Visión por Computador (VxC)

Grado en Informática Industrial y Robótica (GIIROB)

Práctica 8:

Redes Neuronales

Contenido:

1. Introducción
2. Primeros pasos con *Keras*
3. Los conjuntos de entrenamiento y test
4. Creación del modelo CNN
5. Entrenamiento del modelo CNN
6. Evaluar el modelo CNN
7. Utilización de una red CNN estándar: VGG16
8. Sobre el trabajo2D de la asignatura VxC

Autor:

José Miguel Valiente González

Bibliografía:

- J. Howse & J. Minichino. “Learning OpenCV 4 Computer Vision with Python 3” (third Ed.) Packt Publishing Ltd. 2020.
- Nikhil Singh & Paras Ahuja, Fundamentals of Deep Learnoing and Computer Vision. BPB Publications. 2020.
- I. Zafar, C. Tzanidou, R. Burtton, N. Patel and L. Araujo. Hands-On Convolutional Neural Networks with TensorFlow. Packt ed. 2018.
- Sandigan Dei, “Python Image Processing. Cookbook”. Packt Publishing Ltd. 2020.
- Bharat Sikka. Elements of Deep Learnibng for Computer Vision. BPB Publishing, 2021.
- Official documentation for Python 3.10 https://docs.python.org/es/3/tutorial/index.html
- Official documentation for OpenCV 3 https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

En esta práctica vamos a estudiar las funciones de diversas bibliotecas (**TensorFlow**, **Keras**) para el manejo de Redes Neuronales Convolucionales (**CNN**) del ámbito de Deep Learning. Junto con OpenCV, estas bibliotecas permitirán las operaciones de clasificación y reconocimiento de formas en general. Estas redes CNN serán el último elemento del trabajo de visión 2D sobre la identificación de las cartas de póker. El código desarrollado en esta práctica servirá para el citado trabajo y no será necesario realizar un documento de respuestas de esta práctica.

Objetivos

- Describir las funciones que tiene TensorFlow y Keras para crear, entrenar y utilizar redes CNN en operaciones de clasificación y reconocimiento de formas.
- Preparación del código para el trabajo 2D.

Material

- Archivos de imagen de las cartas de póker etiquetadas, '*trainCards.npz*' y '*testCards.npz*', generados en las prácticas anteriores.

Se asume que se dispone de una carpeta '*/trabajo2D/*' con los scripts de las prácticas anteriores, así como una carpeta '*/images/*' con el *dataset* de las cartas de póker. En esta carpeta iremos creando nuevos scripts y todo el conjunto constituirá el trabajo 2D de la asignatura. Sólo habrá que completarlo con un documento *pdf* de explicación del trabajo y los resultados.

1. Introducción

TensorFlow es una biblioteca de código abierto, desarrollada por Google, para su utilización en el campo de las redes neuronales y el aprendizaje automático. TensorFlow proporciona un conjunto completo de herramientas y funcionalidades que facilitan el desarrollo, entrenamiento y despliegue de redes neuronales, convirtiéndolo en una de las bibliotecas más populares en el campo del aprendizaje automático y la inteligencia artificial. En el contexto de las redes neuronales, TensorFlow ofrece las siguientes características y funcionalidades clave:

1. Creación y entrenamiento de redes neuronales: TensorFlow permite construir y entrenar redes neuronales artificiales para detectar patrones y razonamientos utilizados por los humanos. Facilita la implementación de modelos de deep learning y aprendizaje automático para los desarrolladores.
2. Procesamiento de datos: La biblioteca permite trabajar con grandes cantidades de datos y realizar cálculos numéricos complejos necesarios para el entrenamiento de redes neuronales.
3. Grafos computacionales: TensorFlow se basa en el concepto de grafos computacionales, donde cada nodo representa una operación matemática y las conexiones entre nodos son tensores (matrices multidimensionales).
4. Versatilidad: Puede ejecutarse en diversas plataformas, incluyendo CPUs, GPUs y TPUs (Unidades de Procesamiento Tensorial), lo que permite una gran flexibilidad en el entrenamiento y despliegue de modelos.
5. Aplicaciones: TensorFlow se utiliza en tareas como clasificación de imágenes, reconocimiento facial, procesamiento del lenguaje natural y muchas otras aplicaciones de inteligencia artificial.

