Visión por Computador (VxC)

Grado en Informática Industrial y Robótica (GIIROB)

Práctica 7:

Clasificadores estadísticos

Contenido:

1. Introducción
2. Los conjuntos de entrenamiento y test
3. Extracción de los vectores de características de cada carta
4. Etiquetados de las clases de los motivos: Clases reales
5. El clasificador
6. Sobre el trabajo2D de la asignatura

Autor:

José Miguel Valiente González

Bibliografía:

- J. Howse & J. Minichino. “Learning OpenCV 4 Computer Vision with Python 3” (third Ed.) Packt Publishing Ltd. 2020.
- Sandigan Dei, “Python Image Processing. Cookbook”. Packt Publishing Ltd. 2020.
- B.R. Hunt; R.L. Lipsman, J.M. Rosenberg. “A guide to MATLAB”. Cambridge University Press. 2001.
- Rafael Gonzalez, Richard Woods, Steven Eddins. ”Digital Image Processing Using Matlab”. Second Edition, McGraw Hill, 2010
- Official documentation for Python 3.10 https://docs.python.org/es/3/tutorial/index.html

En esta práctica vamos a estudiar las funciones de OpenCV para las operaciones de clasificación y reconocimiento de formas en general. Este tipo de operaciones son las que vamos a utilizar en la aplicación básica para el 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 OpenCV para hacer operaciones de clasificación y reconocimiento de formas.
- Preparación del código básico para el trabajo 2D.

Material

- Scripts de PYTHON: *segmentar_cartas.py*, *clases_cartas.py,* *etiquetar_cartas_OK.py'* y *modelo_kNN.py*.
- Archivos de imagen de las cartas de póker, utilizadas en las prácticas anteriores.

Se aconseja crear una carpeta '*/trabajo2D/*' y copiar todos los scripts indicados y copiar la carpeta '*/images/*' con el *dataset* de las cartas de póker. Así mismo, 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

En la práctica anterior se han desarrollado diversas funciones que nos han permitido segmentar la imagen original en color y, en primera instancia, extraer los objetos 'Carta' distinguiéndolos del fondo. (Tapete verde). En el script de segmentación de las cartas, leemos cada archivo de imagen, lo umbralizamos y extraemos los objetos de más de 300000 píxeles, que se corresponden con las cartas. Para cada objeto, declaramos una variable de la clase Card, donde anotamos todos los datos de esa carta.

```
import numpy as np
……
import cv2
from clases_cartas import Card, Motif

  Cards = []   # Lista inicial de las cartas
  filecard = 'cartas.npz'
  icard = 0
  ……..
      c = Card()              # Crea el objeto carta
      c.cardId = icard        # Rellena sus datos
      ……..
      Cards.append(c)      # Añade la carta a la lista
      icard +=1
      ……
  ……
  np.savez(filecard, Cartas=Cards)    # Al final se guarda la lista de cartas
```

El resultado es una lista, llamada Cards, de objetos de tipo Card, con los campos que describen el objeto en su conjunto. Al final del script se guarda esta lista en un archivo *cartas.npz* para su uso posterior.

A continuación, se muestra la información de esta clase:

```
import numpy as np

class Card:

    DIAMONDS = 'Diamonds'   # Rombos
    SPADES = 'Spades'           # Picas
    HEARDS = 'Hearts'           # Corazones
    CLUBS ='Clubs'              # Tréboles
        # Figuras y cifras de las cartas de póker
    SUITS = ('Rombos','Picas','Corazones','Treboles')
    FIGURES = ('0','A','2','3','4','5','6','7','8','9','J','Q','K') # Se accede mediante Card.FIGURES[i]

    def __init__(self):  # Constructor
      self.cardId = 0
      self.realSuit = 'i'             # Etiquetas reales de la carta
      self.realFigure = 'iii'
      self.predictedSuit = 'o'    # Etiquetas predichas de la carta
      self.predictedFigure = 'ooo'
      bboxType = [('x', np.intc),('y',np.intc),('width',np.intc),('height',np.intc)]
      self.boundingBox = np.zeros(1, dtype=bboxType).view(np.recarray)
      self.angle = 0.0
      self.grayImage = np.empty([0,0], dtype=np.uint8)
      self.colorImage = np.empty([0,0,0], dtype=np.uint8)
      self.motifs = []    # Lista de motivos de la carta
```

