Reporte_CNN_Flores

June 13, 2022

1 Ejercicio Propuesto CNN Flores

Para el ejercicio se pidio crear un "Data Set Propio" con imagenes de distintas flores. Por ello realice una selección de 4 flores a identificar las cuales son: Dhalias, Marigold (Cempasuchil), Orquideas, Rosas Rojas.

Se obtuvieron entre 7,000 y 9,000 imagenes de cada flor, a partir de videos de Youtube extrayendo frame por frame del video y clasificandolo por color. Las imagenes tienen un tamaño de 50x50 px en color (RGB), con un total de fotos de 29,962.

Por lo tanto podemos concluir que el objetivo es que nuestra "Red Neuronal Convolucional" aprenda a clasificar, por sí misma, de qué tipo de flor se trata, dando una imagen con la cual no se entreno o testeo en la creación de nuestra red.

2 Generar Data Set

Para generar mi Data Set me apoye de videos de Timelaps de flores de Youtube, los cuales saque las imagenes frame por frame, para después por medio de mascaras en HSV sacar los colores necesarios de cada flor, aplicar una redimensión al 50x50.

Por ultimo con los frame ya realizados y redimensionados tome una imágen, la cual gire 360 grados y translade sobre el eje horizontal (x) dichos movimientos nos aumentaron en gran escala nuestro número de imégenes dentro de cada clase de nuestro DataSet

2.1 Importar Librerías

[ ]:
```
import cv2 as cv
import numpy as np
import math
```

2.2 Guardamos la imágen

En está función aplicamos la máscara de color a nuestro frame, lo redimensionamos a 50x50 pixeles y por ultimo lo guardamos, estás acciones están desglosadas y explicadas aquí abajo:

BITWISE_AND:Busca un punto de intersección entre dos imágenes, si ambas son de la misma intensidad aplica la misma intensidad al pixel

- PRIMER PARAMETRO PRIMERA IMAGEN
- SEGUNDO PARAMETRO SEGUNDA IMAGEN EN ESTE CASO ES LA MISMA

- TERCER PARAMETRO LA MASCARA DE COLOR QUE SE LA APLICARA

RESIZE: Redimensionar el tamaño de la imagen

- PRIMER PARAMETRO IMAGEN A REDIMENSIONAR
- SEGUNDO PARAMETRO TAMANIO DE REDIMENSION
- TERCER PARAMETRO TIPO DE INTERPOLACION, INTERPOLACION CUBICA GENERA UN PROMEDIO DEL PIXEL PARA SUSTITUIRLO CON UNO NUEVO

[ ]:
```
def guardar (mask1, frame, hsv, num_img):

    res=cv.bitwise_and(frame, frame, mask=mask1)
    new_img = cv.resize(res, dsize=(50, 50), interpolation=cv.INTER_CUBIC)

    #DIRECCION DE GUARDAR
    name='C:\\Users\\gilba\\Desktop\\IA\\Micke\\Flor\\img_'+str(num_img)+'.jpg'
    #GUARDAR
    cv.imwrite(name,new_img)
```

2.3 VideoCapture, Segmentación por Color

Está es nuestra función principal donde sacamos los frames del video, así como calculamos las máscaras de cada color por medio de np.array.

Cada linea de código contiene el comentario de lo que realiza, así como los parametros que se tiene que usar en cada función en caso de que fueran confusos, decidi agregarlo para mayor comprensión del código.

[ ]:
```
video=cv.VideoCapture("C:\\Users\\gilba\\Desktop\\IA\\Micke\\Flor.mp4")

#CONTADOR, NOMBRE UNICO DE IMAGEN
num_img = 108
while (video.isOpened()):
    #SACAMOS EL FRAME DEL VIDEO
    ret, frame = video.read()
    if ret == False:
        break

    #CONVIERTE DE RGB A HSV PARA RESALTAR EL COLOR Y PODER USAR LOS PARAMETROS␣
 ↪DE LA "HSV COLOR TABLE"
    hsv=cv.cvtColor(frame, cv.COLOR_BGR2HSV)

    #MASCARA DE COLOR MARIGOLD AMARILLO Y ROJO
    #HSV H = COLOR BASE, S = SATURACION/BLANCOS, V = VALOR/PROFUNDIDAD/NEGRO
    YellowBajo1 = np.array([0,150,60])
    YellowAlto1 = np.array([32,255,255])
    redBajo2 = np.array([170,150,50])
    redAlto2 = np.array([180,255,255])
```

```
    maskRed1 = cv.inRange(hsv, YellowBajo1, YellowAlto1)
    maskRed2 = cv.inRange(hsv, redBajo2, redAlto2)
    mask1 = cv.add(maskRed1, maskRed2)


    #MASCARA DE COLOR DHALIA MORADA

    #WhiteBajo1 = np.array([0,0,0])
    #WhiteAlto1 = np.array([0,0,255])
    #PurpleBajo2 = np.array([135,0,0])
    #PurpleAlto2 = np.array([165,255,255])

    #maskRed1 = cv.inRange(hsv, WhiteBajo1, WhiteAlto1)
    #maskRed2 = cv.inRange(hsv, PurpleBajo2, PurpleAlto2)
    #mask1 = cv.add(maskRed1, maskRed2)


    #MASCARA DE COLOR ORQUIDEA BLANCO

    #WhiteBajo1 = np.array([0,0,168])
    #WhiteAlto1 = np.array([150,111,255])
    #mask1 = cv.inRange(hsv, WhiteBajo1, WhiteAlto1)


    #MASCARA DE COLOR ROSA ROJO

    #redBajo1 = np.array([0,50,50])
    #redAlto1 = np.array([10,255,255])
    #redBajo2 = np.array([170,50,50])
    #redAlto2 = np.array([180,255,255])

    #maskRed1 = cv.inRange(hsv, redBajo1, redAlto1)
    #maskRed2 = cv.inRange(hsv, redBajo2, redAlto2)
    #mask1 = cv.add(maskRed1, maskRed2)

    #FUNCION GUARDAR
    guardar(mask1 , frame, hsv, num_img)

    num_img += 1

video.release()
cv.waitKey(0)
cv.destroyAllWindows()
```

