

Tratamiento de datos

Nombre: Jorge Sampedro

<https://github.com/JorgeY2J/TratamientoDatos.git>

Introducción

Para este proyecto final se busca hacer un modelo que nos permita identificar los tipos de carnes que tenemos en una base de datos interna, para lo cual luego de analizar el caso se ha optado por el uso de una red neuronal convolucional (CNN).

CNN

Por el tipo de imágenes que posee la base de datos se consideró el CNN como el mejor método para llevarlo a cabo.

Debido a que por medio de las convoluciones nos permite tratar las imágenes como una matriz, a la vez segmentar y comparar segmentos de la imagen para buscar similitudes que para el ojo humano son imperceptibles.

Se hace una normalización para trabajar las imágenes con valores entre 0 y 1, de esta forma se cree que el modelo puede ser algo más efectivo y no causar overfitting al usar demasiados valores.

Con el modelo que se busca aplicar las primeras convoluciones se ocuparán de detectar características un poco básicas como líneas o curvas. Precisamente se escoge este modelo porque entre más convoluciones se hagan, características más complejas se podrán detectar.

Este método usa el comportamiento del ojo humano por lo que también se ha considerado útil para este proyecto.

Procedimiento

Vamos a crear una red neuronal convolucional utilizando keras.

Primero debemos importar las librerías que vamos a necesitar.

Numpy nos va a servir para realizar distintas operaciones, mientras que matplotlib nos sirve para mostrar las variables con las que estamos trabajando.

```
In [2]: import numpy as np
import os
import re
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
In [3]: import keras
```

```
In [4]: from keras.utils import to_categorical
```

```
In [20]: from tensorflow.keras.layers import (BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense)
```

```
In [6]: from tensorflow.keras.layers import LeakyReLU
```

Lo siguiente será cargar las imágenes que serán usadas para el preprocesamiento, en nuestro caso será una carpeta interna llamada train.

```
In [7]: dirname = os.path.join(os.getcwd(), 'train')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|png|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            image = plt.imread(filepath)
            images.append(image)
            b = "Leyendo..." + str(cant)
            print (b, end="\n")
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
                dircount.append(cant)
                cant=0
    dircount.append(cant)

dircount = dircount[1:]
dircount[0]=dircount[0]+1
print('Directorios leidos:',len(directories))
```

```
leyendo imagenes de C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_02 1
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_03 62
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_04 213
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_05 105
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_06 949
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_07 37
C:\Users\Jorge\Documents\Ciberseguridad\Tratamiento de datos\final\train\CLASS_08 204
Directorios leidos: 7
Imágenes en cada directorio [63, 213, 105, 949, 37, 204, 62]
suma Total de imágenes en subdirs: 1633
```

En la siguiente imagen se muestra como se crean las etiquetas de las carpetas de las distintas imágenes, le damos un valor de 0 a 7 según la carpeta en la que se encuentre.

```

In [8]: labels=[]
        indice=0
        for cantidad in dircount:
            for i in range(cantidad):
                labels.append(indice)
                indice=indice+1
        print('Cantidad etiquetas creadas: ',len(labels))

        carnes=[]
        indice=0
        for directorio in directories:
            name = directorio.split(os.sep)
            print(indice , name[len(name)-1])
            carnes.append(name[len(name)-1])
            indice=indice+1

        y = np.array(labels)
        X = np.array(images, dtype=np.uint8) #para crear la lista a numpy

        # Los número de las etiquetas
        classes = np.unique(y)
        nClasses = len(classes)
        print('Total number of outputs : ', nClasses)
        print('Output classes : ', classes)

```

```

Cantidad etiquetas creadas: 1633
0 CLASS_02
1 CLASS_03
2 CLASS_04
3 CLASS_05
4 CLASS_06
5 CLASS_07
6 CLASS_08
Total number of outputs : 7
Output classes : [0 1 2 3 4 5 6]

```

Teniendo listo todo esto, es preparar los datos de “train” y de “test”, es decir, hacer una permutación aleatoria y dividir los datos de entrenamiento y validación.

```

In [9]: #Permutación aleatoria entre datos de validación y de test
        train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.1)
        print('Training data shape : ', train_X.shape, train_Y.shape)
        print('Testing data shape : ', test_X.shape, test_Y.shape)

        Training data shape : (1469, 216, 384, 3) (1469,)
        Testing data shape : (164, 216, 384, 3) (164,)