En un segundo nivel de proceso, se han tomado las imágenes recortadas de esas cartas y, con una nueva función - *segmentar_objetos_carta(c)* - se han extraído los pequeños objetos presentes dentro de cada carta '*c*', a los que denominamos '*Motivos*'. Estos motivos son de la clase Motif, que contiene todos los datos que hemos podido extraer de ellos. El resultado se ha añadido a la carta mediante una lista llamada motifs, de la misma forma que hacíamos con las cartas.

```
……..
    m = Motif()             # Crea el objeto motivo
    m.motifId = imot        # Rellena sus datos
    ……….
    c.motifs.append(m)   # Añade a la lista de motivos de la carta
    imot +=1
    ……
```

Los detalles de la clase Motif se muestran a continuación:

```
class Motif:

  MOTIF_LABELS = ('Diamonds','Spades','Hearts','Clubs','0','2','3','4','5','6','7','8','9','A','J','Q','K','Others')

  def __init__(self):  # Constructor
    self.motifId = 0
    self.motifLabel = 'i'
    self.motifPredictedLabel = 'iii'
    self.area = 0.0
    self.contour = []
    self.perimeter = 0.0
    self.features = []
    self.moments = []
```

```
        self.huMoments = []
        self.centroid = []
        self.circleCenter = []
        self.circleRadious = 0.0
```

Un último detalle acerca del código del archivo '*segmentar_cartas.py*' es la visualización de los resultados, que se puede hacer en distintas partes del código. Se aconseja reunir las visualizaciones en uno o varios grupos y englobarlos con un 'IF' mediante una constante booleana 'VISUALIZAR'. Como ejemplo, para la función de '*segmentar_objetos_carta(c)*' podemos incluir:

```
  # Visualización de resultados de los motivos
            if VISUALIZAR:
               img = roi_color.copy()
               cv2.rectangle(img,(x0,y0),(x0+w0,y0+h0),(0,255,0),2)
               cv2.drawContours(img, [cnt], -1, (255,0,0), 5)
               cv2.circle(img,center,radious, (255,0,0),2)
               print(f ' * Índice {c.cardId}  -  Nº puntos {len(cnt)} , - Área motivo: {motif.area}  - Perímetro: {motif.perimeter} \n')
               cv2.imshow(window_roi, img)
               cv2.waitKey()
```

De esta forma, si ponemos VISUALIAR = True se mostrarán lo resultados de cada motivo y esperará a pulsar una tecla para pasar al siguiente. Lo mismo ocurrirá con cada carta. Pero, si ponemos VISUALIZAR = False, se extraerán todas las cartas de la carpeta sin esperas, lo cual costará muy poco.

► *Ejercicio 1 – Segmentar objetos de las cartas*

1. Tome el script *segmentar_cartas()*, introducido en la práctica anterior, y en la función *segmentar_objetos_carta(c)* introduzca el código que distingue los motivos.
2. Busque en la biblioteca de OpenCV funciones para obtener de un contorno: Área, perímetro, boundingBox, momentos, invariantes geométricas (HuMoments), etc., y añádalos a la función
3. Codifique la visualización con la variable booleana VISUALIZAR = True
4. Compruebe su correcto funcionamiento. Esto se completará en el ejercicio 3.

2. 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. Para realizar una correcta experimentación debemos dividir este *Dataset* en dos subconjuntos: Entrenamiento (Training) y Prueba (Test).

► *Ejercicio 2 – Conjuntos de entrenamiento y test*

1. Tome las imágenes disponibles en la práctica anterior. Cree en la misma carpeta '***images***' dos subcarpetas '***Training***' y '***Test***'.
2. Coloque en la carpeta **Training** todas las imágenes que contienen sólo una carta.
3. Coloque en la carpeta **Test** todas las imágenes que contienen varias cartas. Esta es una subdivisión tentativa, que podemos cambiar cuando queramos.

3. Extracción de los vectores de características de cada motivo

Las imágenes no son las características que empleamos en un clasificador estadístico. Son los Vectores de Características (Feature Vectors) lo que necesitamos, en particular los vectores asociados a todos los motivos de cada carta. Estos vectores son vectores n-dimensionales de números reales (double), donde cada dimensión representa una magnitud geométrica. Con estos vectores vamos a realizar la identificación de cifra y palo de las cartas, obteniendo una predicción de la clase de los motivos que contiene.

Como se ha visto en el apartado inicial, la estructura Motif incluye todas las características que hayamos podido extraer de cada motivo. También incluye una lista llamada *motif.features* que deberá ser ese vector de características antes indicado.