Por su parte, **Keras** es una API de alto nivel integrada en TensorFlow que simplifica la creación y entrenamiento de modelos de aprendizaje profundo. Algunas características clave de Keras en TensorFlow son:

1. Interfaz amigable: Keras proporciona una interfaz simple y consistente, optimizada para casos de uso comunes en el desarrollo de redes neuronales.
2. Modularidad: Permite construir modelos conectando bloques configurables, ofreciendo flexibilidad en el diseño de arquitecturas de redes neuronales.
3. Extensibilidad: Facilita la creación de capas personalizadas, métricas y funciones de pérdida, lo que es útil para la investigación y el desarrollo de modelos avanzados.
4. Integración con TensorFlow: Desde TensorFlow 2.0, Keras (tf.keras) se ha convertido en la API oficial de alto nivel de TensorFlow.
5. Versatilidad: Keras en TensorFlow soporta diversos tipos de redes neuronales, incluyendo redes convolucionales y recurrentes.
6. Ejecución eficiente: Al estar integrado en TensorFlow, Keras puede aprovechar las capacidades de cálculo distribuido y la optimización de hardware de TensorFlow.
7. Prototipado rápido: Permite crear y experimentar con modelos de forma rápida y sencilla, lo que es especialmente útil en las fases iniciales de desarrollo.

Keras en **TensorFlow** combina la facilidad de uso de Keras con la potencia y flexibilidad de TensorFlow, proporcionando una herramienta robusta para el desarrollo de modelos de aprendizaje profundo tanto para principiantes como para expertos en el campo. En esta práctica utilizaremos Keras, pues la API de alto nivel de Keras es más sencilla, rápida y manejable que la de TensorFlow, lo que la hace más apropiada para pequeños proyectos y para educación. Se puede encontrar una explicación y documentación de Keras 3.0 API en:

https://keras.io/about/

Además de la anteriores, existen otras bibliotecas o *frameworks* para el manejo de CNNs, como pueden ser: PyTorch, Scikit-learn, Caffe, Sonnet, OpenNN, etc...

En cuanto a la práctica actual, ya se dispone de diversas funciones que nos han permitido segmentar las cartas, extraer los motivos de cada carta y definir, para cada motivo, un conjunto de características en forma de vector. En este proceso, hemos podido guardar la imagen de cada carta, en niveles de gris y color, así como el rectángulo que inscribe cada motivo. Ahora nos proponemos tomar la imagen de cada motivo en color y utilizar una red CNN para clasificarlo.

2. Primeros pasos con Keras

Los elementos fundamentales de Keras son las capas (*layers*) y los modelos (*model*). Un modelo es una secuencia lineal de capas, donde cada capa puede ser de tipo convolucional, dropout, dense, etc… La entrada es una imagen, que viene determinada por la primera capa. En los siguientes fragmentos vemos cómo comenzar con el modelo secuencial.

```
from tensorflow import  keras
……
import cv2
from keras import layers

model = keras.Sequential()

model.add(layers.Dense(units=64, activation='relku'))
model.add(layers.Dense(units=10, activation='softmax'))

# Para configurar el modelo
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics='accuracy')
```

```
# o bien utilizar una forma que permite más control de los parámetros
model.compile(loss=keras.losses.categorical_crossentropy,
                    optimizer=keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, Nesterov=True)
# Para entrenar el modelo
model.fit(x_train, y_train, epochs=5, batch_size=32)

# Para evaluar el modelo con el conjunto de test
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

# o bien, para predecir una muestra con el modelo
Classes = model.predict(x_test, batch_size=128)
```

Además del modelo básico secuencial, Keras dispone de una API funcional, que ofrece mayor flexibilidad y control para construir modelos más complejos y versátiles.