```

```

In [10]: train_X = train_X.astype('float32')
        test_X = test_X.astype('float32')
        train_X = train_X / 255.
        test_X = test_X / 255.

        # Cambiar las etiquetas de categorial a one hot encoding
        train_Y_one_hot = to_categorical(train_Y)
        test_Y_one_hot = to_categorical(test_Y)

        plt.figure(figsize=[5,5])

        # Mostrar la primera imagen del entrenamiento en grises
        plt.subplot(121)
        plt.imshow(train_X[0,:,:], cmap='gray')
        plt.title("Ground Truth : {}".format(train_Y[0]))

```

```

Out[10]: Text(0.5, 1.0, 'Ground Truth : 3')

```

Aquí se propone dividir la base de datos en el 90% de entrenamiento y el 10% de test.

```

In [11]: #Mostrar el cambio
        print('Original label:', train_Y[0])
        print('After conversion to one-hot:', train_Y_one_hot[0])

        train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_hot, test_size=0.1, random_state=13)
        print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)

        Original label: 3
        After conversion to one-hot: [0. 0. 0. 1. 0. 0. 0.]
        (1322, 216, 384, 3) (147, 216, 384, 3) (1322, 7) (147, 7)

```

Aquí simplemente se preprocesaron los pixeles, como tienen un valor de 255 y lo que buscamos es un valor entre 0 y 1 solo lo dividimos entre 255(normalizar). Adicionalmente se hace el “one hot encoding” con “categorical”; es decir convertir las etiquetas.

Definir la CNN

```
In [12]: INIT_LR = 1e-3
epochs = 20
batch_size = 32

In [15]: from keras.models import Sequential

In [24]: from keras.layers import Convolution2D

In [30]: carnes_model = Sequential()
carnes_model.add(Convolution2D(32, kernel_size=(3, 3), activation='linear', padding='same', input_shape=(216,384,3)))
carnes_model.add(LeakyReLU(alpha=0.1))

In [31]: carnes_model.add(Flatten())
carnes_model.add(Dense(32, activation='linear'))
carnes_model.add(LeakyReLU(alpha=0.1))
carnes_model.add(Dropout(0.5))
carnes_model.add(Dense(nClasses, activation='softmax'))

In [32]: carnes_model.summary()

carnes_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adagrad(learning_rate=INIT_LR, decay=
```

Definimos la arquitectura de nuestra red neuronal convolucional, la entrada será una matriz de 216, 384; esta corresponde a las dimensiones de las imágenes en tres canales.

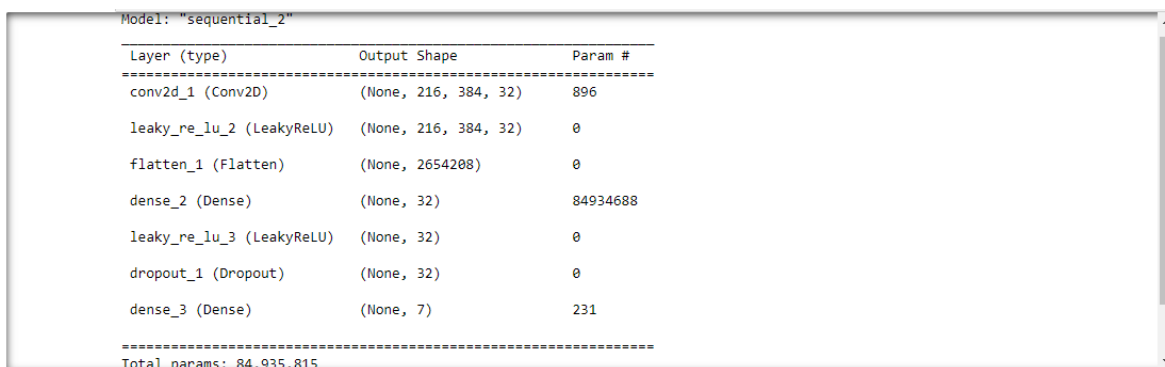
Va a pasar a través de filtros convolucionales de tamaño 3x3.

Luego esta pasará por una capa que nos permite unir la zona de la capa convolucional con la capa del full connected, después tenemos la etapa de full connected donde tenemos las capas densas y se puede ver la función de activación ReLu.

Esta pasará por un dropout la cual ayudará a evitar el overfitting.

Finalizamos la capa de salida con n neuronas con activación softmax para que corresponda con el hot encoding que se hizo antes, la capa de salida debe tener la función softmax para arrojar el resultado como si fueran probabilidades.