Así pues, en el código de identifica cada motivo - *segmentar_objetos_carta(c)* - hay que incluir al final el relleno de esas características. Algunos detalles de esta función son:

- En cada carta se tiene que umbralizar la imagen de niveles de gris *c.grayImage* y después obtener las componentes conectadas de la imagen binaria obtenida.
- Se obtendrán múltiples objetos, que habrá que filtrar por tamaño. Sólo los objetos entre MIN_AREA = 1000 y MAX_AREA = 40000 deben ser considerados.
- Para cada objeto que pasa el filtro de tamaño, hay que obtener los contornos. Pero, aun así, se obtienen múltiples contornos (de objetos y de huecos). Por eso hay que añadir un segundo filtro, para quedarse sólo con los objetos de máximo contorno. Para ello emplearemos:

```
contours_roi, hi = cv2.findContours(componentMask2, cv2.RETR_LIST , cv2.CHAIN_APPROX_NONE)
cnt = max(contours_roi, key=lambda k: len(k))   # contorno de mayor longitud
```

- Algunas de las características que podríamos incluir en la estructura Mofit serían: Momentos, invariante geométricos, excentricidad, Solidity, Ratio (eje mayor, eje menor), etc… . Para ello consulte https://docs.opencv.org/4.x/d1/d32/tutorial_py_contour_properties.html

- Los datos que se incluyen en ambas clases se pueden ampliar/modificar al gusto del usuario, para incluir toda aquella información que considere relevante para la aplicación. Repase la práctica anterior para ver qué tipo de información se puede incluir.

► *Ejercicio 3 – Características de los motivos*

1. Tome el script anterior, y en la función *segmentar_objetos_carta(c)* introduzca el código que añade las características a la lista *motif.features*. Ojo que se trata de una lista, por lo que hay que introducir los datos mediante *motif.features.append(datos)*, pero con cuidado de que todos los datos sean números reales o enteros.
2. Compruebe que el código funciona correctamente y que, al final, todo se guarda en un archivo **.npz*.
3. Ejecute el script para las cartas de entrenamiento y obtenga '*trainCards.npz*'. En este caso utilice VISUALICAR=False y verá que el proceso es muy rápido.
4. Repita el proceso para las cartas de test y obtenga '*testCards.npz*'.

IMPORTANTE:

Este ejercicio es bastante crítico, en el sentido de que debemos asegurarnos el incluir TODAS las características que podamos antes de pasar al etiquetado en el ejercicio siguiente. Esto es debido a que los archivos *.npz guardan la estructura de las clases Card y Motif. Si alguna clase cambia, porque queremos añadir más datos, lo que tengamos en esos archivos ya no vale, incluido el etiquetado.

4. Etiquetado de las clases de los motivos: clases reales

Ahora que ya tenemos los archivos *trainCards.npz* y *testCards.npz* , con las estructuras de datos de toda la información y los vectores de características de todos los motivos, vamos a proceder al reconocimiento. Pero, antes de pasar al clasificador debemos '**etiquetar las clases'** de cada motivo, de forma manual. Como hay un número muy alto de motivos, esta labor será tediosa y costosa.

Para esto, las clases Card y Motif incluyen algunos campos para identificar las etiquetas correspondientes. En la clase Mofit, cada motivo tiene los campos *motif.label = 'i'* y *motif.predictedLabel = 'iii'*, que identifican las etiquetas real y predicha de los mismos. Los valores 'i' e 'iii' son los valores iniciales que indican que todavía no se han etiquetado.

La etiqueta real constituye 'la verdad' (Ground Truth), que debemos poner a mano para poder comprobar, más tarde, si lo que predice nuestro clasificador es correcto o no. Como su nombre indica, la *predictedLabel* indica la etiqueta que obtendrá automáticamente nuestro sistema. Las posibles etiquetas que identificaremos en los motivos serán:

```
MOTIF_LABELS = ('Rombos','Picas','Corazones','Treboles','0','2','3','4','5','6','7','8','9','A','J','Q','K','Otros')
```

Observe que el número '10' se etiqueta con el '0'. Si nos aparece también el '1' lo marcaremos como 'Otros' pues esa figura es un simple palito vertical que no nos distingue nada.