2.4 Girar y Transladar en Eje X

Este código se toma como un código totalmente distinto al anterior, ya que no tiene ningun tipo de conexión directa uno con el otro, la conexión es indirecta al únicamente jalar una imagen de las

generadas anteriormente para aplciarle las transformaciones ya mencionadas.

Como en el código anterior al esté ser un poco extenso y confuso decidi comentarlo sobre el mismo código para que nos sirve cada función, así como los parametros que le envian a cada una. Esto para favorecer la comprensión.

[ ]:
```
#ROTACION DE IMAGENES Y TRANSLACION PARA GENERAR MAYOR CANTIDAD DE IMAGENES EN␣
↪NUESTRO DATA SET
def Imagenes(ejex, ejey):

    #ESTABLECEMOS RUTA DE IMAGEN Y SE LEVANTA
    img1 = cv.imread('C:\\Users\\gilba\\Desktop\\IA\\DataSetFlores\\Marigold4.
↪jpg')

    ancho = img1.shape[0]
    alto = img1.shape[1]

    #GENERAMOS LA MATRIZ DE TRANSFORMACION DE LA IMAGEN, PARA TRANSLADARLA
    M= np.float32([[1,0,ejex], [0,1,ejey]])

    #WARPAFFINE TRANSLACION DE IMAGEN AFIN POR MEDIO DE OPERACIONES MATRICIALES
    #PROMER PARAMETRO LA IMAGEN A MODIFICARDE
    #SEGUNDO PARAMETRO LA MATRIZ TRANSFORMADA
    #TERCER PARAMETRO ANCHO Y ALTO DE IMAGEN
    img = cv.warpAffine(img1, M, (ancho, alto))

    #DATOS DE IMAGEN
    h,w = img.shape[:2]        #DEFINE LARGO Y ANCHO
    imgz = np.zeros((h*2, w*2), dtype = 'uint8') #CREA MATRIZ DE 0, IMAGEN A 8␣
↪BITS


    ##CONTADOR DE GUARDADO
    num_img = -180


    ## GIRAMOS LA IMAGEN
    ## GIRA LA LETRA - GIRO A MANECILLAS DEL RELOJ
    ##               + GIRO EN CONTRA DE MANECILLAS
    while(num_img != 181):

        #GIRAMOS LA IMAGEN CON LA FUNCION GETROTATION
        #PRIMER PARAMETRO NOS SIRVE PARA CENTRAR LA IMAGEN
        #SEGUNDO PARAMETRO GRADO DE GIRO
        #TERCER PARAMETRO FACTOR ESCALA
        mw = cv.getRotationMatrix2D( (h//2, w//2), num_img, 1 ) #CENTRAR FOTO,␣
↪ANGULO DE ROTACIÓN, FACTOR A ESCALAR
```

```
        #DEFINIMOS IMAGEN DE SALIDA, AFINAMOS CON WARPAFFINE
        #PRIMER PARAMETRO IMAGEN A MODIFICAR
        #SEGUNDO PARAMETRO LA MATRIZ TRANSFORMADA
        #TERCER PARAMETRO TAMANIO DE LA IMAGEN DE SALIDA
        imgz = cv.warpAffine(img,mw,(h,w)) # IMAGEN A MODIFICAR, MATRIZ␣
↪TRANSFORMADA, TAMAÑO DE IMAGEN DE SALIDA (50,50)

        ruta = 'C:
↪\\Users\\gilba\\Desktop\\IA\\DataSetFlores\\Marigold\\Marigold␣
↪(('+str(ejex)+'_'+str(ejey)+'_'+ str(num_img)+').jpg'
        cv.imwrite(ruta, imgz)
        num_img += 1
```

[ ]:
```
#CICLO, RECORREMOS EL ARREGLO DE RUTA GENERAL Y RUTA DE LA IMAGEN PARA LOGRAR␣
↪LA AUTOMATIZACION DEL PROCESO

#MATRIZ DE TRANSLACION DE IMAGEN DESDE (-1,-1) HASTA (1,1)
lados = [[1,-1,1,-1],[1,1,-1,-1]]

#INICIALIZACION UNICAMENTE PARA PODER CONTINUAR
x = 1
y = 1

#CICLO, RECORREMOS EL ARREGLO DE RUTA GENERAL Y RUTA DE LA IMAGEN PARA LOGRAR␣
↪LA AUTOMATIZACION DEL PROCESO
for i in range(4):
    for j in range(10):

        #CALCULAMOS LA TRANSLACION QUE LE DARA A LA IMAGEN POR CICLO
        #SE MULTIPLICA POR J PARA DARLE UNA TRANSLACION DE HASTA 10 PIXELES DE␣
↪IZQUIERDA A DERECHA, ARRIBA A ABAJO
        #PRIMERO EN LA TRANSLACION EJE X
        x = lados [1][i] * (j)
        Imagenes(x,y)

##FINALIZAR
print ("PROCESO FINALIZADO")
```

Dicho código nos genera un aproximado de 6,000 imagenes de salida con una sola imagen de entrada, puede variar dependiendo el ángulo de giro que desees darle, así como la translación.