Las principales diferencias entre el modelo secuencial y la API funcional de Keras son:

1. Flexibilidad:
   - El modelo secuencial es más simple pero limitado en cuanto a la topología del modelo. Es adecuado para redes neuronales lineales donde cada capa tiene exactamente una entrada y una salida.
   - La API funcional ofrece mayor flexibilidad, permitiendo crear modelos más complejos como arquitecturas con múltiples entradas/salidas, grafos acíclicos dirigidos (DAG) y modelos de capas compartidas.
2. Construcción del modelo:
   - En el modelo secuencial, las capas se añaden secuencialmente una tras otra usando el método add().
   - En la API funcional, se definen primero las capas y luego se especifica cómo se conectan entre sí, creando un grafo de capas.
3. Complejidad:
   - El modelo secuencial es más intuitivo y fácil de usar, especialmente para principiantes.
   - La API funcional requiere un mayor entendimiento de la arquitectura del modelo, pero permite crear estructuras más sofisticadas.
4. Casos de uso:
   - El modelo secuencial es ideal para redes simples y prototipado rápido.
   - La API funcional es preferible para modelos complejos que no pueden representarse como una simple pila de capas.
5. Definición de entradas:
   - En el modelo secuencial, la forma de entrada se define implícitamente al añadir la primera capa.
   - En la API funcional, se define explícitamente un nodo de entrada al inicio del modelo.

3. Los conjuntos de entrenamiento y test

Como en toda aplicación de reconocimiento, se dispone de un conjunto de datos, en este caso se trata de imágenes de cartas de póker. Sin embargo, para trabajar con las redes CNN vamos a utilizar imágenes. La cuestión es qué tipo de imágenes vamos a emplear.

En primera instancia, vamos a intentar reconocer las imágenes de los motivos de las cartas. Para ello debemos recortar la imagen color de todos los motivos presentes en las cartas, separando los de entrenamiento de los de validación y test. Además, debemos etiquetar estos motivos, lo cual ya tendremos hecho de la práctica anterior. Por esto debemos tener los archivos 'trainCards.npz' y 'testCards.npz'. Hay que comprobar que tenemos bien etiquetados todos los motivos de ambos archivos, con el nombre o etiqueta real del motivo, pues ese nombre lo utilizaremos para dar nombre al archivo *.png que vamos a crear.

El siguiente script muestra un esquema del proceso a seguir.

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import cv2
import os
from clases_cartas import Card, Motif

################ Programa principal  ##################################

filecard = 'testCards.npz'   # 'cartasTrainig.npz  cartasTest.npz

cards = []
motifs = []
   # Crear la estructura de cartas
npzfile = np.load(filecard, allow_pickle=True)
cards = npzfile['Cartas']

le = len(cards)
j=1000;
for i in range(0,le):
   print('carta[',i,']')
   mots = cards[i].motifs
   ll = len(mots)
   for m in mots:
      cnt = m.contour
      x,y,w,h = cv2.boundingRect(cnt)
      img = cards[i].colorImage
      roi_color = img[int(y):int(y+h), int(x):int(x+w)].copy()   # Recorte de la imagen del motivo
      img = cv2.cvtColor(roi_color, cv2.COLOR_BGR2RGB)
      label = m.motifLabel

      name = './Motifs_train/' + label + '_' + str(j) + '.png'
      print(name,'  ',w,h)
      image = Image.fromarray(img)
      image.save(name)
      j = j+1

