PROYECTO 3 – REDES CONVOLUCIONALES

AUTORES

CRISTIAN ALBERTO AGUDELO MARQUEZ

JUAN JOSE GOMEZ MEJIA

PROFESOR

SEBASTIAN GUZMAN OBANDO

UNIVERSIDAD DE ANTIOQUIA

DEPARTAMENTO DE INGENIERÍA

PROCESAMIENTO DIGITAL DE IMAGENES

Mayo 29, 2023

Inicialmente, a partir del banco de imágenes trabajado en el **PROYECTO 2 - FILTROS Y DESCRIPTORES**, teníamos frutas, clasificadas en:

'FRESAS', 'MANGOS', 'MANZANAS' y 'PERAS'

Ahora, haremos una extracción de pixeles, para representar cada imagen como una matriz de **64x64** en sus 3 canales **RGB**, esto con el fin de entrenar las imágenes con una **RED CONVOLUCIONAL**.

```python
class ImageDataProcessor:
    def __init__(self, base_dir):
        self.base_dir = base_dir
        self.classes = os.listdir(base_dir)

    def load_image(self, image_path):
        image = cv.imread(image_path)

        if image is None or image.shape == (0, 0):
            raise ValueError(f'Invalid image: {image_path}')

        # Normalización de las imagenes
        image_gaussian = cv.GaussianBlur(image, (3, 3), 0)
        image_resized = cv.resize(image_gaussian, (64, 64), interpolation=cv.INTER_AREA)
        print(image_resized.shape)

        return image_resized

    def stack_images(self):
        images_stack = []
        labels = []
        sorted_classes = sorted(self.classes)

        for i, class_name in enumerate(sorted_classes):
            class_dir = os.path.join(self.base_dir, class_name)
            print(f'Processing class: {i} {class_name}')

            for image_name in os.listdir(class_dir):
                try:
                    image = self.load_image(os.path.join(class_dir, image_name))
                    images_stack.append(image)
                    labels.append(i)
                except ValueError as e:
                    print(e)

        return images_stack, labels

    def save_data(self, data, labels):
        output_dir = os.path.join(ruta + '/BASE_DATOS_CNN')
        os.makedirs(output_dir, exist_ok=True)
        np.save(os.path.join(output_dir, 'vector_x'), data)
        np.save(os.path.join(output_dir, 'vector_y'), labels)

    def process_images(self):
        descriptors, labels = self.stack_images()
        self.save_data(descriptors, labels)
        print('Data saved successfully!')
```

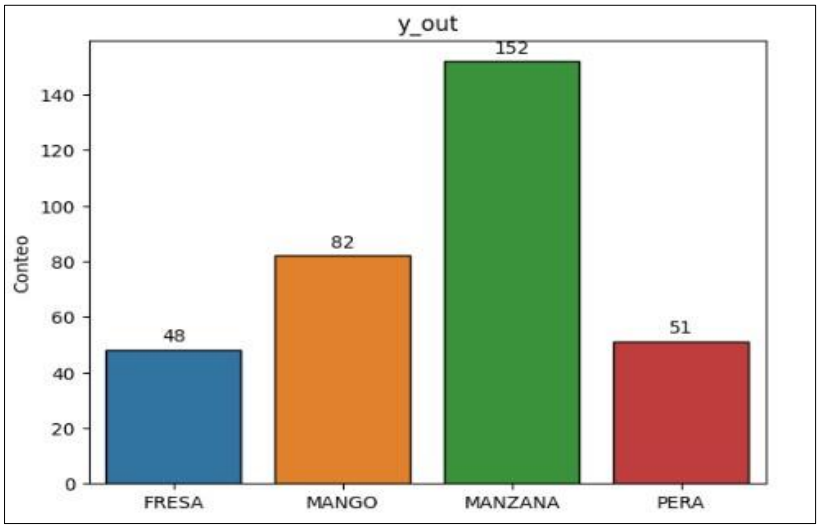
```python
if __name__ == '__main__':
    base_dir = '/content/BASE_DATOS'
    data_processor = ImageDataProcessor(base_dir)
    data_processor.process_images()