Por otro lado, está el etiquetado de las cartas en su conjunto. Cada carta se etiqueta con dos clases: Palo (Suit) y número/letra (Figure). Para ello, la clase Card dispone de los atributos *card.realSuit*; *card.predictedSuit*, *card.realFigure* y *card.predictredFigure*. Al inicializar la carta, se toman los valores iniciales que se indican en la propia definición de la clase (*i,iii,o,ooo*). La figura 1 muestra que deberíamos etiquetar las tres cartas como '7 de picas', 'As de rombos' y 'J de rombos'

```
SUITS = ('Rombos','Picas','Corazones','Treboles')
FIGURES = ('0','A','2','3','4','5','6','7','8','9','J','Q','K') # Se accede mediante Card.FIGURES[i]
```

Figura 1 – Ejemplo de cartas de póker.

El etiquetado de cada carta se puede hacer automáticamente cuando etiquetamos sus motivos. Como asumimos que el usuario no se equivoca, cuando éste introduce una etiqueta de un motivo que se corresponde con los palos, asumimos que esa es también el palo de la carta. Lo mismo con las figuras. Observe que cada carta debe tener sólo un tipo de palo y de figura. La clase 'otros' no pertenece a ninguna de las dos categorías, por lo que nunca se etiquetará así a una carta. A lo sumo, quedará sin etiquetar con sus valores iniciales.

Ejercicio 4 – Etiquetado de los motivos y las cartas

1. Tome el script de Python '*etiquetar_cartas_OK.py*' de PoliformaT. Repase y comprenda el código y ejecútelo. Utilice VISUALIZAR=True.
2. Cancele el etiquetado (<ESC>) y salga después de haber etiquetado los motivos de un par de cartas. Vuelva a ejecutar el script y compruebe que empieza el etiquetado en la última carta en la que acabo antes. Esto permite realizar en etiquetado en varias sesiones, pues es una tarea muy tediosa.
3. Termine el etiquetado de las cartas de entrenamiento trainCards.npz y repita el proceso con testCarrds.npz.

Observaciones:

Tenga cuidado con este proceso pues puede quedar algún motivo o carta sin clasificar. Observe que podemos salir en cualquier momento. Si volvemos a entrar nos llevará a la primera carta (no motivo) sin etiqueta. Pero esto no ocurre con los motivos. Para afrontar esto, podemos ir adelante y atrás en cada carta y motivo, buscando aquellos no etiquetados. El proceso es muy tedioso por lo que se aconseja no dejan nada sin etiquetar. Ojo que nada nos protege de dar una etiqueta errónea.

5. El clasificador

Ahora vamos a proceder con la última etapa de todo proceso típico de visión, la de reconocimiento de formas, donde pretendemos identificar el palo y la cifra o símbolo de cada carta de póker que aparezca en la imagen, mediante técnicas de reconocimiento de formas.

La clase básica de **OpenCV** para todos los algoritmos, más o menos complejos, se tiene en la clase cv2.Algorithm. A partir de ésta aparecen clases para clasificación, bajo la clase derivada cv2.ml.StatModel, que se engloban en el módulo de *Matching Learning*.

https://docs.opencv.org/3.4/db/d7d/classcv_1_1ml_1_1StatModel.html#details

De todos los modelos de clasificadores indicados, nosotros vamos a probar dos: K-Nearest Neighbors model y Support Vector Machines.

Clasificador k_NN

El clasificador de k vecinos más próximos se crea mediante un modelo cv2.ml.kNearest y dispone de las siguientes funciones:

- cv2.ml.kNearest.create() : Crea un modelo kNN vacio.
- cv2.ml.kNearest.load(filename) : Crea un modelo kNN desde un archivo, que se guardó mediante cv2.cv.Algorithm.save(filename).
- Los métodos públicos de esta clase son:
  - int getAlgorithmType () # Devuelve el tipo de algoritmo 'BRUTE_FORCE' o 'KDTREE'
  - int getDefaultK () # Devuelve k
  - int getEmax () # Devuelve el parámetros para KDTree
  - bool getIsClassifier () # Devuelve si el clasificador debe ser entrenado
  - void setAlgorithmType (int val)
  - void setDefaultK (int val)
  - void setEmax (int val)
  - void setIsClassifier (bool val)

El modelo del clasificador kNN se debe crear y luego entrenar mediante el método train de ese objeto:

- cv2.ml.StatModel.train(trainData[, flags]) # Modo complejo
- cv2.ml.StatModel.train(samples, layout, responses) # Modo simple
  - trainData: Estructura de la clase cv2.ml.TrainData. Permite múltiples operaciones para crear y particionar conjuntos de datos. También permite asignar pesos.
  - samples: vector de N muestras de d características float32, en filas o columnas.
  - layout: cv2.ml.ROW_SAMPLE o cv2.ml.COLUMN_SAMPLE
  - responses: vector de N respuestas de dimensión 1 (int32/float32). Cada clase es un número.