     # Idem con el motive girado 90º
     ……
      j = j+1

     # Idem con el motive girado 180º
     ……
      j = j+1
     # Idem con el motive girado 270º
     ……
      j = j+1
```

Observe que se obtiene un recorte de la imagen original de la carta en color, convertida a RGB, y se salva en un archivo *.png. Recuerde que las clases de Card y Motif se encuentran en el archivo '*clases_cartas.py*', que debe estar en la ruta actual de la práctica.

Un aspecto importante de las redes CNN es que necesitan muchas imágenes para entrenar, lo cual no es nuestro caso. Por ese motivo, también se intentarán salvar versiones del mismo motivo, pero giradas 90º, 180º y 270º. Para esto habrá que emplear una función np.rot90().

Para introducir estos datos en una red hay que utilizar las funciones que preparan los datos. Para esto se pueden emplear distintas bibliotecas:

- TensorFlow, con objetos tf.data.Dataset
- PyTorch, con objetos DataLoader
- Keras, con objetos PyDataset

Nosotros emplearemos Keras 3.0. Las funciones a emplear son las siguientes:

```
keras.utils.image_dataset_from_directory(
    directory,
    labels="inferred",
    label_mode="int",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=False,
    data_format=None,
    verbose=True,
)
```

Que genera un tf.data.Dataset desde las imágenes en el /los directorios. La estructura de esos directorios debe ser como se muestra a continuación. Es muy importante saber el orden de las carpetas, pues la función utiliza los nombre de estos como valores de las clases, en el orden alfabético que se obtiene mediante os.walk(Directorio_principal).

```
Directorio_principal/
… clase_1/
…….imagen1_1.png
…….imagen1_2.png
… clase_2/
…….imagen2_1.png
…….imagen2_2.png
… clase_3/
…….imagen3_1.png
…….imagen3_2.png
```

Para describir los parámetros, véase https://keras.io/api/data_loading/image/

Para poder crear las carpetas de las clases, siguiendo el esquema anterior, se propone el siguiente ejercicio:

► *Ejercicio 1 – Conjuntos de entrenamiento y test*

1. Introduzca el script anterior, en un archivo 'obtener_imagenes_motivos.py', y haga las modificaciones pertinentes.
2. En la carpeta de la práctica, cree dos sub-carpetas: './Motifs_train/' y './Motifs_test/'.
3. Tome los archivos de las cartas etiquetadas en 'trainCards.npz' y ejecute el script anterior para guardar las imágenes de los motivos en las sub-carpetas siguientes, dependiendo de la clase a que pertenece cada motivo.
   carpetas = ['000', '002', '003', '004', '005', '006', '007', '008', '009', '00A', '00J', '00K', '00Q', 'corazones', 'picas', 'rombos', 'treboles', 'variados']

   Puede probar y acabar el script tras ocho o diez motivos, para comprobar si las imágenes se han salvado correctamente. Después complete el apartado.

► *Ejercicio 2 – Conjuntos de test*

1. Repita el proceso para las cartas de test, etiquetadas en 'testCards.npz', y guarde las sub-carpetas de las clases en '. /Motif_test/'.

En general se prefieren las funciones tf.data de TensorFlow, porque son mucho más rápidas. Pero existe otra versión de Keras más antigua, llamada *ImageDataGenerator* , que también se puede utilizar, aunque sea más lenta, pero permite la inclusión fácil de la técnica de *Data Augmentation*. El proceso se muestra a continuación.

```
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras import Model
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers, models, Input


#Hyperparameters
target_size = (120,120)
batch_size = 32
epochs=50
labels_txt = ['000', '002', '003', '004', '005', '006', '007', '008', '009',
              '00A', '00J', '00K', '00Q', 'corazones', 'picas', 'rombos', 'treboles',  'variados']
# Importando el set de datos
dataset_path = "./Motifs_train/"   # Path donde tengo el dataset
min_num_samples = 5

print("all samples:")
dirs = os.listdir(dataset_path)
num_samples  = [len(files) for r, d, files in os.walk(dataset_path)]
num_samples = num_samples[1:] # exclude first top directory
print(dirs)
print(num_samples)

ok_samples = [ [j,i] for (i,j) in zip(num_samples,dirs) if i >= min_num_samples ]
print("samples with more than " + str(min_num_samples) + " samples" , ok_samples)

labels = [ j for (i,j) in zip(num_samples,dirs) if i >= min_num_samples ]
print(labels)
```