```

Obtenemos matrices para cada imagen con las siguientes dimensiones:

```
# Imprimir información de las dimensiones
forma = images.shape
print("Número de imágenes:", forma[0])
print("Dimensiones de cada imagen:", forma[1], "x", forma[2])
print("Número de canales:", forma[3])

```

```
Número de imágenes: 333
Dimensiones de cada imagen: 64 x 64
Número de canales: 3
```

Recordando, si analizamos la variable objetivo, tenemos un poco de **desproporción** entre las clases, hay bastantes manzanas respecto a fresas.



Finalmente, entrenaríamos **la red convolucional** realizando una partición de los datos, teniendo **70%** para entrenar el modelo y **30%** para hacer validaciones.

```
# Normalizamos
images_norm = images/np.linalg.norm(255.0)

#Conversión en array
images_norm = np.array(images_norm)
labels = np.array(labels)

#Separación de la base de datos
# X_train contiene los datos del vector de vectores, del mismo tamaño que y_train
# Y_train contiene las clases a las que pertenecen (Key)
x_train, x_test, y_train, y_test = train_test_split(images_norm, labels, test_size=0.3, random_state=42)

print("Tamaño de X para el entrenamiento", x_train.shape);
print("Tamaño de Y para el entrenamiento", y_train.shape);
print("Tamaño de X para prueba ", x_test.shape);
print("Tamaño de Y para prueba ", y_test.shape);

```

```
Tamaño de X para el entrenamiento (233, 64, 64, 3)
Tamaño de Y para el entrenamiento (233,)
Tamaño de X para prueba  (100, 64, 64, 3)
Tamaño de Y para prueba  (100,)
```

Redes Convolucionales 1 - usando una arquitectura Personalizada

Utiliza una arquitectura personalizada definida manualmente utilizando capas de Keras. La red convolucional consta de capas convolucionales, capas de MaxPooling, capas de BactchNormalization, una capa Flatten, capas densas y una capa de salida.

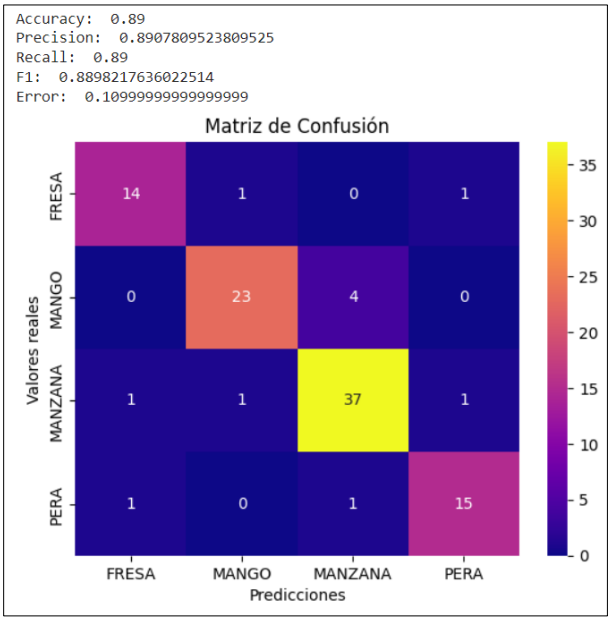
```
inputs = tf.keras.Input(shape=(64, 64, 3), name="input_1")

layers = tf.keras.layers.Conv2D(90, (3, 3), activation="relu")(inputs)
layers = tf.keras.layers.BatchNormalization()(layers)
layers = tf.keras.layers.MaxPool2D((2, 2))(layers)

# Agregar más capas convolucionales y de pooling
layers = tf.keras.layers.Conv2D(60, (3, 3), activation="relu")(layers)
layers = tf.keras.layers.BatchNormalization()(layers)
layers = tf.keras.layers.MaxPool2D((2, 2))(layers)

layers = tf.keras.layers.Conv2D(30, (3, 3), activation="relu")(layers)
layers = tf.keras.layers.BatchNormalization()(layers)
layers = tf.keras.layers.MaxPool2D((2, 2))(layers)

# Capas densas adicionales antes de la salida
layers = tf.keras.layers.BatchNormalization()(layers)
layers = tf.keras.layers.Flatten()(layers)
layers = tf.keras.layers.Dense(64, activation="relu")(layers)
layers = tf.keras.layers.Dense(32, activation="relu")(layers)
layers = tf.keras.layers.Dense(16, activation="relu")(layers)
layers = tf.keras.layers.Dropout(0.2)(layers)

##capa de salida
predictions = tf.keras.layers.Dense(4, activation="softmax", name="output_1")(layers)
```