Una vez entrenado, la predicción con nuevos valores se realiza mediante:

- retval, results, neighborResponses, dist = cv2.ml.KNearest.findNearest(samples, k[, results[, neighborResponses[, dist]]])
  - samples: Matriz de dimensión Nxd: 'N' muestras de dimension 'd'. Una muestra por fila.
  - K: Número de vecinos a utilizar >1.
  - results: Vector de predicciones o clases, una por cada muestra (Nx1)
  - neighborResponses: Valores de salida de los k vecinos de cada muestra (matriz Nxk)
  - dist: Distancias de cada muestra a los k vecinos más próximos (matriz Nxk)

Para cada muestra de entrada, el modelo obtiene los k vecinos más próximos. En caso de regresión, el resultado es la media de las k respuestas. En el caso de clasificación, el resultado se obtiene por votación entre los k vecinos más próximos. Opcionalmente, también devuelve las k respuestas más próximas y sus k distancias a la muestra de entrada. Veamos un ejemplo:

```
import cv2
import numpy as np

        # Muestras de entrenamiento formadas por 200 muestras de 2 coordenados, con valores int[0..100]
trainData = np.random.randint(0,100,(200,2)).astype(np.float32)
        # Etiquetas de cada muestra con valores enteros 0 y 1
responses = np.random.randint(0,2,(200,1)).astype(np.float32)

        # Creación del modelo y entrenamiento
knn = cv2.ml.KNearest_create()
knn.train(trainData, cv.ml.ROW_SAMPLE, responses)

        # Nueva muestra desconocida (12, 15)
newcomer = np.array([(12,15)]).astype(np.float32)

        # Predicción con k=3
ret, results, neighbours ,dist = knn.findNearest(newcomer, 3)

        # Imprime resultados
print( "result: {}\n".format(results) )
print( "neighbours: {}\n".format(neighbours) )
print( "distance: {}\n".format(dist) )
```

que da como resultado:

```
result: [[0.]]
neighbours: [[1. 0. 0.]]
distance: [[64. 68. 90.]]
```

Ejercicio 5 – Entrenar y evaluar el modelo del clasificador kNN

1. Tome el script '*modelo_kNN.py*' de PoliformaT. Repase y comprenda el código de la función.

   Observe que se leen los datos de '*trainCards.npz*' y, para cada motivo de cada carta, se extraen las características y se apuntan en unas listas (*samples*, *responses*). Después estas listas se deben convertir en matrices, pues ese es el modelo de datos que requieren los procedimientos del clasificador.

2. Complete el código para incluir en *samples* todas las características que consideremos relevantes.
3. A continuación, complete el código para crear el modelo kNN y entrenarlo, siguiendo el mismo código que en del párrafo anterior.

   Observe que después se realiza la misma operación con las cartas de test (*testCards.npz*), solo que ahora el modelo ya está y hay que validarlo mediante la función *findNearest* indicada en el párrafo anterior.

4. Complete el código para la evaluación del modelo con las cartas de test.

   La visualización de los resultados se realiza mediante funciones de *sklearn.metrics*, con el siguiente código:

```
from sklearn.metrics import confusion_matrix, classification_report, matthews_corrcoef

CLS_REP=classification_report(real, pred, target_names=MOTIF_LABELS)
print('Classification report ', CLS_REP)
CONF_MAT = confusion_matrix(real,pred)
print('Confusion Matrix', CONF_MAT)
MCC = matthews_corrcoef(real, pred)
print('MCC: ', MCC)
```

5. Guarde los resultados en un archivo *.txt o similar para su posterior uso.

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 2.
   Para ello puede utilizar las funciones:

```
import Matplotlib.pyplot as plt

cm_display = ConfusionMatrixDisplay(……)
cm_display.plot(xticks_orientation='vertical')
plt.show()
```

   El resultado obtenido, con los conjuntos de entrenamiento y test que disponemos, estará alrededor del 75%, dependiendo de las características que empleemos.