De está manera se logro constituir el Data Set para que sirviera de entrada a nuestro Convolutional Neural Networks, la cual se explica como se llevo acabo en el siguiente apartado.

3 Convolutional Neural Networks Flores

Crearemos una Convolutional Neural Networks con Keras y Tensorflow para reconocer imagenes de distintas flores.

3.1 Importar Librerías

Importación de librerias necesarias para realizar el ejercicio

[2]:
```
import cv2 as cv
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
```

[3]:
```
import keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras.models import Sequential,Input,Model
#from keras.layers import Dense, Dropout, Flatten
#rom keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten,␣
 ↪Dropout, Dense, Conv2D
)
from tensorflow.python.keras.layers.advanced_activations import LeakyReLU
```

3.2 Cargar set de Imágenes

El proceso 'plt.imread(filepath)' cargará a memoria en un array las 22,000 imágenes, esté proceso puede llevar varios minutos, depende el procesamiento y la memoria RAM de tu computador.

[4]:
```
dirname = os.path.join(os.getcwd(),'C:
 ↪\\Users\\gilba\\Desktop\\IA\\DataSetFlores')
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="\r")
            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))
print("Imagenes en cada directorio", dircount)
print('suma Total de imagenes en subdirs:',sum(dircount))
```

```
leyendo imagenes de  C:\Users\gilba\Desktop\IA\DataSetFlores\
C:\Users\gilba\Desktop\IA\DataSetFlores\Dhalia 1
C:\Users\gilba\Desktop\IA\DataSetFlores\Marigold 7037
C:\Users\gilba\Desktop\IA\DataSetFlores\Orquidea 7180
C:\Users\gilba\Desktop\IA\DataSetFlores\Rosa 7044
Directorios leidos: 4
Imagenes en cada directorio [7038, 7180, 7044, 8700]
suma Total de imagenes en subdirs: 29962
```

4 Creamos las etiquetas

Las etiquetas se crean en labels, es decir cada flor tendra un valor entre 0 y 3. Esto se realiza para poder hacer uso de el algoritmo supervisado e indicar que cuando cargamos una imagen de Dhalia ya sabemos que corresponde a la etiqueta 0. Con estos parametros de entrada y salida, la red tendrá la capacidad de realizar el ajuste en los pesos de las neuronas.

[5]:
```
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
print("Cantidad etiquetas creadas: ",len(labels))
```

```
Cantidad etiquetas creadas:  29962
```

[6]:
```
letras=[]
indice=0
for directorio in directories:
    name = directorio.split(os.sep)
    print(indice , name[len(name)-1])
    letras.append(name[len(name)-1])
    indice=indice+1
```

```
0 Dhalia
1 Marigold
2 Orquidea
3 Rosa
```

Las etiqueas se convierten en Array con numpy.array

[7]:
```
y = np.array(labels)
X = np.array(images, dtype=np.uint8) #convierto de lista a numpy


# Find the unique numbers from the train labels
classes = np.unique(y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Total number of outputs :  4
Output classes :  [0 1 2 3]
```

4.1 Creamos Sets de Entrenamiento y Test

Debemos destacar que en la forma de los array (shape) veremos que son de 50 x 50 y por 3, esté se refiere a los canales de color (RGB) que tienen valores de 0 a 255.

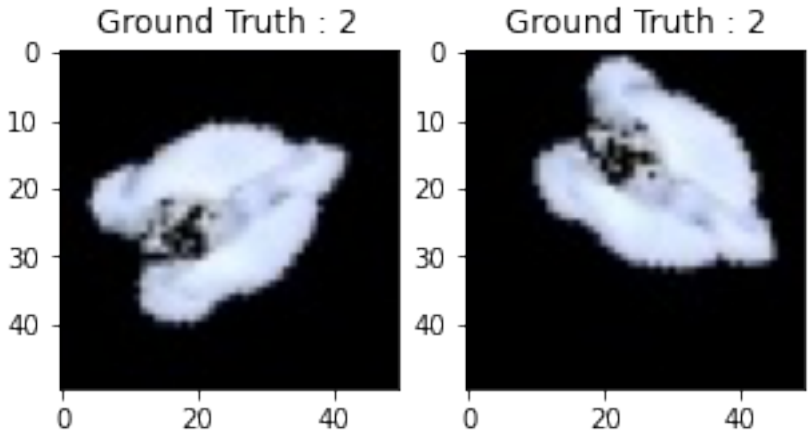
[8]:
```
train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.2)
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
Training data shape :  (23969, 50, 50, 3) (23969,)
Testing data shape :  (5993, 50, 50, 3) (5993,)
```

[9]:
```
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
```

```
plt.title("Ground Truth : {}".format(test_Y[0]))
```