Finalmente, se **entrena** y **valida** el modelo, utilizando **100** épocas y una selección de lote de **32** muestras para cada época.

```
model.fit(x_train, y_train_categorical, epochs=100, batch_size=32, validation_data=(x_test, y_test_categorical))
```

Observando los resultados, las **métricas de desempeño** muestran resultados aceptables, tanto en el **accuracy**, como en la **presicion** y se evidencia en la **matriz de confusión** una cantidad de aciertos apreciable para cada clasificación, a pesar de haber una desproporción de clases y un bajo número de muestras.

```
Accuracy:  0.89
Precision:  0.8907809523809525
Recall:  0.89
F1:  0.8898217636022514
Error:  0.10999999999999999
```



Redes Convolucionales 2 - usando una arquitectura ResNet152

Utiliza una arquitectura predefinida de **ResNet152** utilizando la API de aplicaciones de Keras. La red utiliza una capa de Lambda para redimensionar las imágenes de entrada, seguida de la arquitectura de ResNet152, capas de Flatten y capas densas con dropout.

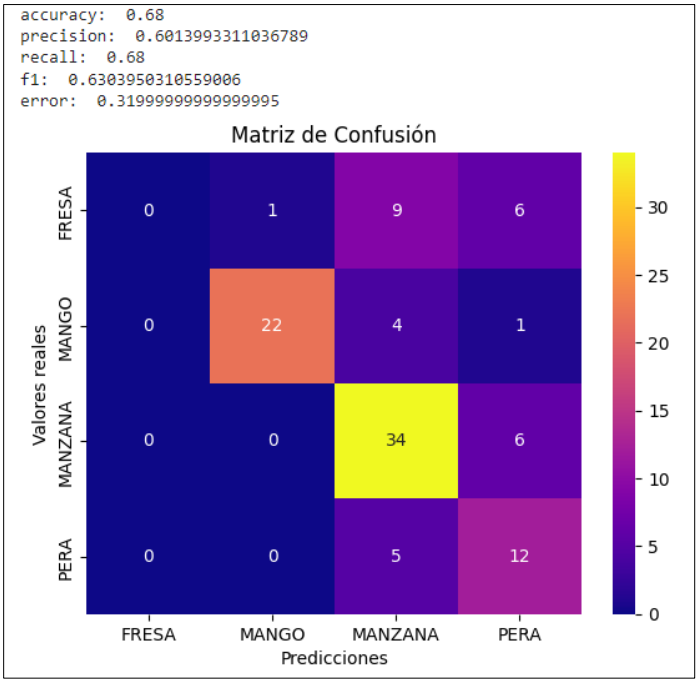
```
# Crear el modelo de la red convolucional basado en ResNet152
res_model = k.applications.ResNet152(include_top=False, weights="imagenet", input_shape=(64, 64, 3))
model.summary()

model = k.models.Sequential()
model.add(k.layers.Lambda(lambda image: tf.image.resize(image, (64, 64))))
model.add(res_model) ##  la instancia de la red ResNet152 pre-entrenada cargada desde k.applications.ResNet152.
model.add(k.layers.BatchNormalization())
model.add(k.layers.Flatten())
model.add(k.layers.Dense(64, activation='relu'))
model.add(k.layers.Dense(32, activation='relu'))
model.add(k.layers.Dense(16, activation='relu'))
model.add(k.layers.Dropout(0.2))

# Capa de salida
model.add(k.layers.Dense(4, activation='softmax'))
```