```
# Data generators para poder hacer data augmentation
datagen  = ImageDataGenerator(
    rotation_range=int(180*0.1),
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip = True,
    # preprocessing_function = preprocess_input
    rescale = 1./255
    )

print("train_ds:")
train_ds = datagen.flow_from_directory("./Motifs_train/", classes=labels, target_size = target_size, batch_size=batch_size,class_mode='categorical')
print("val_ds:")
val_ds = datagen.flow_from_directory("./Motifs_train/", classes=labels, target_size = target_size, batch_size=batch_size,class_mode='categorical')

datagen = ImageDataGenerator(
    # preprocessing_function = preprocess_input
    rescale = 1./255
)

print("test_ds:")
test_ds = datagen.flow_from_directory("./Motifs_test/", classes=labels, target_size = target_size,class_mode='categorical',shuffle=False)
```

► *Ejercicio 3 – Creación de los datasets*

1. Introduzca el script anterior en un archivo *myCNN.py* y ejecútelo.
2. Compruebe si los *datasets* generados (*train_ds, val_ds, test_ds*) son correctos.

Un aspecto importante es que no hemos diferenciado un conjunto de validación. Hemos empleado el mismo conjunto de entrenamiento como conjunto de validación. Para hacer el proceso algo más rápido podemos tomas unos cuantas imágenes de entrenamiento y repetirlas como de validación, en una carpeta ./'Motifs_val/'. El caso es que no disponemos de suficientes imágenes como para hacer divisiones consistentes, con suficientes muestras.

4. Creación del modelo CNN

Ahora vamos a crear un modelo de red CNN. El siguiente script incorpora las funciones habituales. Para comprobar los parámetros y el resultado de esas funciones consulte la ayuda.

https://keras.io/2.16/api/models/

```
def myCNN(width,height,depth, classes):

    inputShape = (height, width, depth)

    model = Sequential()
    model.add(Input(shape=inputShape))
    model.add(layers.Conv2D(filters=64,kernel_size=(3,3),padding="same",activation="relu",name="Conv2D1"))
  # model.add(layers.Conv2D(filters=64,kernel_size=(3,3),padding="same",activation="relu"))
    model.add(layers.MaxPool2D(pool_size=(2,2),strides=(2,2),name="MaxPool2D1"))
    model.add(layers.Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu",name="Conv2D2"))

    model.add(layers.MaxPool2D(pool_size=(2,2),strides=(2,2),name="MaxPool2D2"))
    model.add(layers.Conv2D(filters=32, kernel_size=(3,3), padding="same", activation="relu",name="Conv2D3"))
    model.add(layers.MaxPool2D(pool_size=(2,2),strides=(2,2),name="MaxPool2D3"))
```

```
    model.add(layers.Conv2D(filters=16, kernel_size=(3,3), padding="same", activation="relu",name="Conv2D4"))
    model.add(layers.MaxPool2D(pool_size=(2,2),strides=(2,2),name="MaxPool2D4"))

    model.add(layers.Flatten(name="Flatten"))
    model.add(layers.Dense(units=200,activation="relu",name="Dense1"))
# model.add(layers.Dropout(0.3))
    model.add(layers.Dense(units=classes, activation='softmax',name="FinalStage"))

    return model

# Instanciamos el modelo
model = myCNN(target_size[0],target_size[1],3, train_ds.num_classes)
# Compilamos el modelo
model.compile(loss='categorical_crossentropy', optimizer=SGD(0.005), metrics=['accuracy'])

# Vamos a visualizar el modelo prestando especial atencion en el numero de pesos total y el numero de pesos entrenables.
model.summary()

# Entrenamos el modelo
print("[INFO]: Entrenando la red...")
```

► *Ejercicio 4 – Creación del modelo CNN*

1. Introduzca el script anterior en el archivo *myCNN.py* y ejecútelo.
2. Compruebe si el modelo se ha creado correctamente, con la información que se obtiene del model.summary().

5. Entrenamiento del modelo CNN

El siguiente paso es el entrenamiento del modelo CNN. Para ello utilizamos la función model.fit(), indicando el datase de entrenamiento, el de validación y el número de epocs o pasadas del bucle de optimización, que se repite con todas las muestras.