[9]: `Text(0.5, 1.0, 'Ground Truth : 2')`

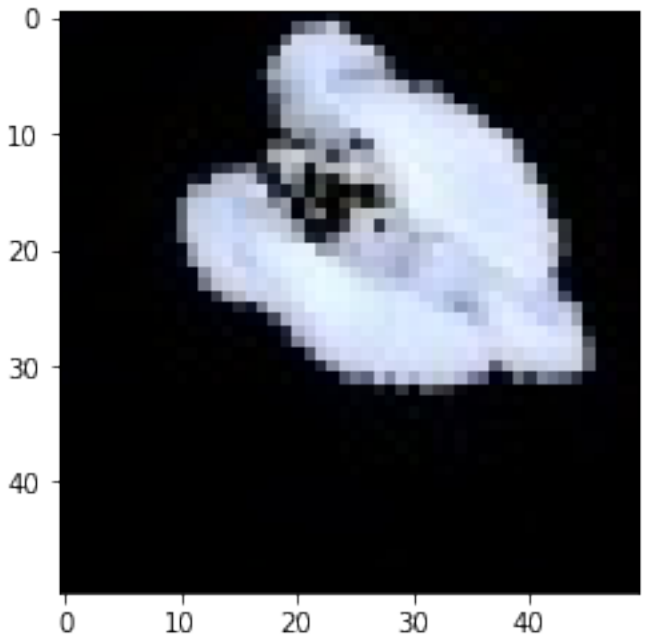


4.2 Preprocesamos las imagenes

Realizamos la normalización de valores para que tengan valor entre 0 y 1, por eso se realiza la división entre 255

[10]:
```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X/255.
test_X = test_X/255.
plt.imshow(test_X[0,:,:])
```

[10]: `<matplotlib.image.AxesImage at 0x2100abfce80>`



4.3 Hacemos el One-hot Encoding para la red

El "One-Hot Encoding" nos ayuda a realizar la clasificación para la red neuronal y corresponde con una capa de salida de la red de 10 neuronas, estl se realiza con "to categorical".

[11]:
```
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Display the change for category label using one-hot encoding
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 2
After conversion to one-hot: [0. 0. 1. 0.]
```

4.4 Creamos el Set de Entrenamiento y Validación

Subdividimos los datos en 80-20 para test y entrenamiento respectivamente. Y con el de training lo subdividimos en 80-20 para obtener un subconjunto de validación.

[12]:
```
#Mezclar todo y crear los grupos de entrenamiento y testing
train_X,valid_X,train_label,valid_label = train_test_split(train_X,␣
↪train_Y_one_hot, test_size=0.2, random_state=13)
```

[13]:
```
print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)
```

```
(19175, 50, 50, 3) (4794, 50, 50, 3) (19175, 4) (4794, 4)
```

4.5 Creamos el modelo de CNN

Aqui se crea la red, nos apoyamos de Keras para crear la CNN.

Las tres primeras constantes:

INIT_LR: Valor inicial de learning rate el 1e-3 corresponde al 0.001. Epochs: Cantidad de iteraciones completadas al conjunto de imagenes de entrenamiento. Batch_size: Cantidd de imágenes a procesar a la vez.

[14]:
```
#declaramos variables con los parámetros de configuración de la red
INIT_LR = 1e-3
epochs = 5
batch_size = 64
```

- Se crea una primera capa de neuronas convolucional de dos dimensiones en Cov2D, donde la entrada son las imagenes de 50x50x3
- Se aplican 32 filtros kernel 3x3
- La función de activacion para la neurona es “linear”
- MaxPooling(2,2) reduce la imagen al 50% es decir quedarán de 25x25
- Se termina con una capa de salida con 10 neurona con activacion SoftMax, que corresponde con el “Hot Encoding”

[15]:
```
letras_model = Sequential()
letras_model.add(Conv2D(32, kernel_size=(3,␣
↪3),activation='linear',padding='same',input_shape=(50,50,3)))#CAMBIAR TAMAÑO␣
↪DE IMAGEN
letras_model.add(LeakyReLU(alpha=0.1))
letras_model.add(MaxPooling2D((2, 2),padding='same'))
letras_model.add(Dropout(0.5))

letras_model.add(Flatten())
letras_model.add(Dense(32, activation='linear'))
letras_model.add(LeakyReLU(alpha=0.1))
letras_model.add(Dropout(0.5))
letras_model.add(Dense(nClasses, activation='softmax'))
```

[16]:
```
letras_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 32)        896

 module_wrapper (ModuleWrapp  (None, 50, 50, 32)       0
```

```
er)

max_pooling2d (MaxPooling2D  (None, 25, 25, 32)       0
)

dropout (Dropout)           (None, 25, 25, 32)        0

flatten (Flatten)           (None, 20000)             0

dense (Dense)               (None, 32)                640032

module_wrapper_1 (ModuleWra  (None, 32)               0
pper)

dropout_1 (Dropout)         (None, 32)                0

dense_1 (Dense)             (None, 4)                 132

=================================================================
Total params: 641,060
Trainable params: 641,060
Non-trainable params: 0
_________________________________________________________________
```

Compilamos y asignamos optimizador, Adagrad es nuestro optimizador para este caso

[17]:
```
letras_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=tf.
↪keras.optimizers.Adagrad(learning_rate=INIT_LR, decay=INIT_LR /␣
↪100),metrics=['accuracy'])
```

4.6 Entrenamos el modelo: Aprende a clasificar imágenes

Entenamiento de la CNN, con la linea "letras_train" se inicia el entrenamiento y validación de nuestra red. Se indica como letras porque use el mismo machote que se uso en el problema Abecedario.