Para compilar se usó el optimizador Adagrad.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 216, 384, 32)	896
leaky_re_lu_2 (LeakyReLU)	(None, 216, 384, 32)	0
flatten_1 (Flatten)	(None, 2654208)	0
dense_2 (Dense)	(None, 32)	84934688
leaky_re_lu_3 (LeakyReLU)	(None, 32)	0
dropout_1 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 7)	231
Total params: 84,935,815		

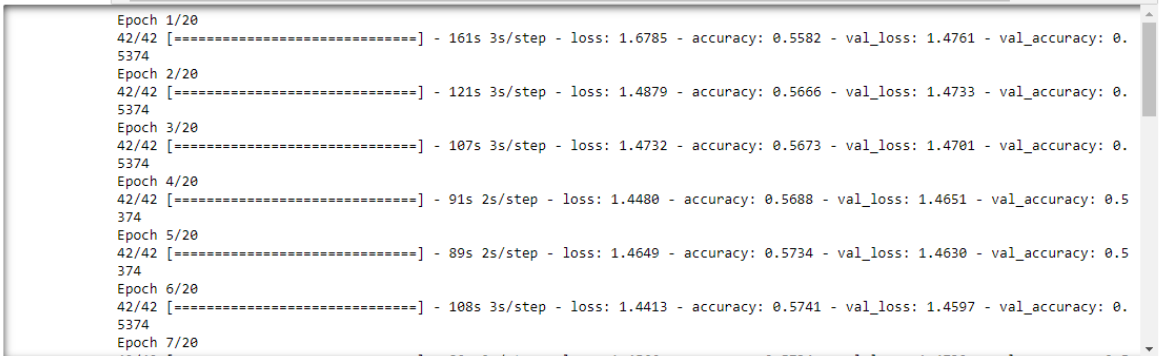
Se puede observar el resumen del modelo secuencial.

En flatten se transforma a una sola columna, como si fueran neuronas; estas estarán conectadas a la capa de full connected, aquí se observa conectada al 32(número de neuronas).

La capa de salida es siete debido a las posibilidades a las que puede pertenecer la imagen.

```
In [33]: carnes_train_dropout = carnes_model.fit(train_X, train_label, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(val_X, val_label))

# guardamos la red, para reutilizarla en el futuro, sin tener que volver a entrenar
carnes_model.save("carnes_mnist.h5py")
```



```
Epoch 1/20
42/42 [=====] - 161s 3s/step - loss: 1.6785 - accuracy: 0.5582 - val_loss: 1.4761 - val_accuracy: 0.5374
Epoch 2/20
42/42 [=====] - 121s 3s/step - loss: 1.4879 - accuracy: 0.5666 - val_loss: 1.4733 - val_accuracy: 0.5374
Epoch 3/20
42/42 [=====] - 107s 3s/step - loss: 1.4732 - accuracy: 0.5673 - val_loss: 1.4701 - val_accuracy: 0.5374
Epoch 4/20
42/42 [=====] - 91s 2s/step - loss: 1.4480 - accuracy: 0.5688 - val_loss: 1.4651 - val_accuracy: 0.5374
Epoch 5/20
42/42 [=====] - 89s 2s/step - loss: 1.4649 - accuracy: 0.5734 - val_loss: 1.4630 - val_accuracy: 0.5374
Epoch 6/20
42/42 [=====] - 108s 3s/step - loss: 1.4413 - accuracy: 0.5741 - val_loss: 1.4597 - val_accuracy: 0.5374
Epoch 7/20
42/42 [=====] - 100s 3s/step - loss: 1.4556 - accuracy: 0.5734 - val_loss: 1.4730 - val_accuracy: 0.5374
```

En la fase de entrenamiento se usan las variables definidas anteriormente, con un número de 20 epoch(épocas) y el batch_size de 32

```
In [35]: test_eval = carnes_model.evaluate(test_X, test_Y_one_hot, verbose=1)

print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
6/6 [=====] - 7s 511ms/step - loss: 1.2288 - accuracy: 0.6585
Test loss: 1.2287826538085938
Test accuracy: 0.6585366129875183
```

Con los resultados encontramos que nuestro modelo tuvo una efectividad de tan solo el 65.85%.

Conclusiones

El porcentaje de precisión no fue el deseado por lo que habría que realizar variaciones hasta que nuestro modelo tenga una mayor efectividad, quizá no usar una red neuronal convolucional sino usar un modelo lineal.