```
# Entrenamos el modelo
print("[INFO]: Entrenando la red...")
H = model.fit(train_ds, epochs=epochs, validation_data=val_ds, verbose=1)

# Graficas
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, epochs), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, epochs), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, epochs), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, epochs), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()

# Almacenamos el modelo empleando la funcion model.save de Keras
model.save("./myCNN.h5")
```

El resultado se muestra por pantalla y se guarda en un archivo '*myCNN.h5*'. Se pueden estudiar otros formatos para guardar la red entrenada resultante. El entrenamiento es un proceso lento, que puede durar más de una hora en un computador sin GPU (Graphic Proccess Unit).

► *Ejercicio 5 – Entrenamiento del modelo CNN*

1. Tome el script anterior e insértelo en el archivo *myCNN.py* y ejecútelo.
2. Comprueba los resultados.

6. Evaluar el modelo CNN

Para evaluar el modelo CNN entrenado se puede utilizar el siguiente script, donde aparecen las funciones de *keras* más significativas:

```
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix
from sklearn.metrics import  classification_report, ConfusionMatrixDisplay, matthews_corrcoef


print("test_ds:")
test_ds = datagen.flow_from_directory("./Motif_test/", classes=labels, target_size = target_size, class_mode='categorical', shuffle=False)

# Cargar red ya entrenada
model = load_model("./myCNN.h5")
# Evaluamos con las muestras de test
print("[INFO]: Evaluando modelo...")
results = model.evaluate(test_ds)
print("test loss, test acc:", results)

# Generate predictions (probabilities -- the output of the last layer) on new data using predict
print("Generate predictions")
predictions = model.predict(test_ds)
print("predictions shape:", predictions.shape)
# Obtenemos el report
pred = predictions.argmax(axis=1)
class = test_ds.classes
print(classification_report(class, pred, target_names=labels)) # Etiquetas en decimal #(X)
print('Confusion Matrix')
cm = confusion_matrix(class)
print(cm)
MCC = matthews_corrcoef(class, pred)
print('MCC: ', MCC)
```

Ejercicio 5 – Evaluar el modelo CNN y visualizar resultados

1. Ejecute el código anterior en el mismo archivo *myCNN.py* o en otro. No olvide importar todos los módulos de Python necesarios.
2. Compruebe los resultados.
3. Guarde los resultados en un archivo *.txt o similar para su posterior uso.

Los resultados del modelo son los siguientes:

accuracy			0.82	516
macro avg	0.57	0.53	0.50	516
weighted avg	0.85	0.82	0.82	516

```
[ 7  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  5  1  0  0  0  2  0  0  0  0  0  0  0  0  0  0  2]
 [ 0  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0  4  0  0  4  0  1  0  0  0  0  0  0]
 [ 0  0  1  0  1  0  6  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  2  2  0  0  4  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  6  0  0  0  0  0  0  0  0  0  0  1]
 [ 0  0  5  0  2  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  4  0  0  0  2  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0  3  0  0  2  0  1  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  7  0  0  0  0  0  0  0  0  0  0  1]
 [ 0  0  0  0  0  0  2  0  0  0  0  6  0  0  0  0  0  0]
 [ 2  0  0  0  3  0  3  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 93  0  2  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 91  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  3 88  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 87  0]
 [ 1  1  0  4  0  0  4  0  2  1  0  2  0  1  1  1  5 24]
```

Se observa que los palos se clasifican bastante bien, pero los números son un fracaso, pues hay muchas equivocaciones. Lógicamente, esto de debido al escaso número de muestras de algunos números. Por ejemplo, las 'J' y 'Q' no han sido predichos nunca. Los '9' y '6' se confunden, lo mismo que los '8' y '3'. Es probable que estos resultados se puedan mejorar, empleando un *dataset* ampliado.

Ejercicio 6 – Modificar visualización de resultados

1. En el código anterior modifique la visualización de resultados para mostrar la matriz de confusión como indica la figura 1.
   Para ello puede utilizar las funciones:

```
import Matplotlib.pyplot as plt
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

cm_display = ConfusionMatrixDisplay(cm, display_labels=labels)
cm_display.plot(xticks_orientation='vertical')
plt.show()
```

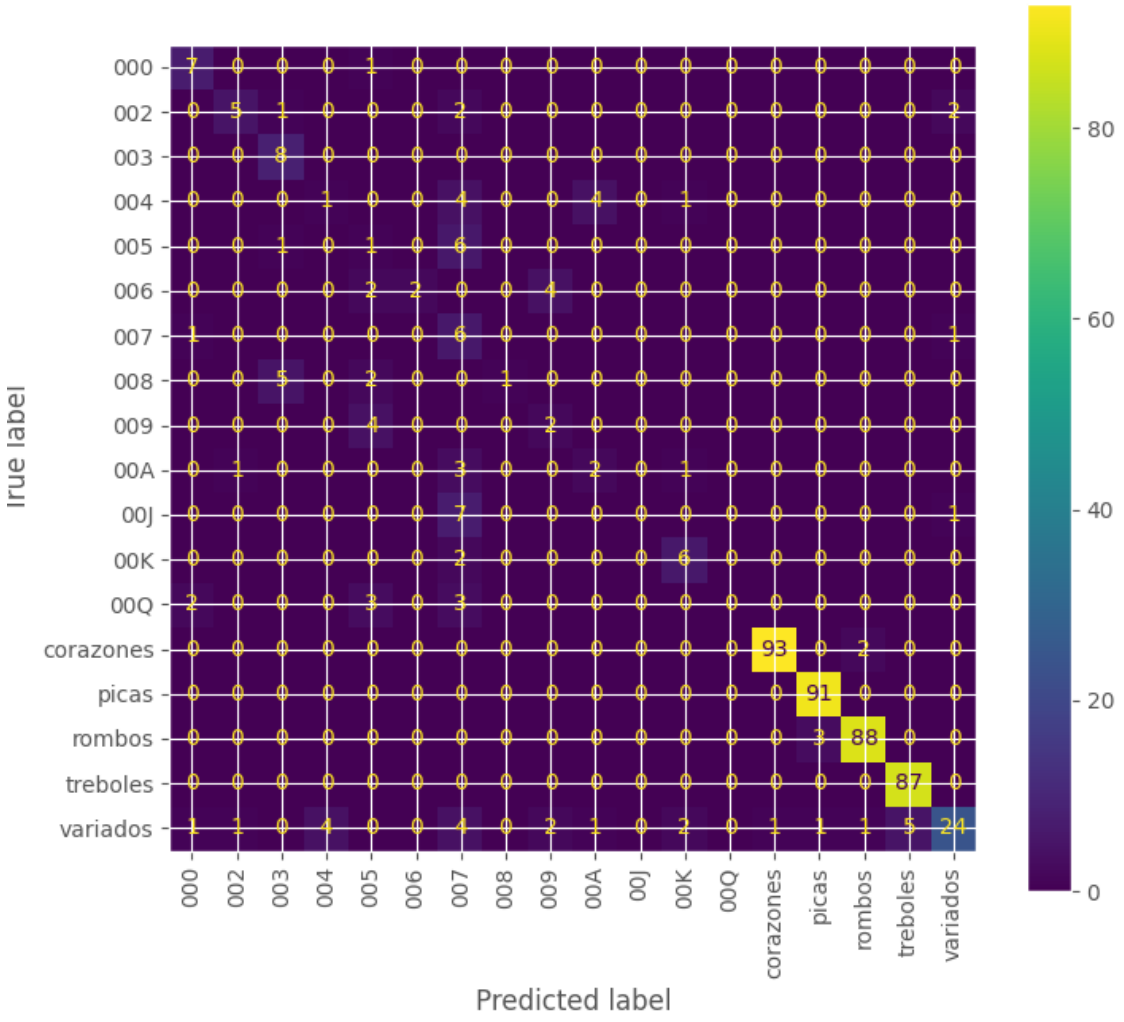

Figura 1 – Visualización de la matriz de confusión.

7. Utilización de una red estándar: VGG16

En este último paso se va a probar otra red CNN. Para esto es necesario descargar esa red desde *keras*, y utilizar las funciones apropiadas. Para más información consulte la página:

https://keras.io/api/applications/#usage-examples-for-image-classification-models

Tenga cuidado con el preproceso que necesitan algunas redes. También con los tamaños de la imagen de entrada y el número de clases de salida. La idea es no congelar ninguna capa de la red, pero emplear los coeficientes que vienen por defecto en la red como valores iniciales. Se pueden emplear entre 50 y 80 epocs.

Es interesante comparar ambas redes, para ver cual da mejores resultados.

8. Sobre el trabajo 2D de la asignatura VxC

Si hemos seguido fielmente los ejercicios anteriores, tenemos una versión básica operativa de la aplicación de reconocimiento de las cartas con CNN. La cuestión es ¿qué nos falta por hacer?

Pues lo que falta son las aportaciones personales al conjunto de códigos que hemos creado, y que constituyen el valor añadido de nuestro trabajo. Algunas de estas aportaciones ya se indicaron en la práctica anterior, como probar nuevas características, probar otros clasificadores o cambiar el formato de color.

Ahora se trata de realizar alguna de las siguientes aportaciones, para mejorar el resultado final:

- Probar con nuevas características de color

  Se puede emplear otros espacios de color, como HSI o CIE La*b* para guardar la información de color de cada motivo, en vez del esquema RGB. La aplicación no parece especialmente sensible a color, cuanto que los motivos sólo son de tonos rojos o negros, pero se puede probar.