Este paso puede tomar varios minutos, dependiendo de tu ordenador, cpu y memoria ram libre

[18]:
```
letras_train = letras_model.fit(train_X, train_label,␣
↪batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,␣
↪valid_label))
```

```
Epoch 1/5
300/300 [==============================] - 29s 93ms/step - loss: 0.7420 -
accuracy: 0.7635 - val_loss: 0.2288 - val_accuracy: 0.9973
Epoch 2/5
300/300 [==============================] - 28s 92ms/step - loss: 0.2438 -
accuracy: 0.9628 - val_loss: 0.0743 - val_accuracy: 0.9973
Epoch 3/5
300/300 [==============================] - 28s 94ms/step - loss: 0.1456 -
accuracy: 0.9779 - val_loss: 0.0398 - val_accuracy: 0.9973
```

```
Epoch 4/5
300/300 [==============================] - 29s 96ms/step - loss: 0.1021 -
accuracy: 0.9846 - val_loss: 0.0258 - val_accuracy: 0.9973
Epoch 5/5
300/300 [==============================] - 29s 96ms/step - loss: 0.0824 -
accuracy: 0.9866 - val_loss: 0.0199 - val_accuracy: 0.9973
```

Guardamos la red, para reutilizarla en el futuro, sin tener que volver a entrenar

Vemos que tras 5 iteraciones completas al set de entrenamiento obtenemos un valor de presición de 98.66% y en el set de validación alcanza un 99.73%

[19]:
```
letras_model.save("C:
↪\\Users\\gilba\\Desktop\\IA\\ProyectoFinal\\Flores\\flores99.h5py")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
leaky_re_lu_layer_call_fn, leaky_re_lu_layer_call_and_return_conditional_losses,
leaky_re_lu_1_layer_call_fn,
leaky_re_lu_1_layer_call_and_return_conditional_losses while saving (showing 5
of 5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to:
C:\Users\gilba\Desktop\IA\ProyectoFinal\Flores\floresAux.h5py\assets

INFO:tensorflow:Assets written to:
C:\Users\gilba\Desktop\IA\ProyectoFinal\Flores\floresAux.h5py\assets
```

4.7 Evaluamos la red

Ya con la red entrenada, la ponemos a prueba con el set de imágenes para Test que separamos al principio, cabe resaltar que son muestras que nunca han sido "vistas" por la máquina.

[20]:
```
test_eval = letras_model.evaluate(test_X, test_Y_one_hot, verbose=1)
```

```
188/188 [==============================] - 3s 15ms/step - loss: 0.0195 -
accuracy: 0.9968
```

[21]:
```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.019467871636152267
Test accuracy: 0.996829628944397
```

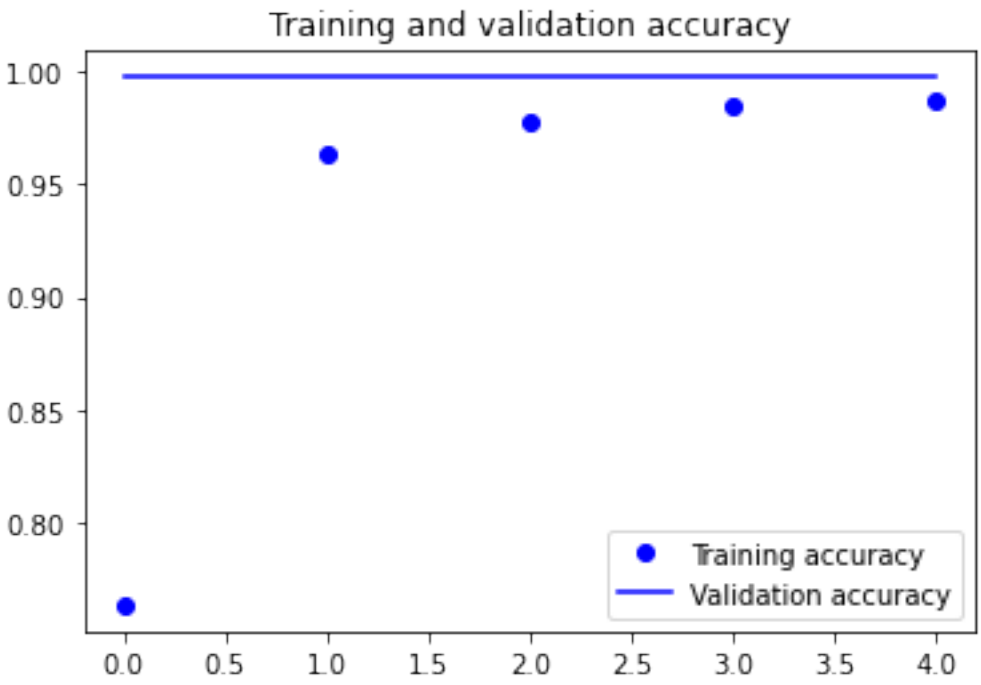
Vemor que el conjunto de test alcanza una presición del 99.68%