Finalmente, se **entrena** y **valida** el modelo, utilizando **100** épocas y una selección de lote de **32** muestras para cada época.

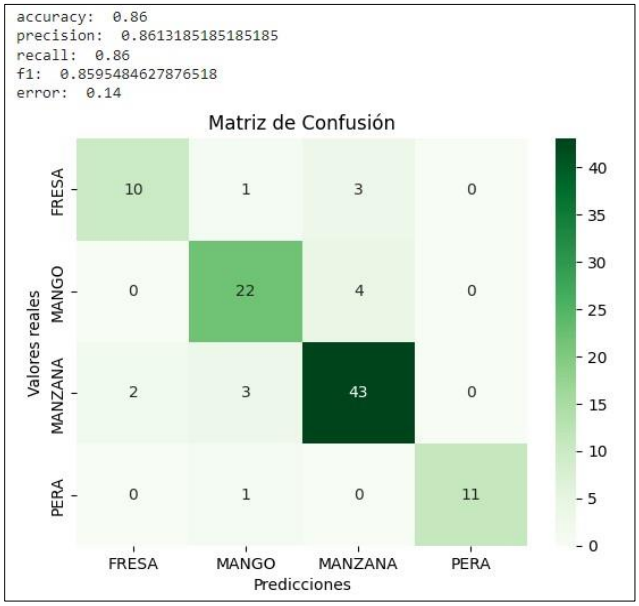
```
model.fit(x_train, y_train_categorical, epochs=100, batch_size=32, validation_data=(x_test, y_test_categorical))
```

Observando los resultados, las **métricas de desempeño** muestran resultados un poco peores en comparación con la red convolucional anterior, tanto en el **accuracy**, como en la **precisión.** **Además el consto computacional fue más alto.**

```
accuracy:  0.68
precision:  0.6013993311036789
recall:  0.68
f1:  0.6303950310559006
error:  0.31999999999999995
```



Ahora, recordando los resultados obtenidos en **PROYECTO 2 - FILTROS Y DESCRIPTORES**:



Podemos concluir que utilizando tanto extracción de características de las imágenes mediante descriptores **HOG** y entrenando en una red neuronal con capas densas; así como entrenar las imágenes con **redes convolucionales**, se obtienen aproximadamente los mismos resultados, donde se observó que ambos enfoques ofrecen resultados similares en términos de métricas de rendimiento, aunque la red neuronal convolucional mostró una ligera ventaja sobre el enfoque basado en descriptores HOG.

Esto indica que la red neuronal convolucional es capaz de extraer automáticamente características relevantes de las imágenes sin la necesidad de utilizar descriptores de bajo nivel.
Además, la ventaja de la red neuronal convolucional puede atribuirse a su capacidad de aprendizaje jerárquico, donde las capas convolucionales aprenden características de nivel inferior y las capas densas aprenden características de nivel superior. Sin embargo, hay que considerar que el coste computaciones es más elevado al utilizar redes convolucionales.

Referencias:

- "How Good Is Your Machine Learning Algorithm?" https://www.mydatamodels.com/learn/how-good-is-your-machine-learning-algorithm/

- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. https://arxiv.org/abs/1409.1556

- "Convolutional Neural Networks" - Stanford University https://cs231n.github.io/convolutional-networks/

- "Deep Learning for Computer Vision" - Adrian Rosebrock, PyImageSearch https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/