- Aumentar el *Dataset* de entrenamiento:

  El conjunto de entrenamiento utilizado hasta ahora es el inicial de las cartas de póker, formado por una imagen de cada carta. Esto implica que sólo tenemos 8 muestras para las cifras, mientras que para los motivos de diamantes, picas, etc… tenemos muchas muestras. Esto puede dar lugar a un resultado pobre, que podemos mejorar aumentando los *Datasets*. Para esto tenemos un segundo conjunto de muestras en PoliformaT, que habrá que añadir y etiquetar.

- Probar con dos o más clasificadores CNN:

  Es muy conveniente probar con otro clasificador CNN, por ejemplo, un *InceptionV3*.

- Clasificación de las cartas

  Con el código básico tenemos clasificados lo motivos del *Dataset*, pero no tenemos etiquetadas las cartas en sí. Esto ya se indicó en la práctica anterior, con el script '*clasificarCartas.py*'. Ahora se puede repetir o ampliar ese programa con la inclusión de la clasificación con las CNN desarrolladas.

- Añadir o mejorar la visualización de resultados

  Se puede incluir una visualización más clara e informativa que la que tiene el código básico, especialmente en la parte final de clasificación y test de las cartas.

La entrega del trabajo 2D consistirá en generar un archivo pdf con la información de los scripts desarrollados incluyendo, en su caso, alguna visualización. De la parte final de los clasificadores, hay que indicar los experimentos realizados con cada clasificador y una tabla final de resultados.

En un apartado final de conclusiones, se pueden indicar los mejores resultados obtenidos, así como las dificultades encontradas. También, si ha sido fructífera la experiencia realizada.

No olvidar incluir un apartado de referencias bibliográficas o de webs.