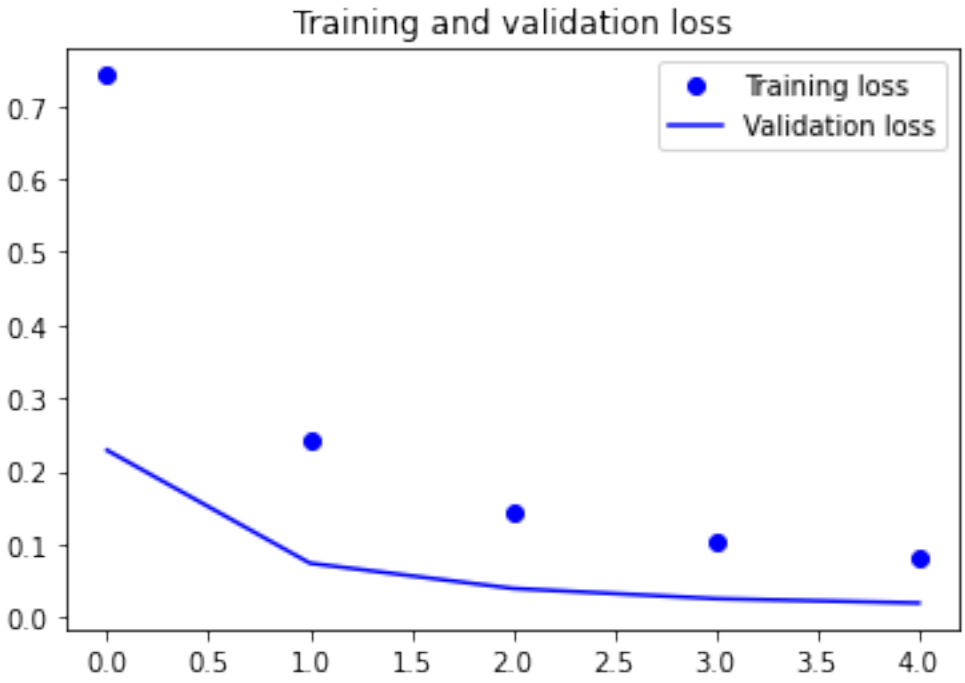
[22]:
```
letras_train.history
```

[22]:
```
{'loss': [0.7420064210891724,
  0.24384678900241852,
  0.14559751749038696,
  0.10206397622823715,
  0.08238697797060013],
```

```
 'accuracy': [0.7635462880134583,
  0.9628161787986755,
  0.977940022945404,
  0.9846153855323792,
  0.9866492748260498],
 'val_loss': [0.22879138588905334,
  0.07426217198371887,
  0.03975499048829079,
  0.025792190805077553,
  0.019894812256097794],
 'val_accuracy': [0.9972882866859436,
  0.9972882866859436,
  0.9972882866859436,
  0.9972882866859436,
  0.9972882866859436]}
```

[23]:
```
accuracy = letras_train.history['accuracy']
val_accuracy = letras_train.history['val_accuracy']
loss = letras_train.history['loss']
val_loss = letras_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





[24]:
```
predicted_classes2 = letras_model.predict(test_X)
```

