categorical = to_categorical(y_test, num_classes)
    k = k + 1

    # Define CNN model
    clf_cv = Sequential()
    clf_cv.add(Conv2D(64, kernel_size=(3, 3),activation='relu',input_shape =␣
 ↪input_shape))
    clf_cv.add(MaxPooling2D(2, 2))
    clf_cv.add(Dropout(0.10))
    clf_cv.add(Conv2D(128, (3, 3), activation='relu'))
    clf_cv.add(MaxPooling2D(pool_size=(2, 2)))
    clf_cv.add(Dropout(0.15))
    clf_cv.add(Conv2D(256, (3, 3), activation='relu'))
    clf_cv.add(MaxPooling2D(pool_size=(2, 2)))
```

```
    clf_cv.add(Conv2D(512, kernel_size=3, activation='relu'))
    clf_cv.add(MaxPooling2D(2, 2))
    clf_cv.add(Dropout(0.15))
    clf_cv.add(Flatten())
    clf_cv.add(Dropout(0.25))
    clf_cv.add(Dense(128, activation = 'relu'))
    clf_cv.add(Dense(num_classes, activation='softmax'))

    clf_cv.compile(loss='categorical_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])
    clf_cv.fit(x_train, y_train_categorical,␣
 ↪batch_size=batch_size,epochs=no_epochs,verbose=0,validation_data=(x_test,␣
 ↪y_test_categorical))

    # Evaluate model using test data
    y_pred = np.argmax(clf_cv.predict(x_test), axis=-1)
    cv_y_test.append(y_test)
    cv_y_pred.append(y_pred)

    print('***** Pliegue ', k, ' terminado *****')

print(classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred)))
```

```
16/16 [==============================] - 10s 617ms/step
***** Pliegue  1  terminado *****
16/16 [==============================] - 10s 612ms/step
***** Pliegue  2  terminado *****
16/16 [==============================] - 15s 918ms/step
***** Pliegue  3  terminado *****
16/16 [==============================] - 14s 880ms/step
***** Pliegue  4  terminado *****
16/16 [==============================] - 14s 890ms/step
***** Pliegue  5  terminado *****
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       501
           1       0.99      1.00      0.99       503
           2       0.97      0.88      0.92       524
           3       0.89      0.95      0.91       500
           4       0.97      0.99      0.98       500

    accuracy                           0.96      2528
   macro avg       0.96      0.96      0.96      2528
weighted avg       0.96      0.96      0.96      2528
```