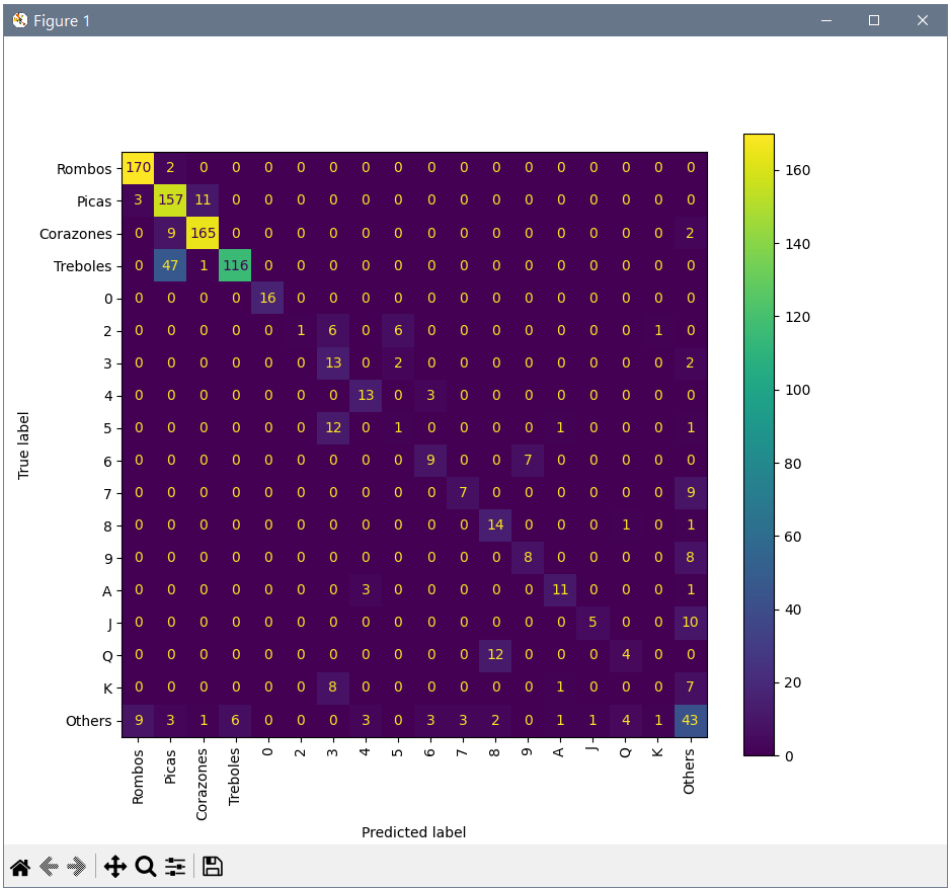

Figura 2 – Visualización de la matriz de confusión.

SI utilizamos para el test las mismas muestras que para el entrenamiento nos dará, lógicamente, un 100% de aciertos pues la distancia mínima de cada muestra es consigo misma.

A partir de este punto podemos hacer distintas pruebas, quitando o poniendo alguna característica, o probando con otros parámetros del kNN, por ejemplo con k=1,5,7,….

6. 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. 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. Esas aportaciones pueden ser algunas de las que se indican a continuación:

- Vectores de características:
  En el ejercicio 3, en la función '*segmentar_objetos_carta*()' se podía añadir el código para calcular nuevas características. También podemos pensar poner las cartas en una posición vertical y recalcular nuevas características ya que la invarianza a la rotación no es imprescindible.

- 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,… 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:

  Es muy conveniente probar con distintos clasificadores. En este caso se sugiere probar con un clasificador SVM (Support Vector Machine). La documentación de este clasificador en OpenCV se puede encontrar en :
  https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html

  Las ideas son similares al kNN, salvo que hay que definir más parámetros para el clasificador, pero el resto es igual que en el kNN.

- Clasificación de las cartas

Con el código básico tenemos clasificados lo motivos del Dataset, pero no tenemos etiquetadas las cartas en sí. En el ejercicio 4 se ha utilizado un scritps para clasificar los motivos, pero las cartas se clasificaban automáticamente mediante 'la verdad' que el usuario introducía.

Pero esto no es bastante. Debemos crear un nuevo código (c*lasificarCartas.py)* para dar una respuesta a la clasificación de cada carta. Observe que, en este caso, no hay que acudir a un clasificador sino a ver qué motivos hay en cada carta y, mediante un conjunto de reglas a estudiar, dar una indicación del palo y la cifra estimados de la carta. Es un tema que cada uno resuelve empleando el sentido común, o una idea maravillosa.

- 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.