```
188/188 [==============================] - 3s 14ms/step
```

[25]:
```
predicted_classes=[]
for predicted_letras in predicted_classes2:
    predicted_classes.append(predicted_letras.tolist().
 ↪index(max(predicted_letras)))
predicted_classes=np.array(predicted_classes)
```

[26]:
```
predicted_classes.shape, test_Y.shape
```

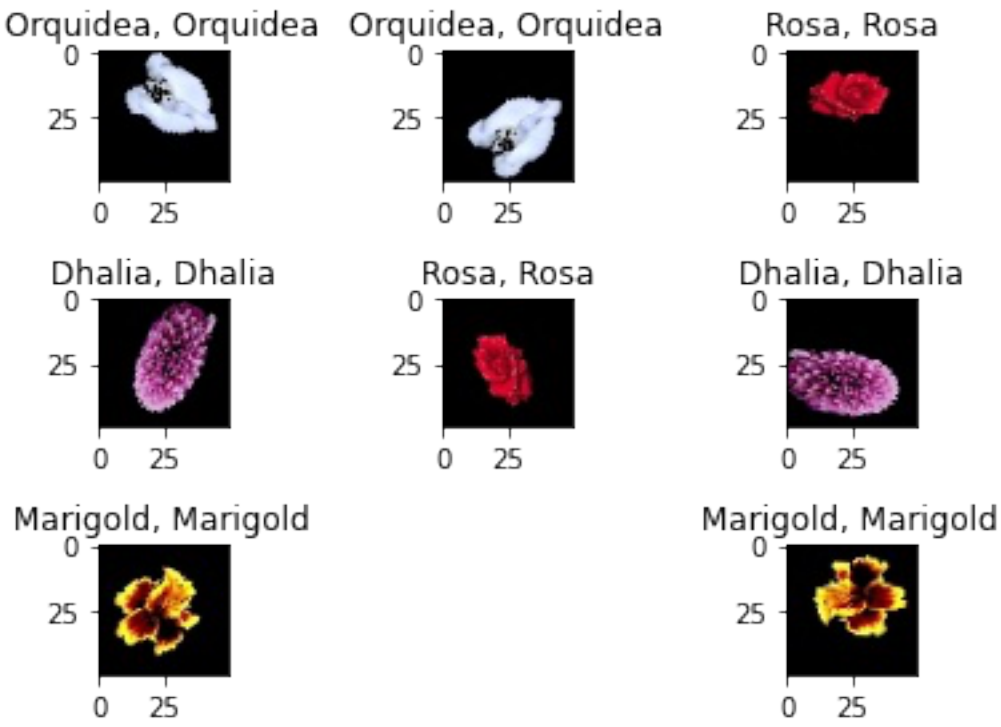
[26]:
```
((5993,), (5993,))
```

4.8 Aprendamos de los errores: Qué mejorar

Realizamos un analisis más profundo para darnos cuenta de que mejorar y como realizarlo, ya sea mejorando nuestro data set o dandole mas ejemplos a la neurona para que así logre un mejor aprendizaje del objeto, panorama, acción, etc. a detectar.

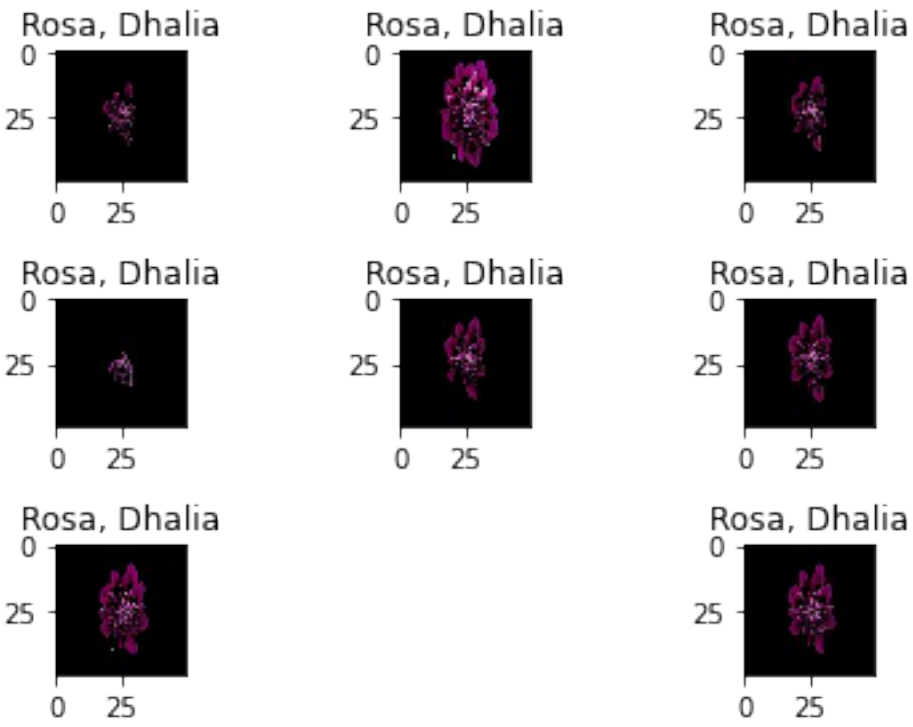
[27]:
```
correct = np.where(predicted_classes==test_Y)[0]
print("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[0:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[correct].reshape(50,50,3), cmap='gray',␣
 ↪interpolation='none')#CAMBIAR TAMAÑO DE IMAGEN
    plt.title("{}, {}".format(letras[predicted_classes[correct]],
                                                    letras[test_Y[correct]]))

    plt.tight_layout()
```

```
Found 5974 correct labels
```



[28]:
```
incorrect = np.where(predicted_classes!=test_Y)[0]
print("Found %d incorrect labels" % len(incorrect))
for i, incorrect in enumerate(incorrect[0:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[incorrect].reshape(50,50,3), cmap='gray',␣
 ↪interpolation='none')
    plt.title("{}, {}".format(letras[predicted_classes[incorrect]],
                                                    letras[test_Y[incorrect]]))
    plt.tight_layout()
```

```
Found 19 incorrect labels
```



[29]:
```
target_names = ["Class {}".format(i) for i in range(nClasses)]
print(classification_report(test_Y, predicted_classes,␣
 ↪target_names=target_names))
```

```
              precision    recall  f1-score   support

     Class 0       1.00      0.99      0.99      1415
     Class 1       1.00      1.00      1.00      1459
     Class 2       1.00      1.00      1.00      1409
     Class 3       0.99      1.00      0.99      1710

    accuracy                           1.00      5993
   macro avg       1.00      1.00      1.00      5993
weighted avg       1.00      1.00      1.00      5993
```

Podemos observar que en la presición final de un analisis más profundo tenemos un acercamiento al reconocimiento de Dhalias del 100%, de Marigold del 100%, de Orquideas del 100% y por ultimo de Rosas del 98%.

Dandonos un promedio de presición del 100%

5 Aplicación del Módelo, Validación

Para la aplicación (Validación) del Modelo, haremos uso del archivo guardado "Flores99.hp5py".

El código no es extenso, ni dificil de comprender pero decidi comentarlo en el mismo para mayor conprensión y poder darme a explicar más conforme a las funciones y parametros utilizados.

5.1 Importación de Librerías

[30]:
```
import numpy as np
from keras import models
import cv2 as cv
```

5.2 Cargamos el Módelo

[34]:
```
model = models.load_model("C:
↪\\Users\\gilba\\Desktop\\IA\\ia\\ProyectoFinal\\Flores\\flores99.h5py")
```

5.3 Aseguramos la carga del Módelo

La impresión que nos genere el "summary" debe ser identica al que generamos al crear el Modelo en la Red Convolucional

[35]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 32)        896

 module_wrapper (ModuleWrapp  (None, 50, 50, 32)       0
 er)

 max_pooling2d (MaxPooling2D  (None, 25, 25, 32)       0
 )

 dropout (Dropout)           (None, 25, 25, 32)        0

 flatten (Flatten)           (None, 20000)             0

 dense (Dense)               (None, 32)                640032

 module_wrapper_1 (ModuleWra  (None, 32)               0
 pper)

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 4)                 132
```

```
=================================================================
Total params: 641,060
Trainable params: 641,060
Non-trainable params: 0
_________________________________________________________________
```

5.4 Arreglo de Clases

Me ayuda a empatar la predicción con un entorno gráfico

[36]:
```
#ARREGLO DE CLASES PARA CUANDO REALICEMOS LA PREDICCION COINCIDAN
clases_flores = ['Dhalia', 'Marigold', 'Orquidea', 'Rosa']

#DECALRAMOS UN CONTADOR PARA NO DESBORDAR EL CICLO DE BUSQUEDA DE LA␣
↪PREDICCION,
#SOBRE TODO PARA QUE SE VEA MAS CLARO, VISUAL
contador = len(clases_flores)
```

5.5 Tratamiento de Imagen, Np Array

[37]:
```
def tratarImg(im):
    #SE PODRIAN AGREGAR FILTRO DE COLOR Y REDIMENSION PARA TRATAR LA IMAGEN␣
↪ANTES DE
    #CONVERTIRLA A UN ARRAY Y REALIZAR LA PREDICCION, ESTO NOS DARIA MAYOR␣
↪PRESICION

    #SE CORRIJE LA GAMA DE LA IMAGEN DE BGR A RGB :) SIN ESTA LINEA LA␣
↪PREDICCION NO ESTAN EN LA MISMA GAMA DE COLOR
    im = cv.cvtColor(im, cv.COLOR_BGR2RGB)

    #SE CREA EL NUMPY ARRAY
    img_array = np.array(im)

    #EXPANDE LA MATRIZ ARRAY PARA EVITAR EL ERROR NONE,50,50,3
    #PRIMER PARAMETRO MATRIZ A EXPANDIR
    #SEGUNDO PARAMETRO POSICION INICIAL
    img_array = np.expand_dims(img_array, axis=0)

    #RETORNAMOS EL NUMPY ARRAY DE LA IMAGEN
    return img_array
```

5.6 Realizamos la Predicción

[40]:
```
#LEEMOS LA IMAGEN
im = cv.imread("C:
↪\\Users\\gilba\\Desktop\\IA\\ia\\ProyectoFinal\\Flores\\ImagenesPruebas\\Rosa2.
↪jpg")
```

```
#IM2 LA USAREMOS PARA MOSTRARLA UNICAMENTE
imagen2 = cv.imread("C:
 ↪\\Users\\gilba\\Desktop\\IA\\ia\\ProyectoFinal\\Flores\\ImagenesPruebas\\Rosa2.
 ↪jpg")

#RESIZE PARA VER EL RESULTADO MAS AMPLIO, FUNCIONALIDAD ESTÉTICA
imagen2 = cv.resize(imagen2,(600,600))

#SE ENVIA IM AL METODO PARA TRATARLA Y CONVERTIRLA EN ARRAY
img_array = tratarImg(im)

#SE REALIZA LA PREDICCION CON EL MODELO Y LA IMAGEN ARRAY
prediction = model.predict(img_array)

#SE IMPRIME LA PREDICCION
print (prediction)

#RECORREMOS EL ARREGLO Y GUARDAMOS LA PREDICCION PARA AGREGARLA A LA IMAGEN␣
 ↪FINAL
for i in range(contador):
    #SI EL VALOR DE LA CASILLA [0][i] ES IGUAL A 1 SACAMOS EL NOMBRE DE ESA␣
 ↪MISMA CELDA EN EL ARRAY CLASES
    if prediction[0][i].all() == 1 :
        #GUARDA LA PREDICCION PARA MOSTRAR DICHO TEXTO EN LA IMAGEN
        mostrar="Predice: "+clases_flores[i]
        #ROMPE EL CICLO SI ENCONTRO UN 1
        break

#PUT TEXT AGREGA EL TEXTO EN LA IMAGEN
#PRIMER PARAMETRO IMAGEN A MOSTRAR
#SEGUNDO PARAMETRO TEXTO A AGREGAR
#TERCER PARAMETRO POSICION
#CUARTO PARAMETRO TIPO DE LETRA
#QUINTO PARAMETRO TAMANIO DE LA LETRA
#SEXTO PARAMETRO COLOR
#SEPTIMO PARAMETRO TRAZO DE LA LETRA, NUMERO DE PIXELES
cv.putText(imagen2, mostrar,(0,50), cv.FONT_HERSHEY_SIMPLEX, 1,(50,255,0),2)

#MUESTRA IMAGEN
cv.imshow("imagen", imagen2)
cv.waitKey(0)
cv.destroyAllWindows()
```

```
1/1 [==============================] - 0s 114ms/step
[[0. 0. 0. 1.]]
```

No se muestra el entorno grafico en la predicción pero usted puede ejecutar todo el código en esté

reporte y en esta ultima instancia le generará un entorno gráfico donde muestra la imágen a predecir y la predicción dada por la CNN.