PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
IRB2001- FUNDAMENTOS DE ROBÓTICA

TAREA ROBÓTICA COGNITIVA

Fecha Máxima de Entrega: 17/06, 23:59

Esta tarea se centra en el reconocimiento de poses humanas usando técnicas de reconocimiento visual. Para esto deberán entrenar un clasificador SVM, modificar uno de sus parámetros y analizar el rendimiento del modelo resultante.

1 Support Vector Machines

Como se vió en clases, los SVM permiten resolver problemas de clasificación. Si bien en clases vimos el caso de clasificación binaria, este tipo de clasificador puede extenderse también al caso de múltiples clases. Dado que en general los datos pueden no ser linealmente separables, a la hora de encontrar el plano separador nos gustaría dar al modelo una idea de que tanto se quiere evitar clasificar mal un dato. Para esto existe un coeficiente de penalización asociados a un conjunto de variables auxiliares o de slack (C) que nos ayuda a modificar el margen de distancia del hiperplano a los puntos a clasificar, permitiendo modelar problemas que no sean linealmente separables.

2 Espacio de Características (Feature space)

Dado que queremos clasificar imágenes con un SVM, necesitamos obtener una representación de éstas en un espacio de características. Una alternativa sería considerar cada pixel de la imagen como el valor de una dimensión del dato. Sin embargo, con imágenes RGB de resolución 388x284, nuestro espacio de características tendría 330576 dimensiones, donde cada dimensión (pixel) aportaría informaciíon limitada sobre el contenido de la imagen.

Como comentamos en clase, las redes neuronales de aprendizaje profundo (deep learning) han surgido recientemente como una poderosa herramienta que permite aprender características (*features*) que permiten aumentar significativamente el rendimiento de diversas tareas de clasificación, tales como reconocimiento de voz, texto, o imágenes. Como mencionamos en clase, en general, el entrenamiento de estas redes requiere ajustar millones de parámetros, lo cual es un proceso lento y altamente demandante de recursos computacionales. Afortunadamente, existen redes pre-entrenadas en base a grandes volúmenes de datos, las cuales pueden ser usadas para obtener un vector de características (*features*) de alta discriminabilidad. En esta tarea usaremos este tipo de *features* para realizar una aplicación relacionada a la clasificación de poses humanas.

3 Preprocesamiento: Estimación de poses humanas

Un problema interesante dentro de la comunidad de visión por computador es la clasificación de acciones humanas en videos. En años recientes se ha abordado este problema mediante la estimación de las poses de las personas. Un *framework* útil para esta tarea es PifPaf [1] cuyo repositorio original podemos encontrar en `https://github.com/vita-epfl/openpifpaf`. Esta herramienta nos ayudará a encontrar 17 puntos de junturas de articulaciones y puntos de rostros para tener un modelo de esqueleto humano, como muestra la Figura 1.

Figure 1: Puntos de esqueletos encontrados con PifPaf.

Este modelo no solo permite reducir la cantidad de información de la imagen para hacerla más manejable, sino que también ayuda a enfocar el análisis en información relevante para la clasificación de acciones humanas.
Para el desarrollo de la tarea contarán con una implementación que permite extraer los n esqueletos de una imagen a un arreglo multidimensional $skeletons[n][17][3]$. Las tres últimas posiciones del arreglo corresponden a una tripleta (x, y, c) donde c señala una confianza de que ese punto esté donde indica el modelo.

4 Google Colab

Para la implementación de su tarea usarán Google Colab `https://colab.research.google.com/`. Esto permitirá que ejecuten sus códigos usando GPUs (graphic processing units), las cuales son necesarias para la implementación de modelos de aprendizaje profundo. Más detalles sobre esto en la ayudantía de la tarea.

5 Métricas de rendimiento

Como vimos en clase, para realizar el entrenamiento de nuestro modelo, i.e., encontrar nuestro clasificador SVM, usaremos el denominado set de entrenamiento. Luego, para la evaluación del clasificador, debemos evaluar su capacidad para predecir nuevos datos (i.e. datos no usados en su entrenamiento). Ya que la cantidad de datos disponible es limitada, es necesario separar de antemano una porción de los datos (set de test) que se usará para evaluar rendimiento según la métrica de precisión.

5.1 Puntaje de precisión

Una de las métricas más simples para esta tarea es el puntaje de precisión (accuracy score). Este puntaje se obtiene comparando las respuestas correctas (datos clasificados en la categoría correcta) con respecto al total de las predicciones hechas.

$$accuracy\_score = \frac{\#Predicciones\_correctas}{\#Predicciones\_totales}$$

La fórmula anterior puede ser multiplicada por 100 para obtener el porcentaje de acierto sobre el set de test.

Figure 2: Lista de puntos de esqueleto humano encontrados con PifPaf.

5.2 Matriz de confusión

Si bien el puntaje de precisión nos permite evaluar el rendimiento de nuestro modelo, también nos interesa saber en qué casos éste se equivoca o qué tan bueno es para predecir ciertas categorías específicas. Para esto otra métrica interesante es la matriz de confusión. Ésta se construye comparando las predicciones del modelo con las categorías reales de las instancias del set de test. Cada celda de la matriz contiene la cantidad de datos correspondiente a la clase indicada con el rótulo o índice fila que fueron erróneamente clasificados como parte de la clase indicada con el rótulo o índice columna. Por ejemplo, en la figura 2 para un clasificador de 7 categorías (clases 0 a 6), se puede observar que 19 ejemplos de la **clase 3**, fueron clasificados erróneamente como **clase 5** (*Nota: en algunos textos la matriz de confusión puede ser definida según una convención inversa, es decir, columnas indicar la clase correcta y filas las predicciones.*). Así, la diagonal de la matriz representa los datos del set que fueron correctamente predichos.
Una práctica común a la hora de observar matrices de confusión es obtener una matriz normalizada con el total de los datos. Con esto cada celda representa la porción de datos clasificados en dicha categoría contra la respuesta correcta.

6 Actividades

El objetivo de la tarea es el reconocimiento de acciones por medio de la comparación de secuencias de poses. Las poses las obtendremos con el ya mencionado *OpenPifPaf* y la comparación la realizaremos entrenando clasificadores SVM, y probando su rendimiento. Específicamente, usaremos el código en el archivo Ayudantía-SVM-fundamentos-de-robótica.ipynb. Allí encontraremos una serie de scripts que nos

Predicción

Real

	0	1	2	3	4	5	6
0	972	1	1	0	0	1	3
1	0	1129	3	0	1	1	1
2	7	6	992	5	1	0	2
3	0	1	2	970	1	19	0
4	0	7	0	0	944	0	3
5	1	1	0	12	2	860	5
6	4	2	0	0	3	5	944

Figure 3: Ejemplo de matriz de confusión para 5 categorías

ayudarán con el trabajo.

6.1 Librerías

6.1.1 Scikit-learn

La actividad la realizaremos utilizando códigos en lenguaje de programación Python. Adicionalmente, usaremos la librería scikit-learn `http:\\www.scikit-learn.org`, la cual provee una gran gama de técnicas de aprendizaje de máquina, tal como clasificadores del tipo SVM disponibles en la clase SVC. En particular, la clase SVC se importa e inicializa de la siguiente manera:

```
from sklearn.svm import SVC
classifier = SVC()
```

Es importante entender algunos métodos definidos en la clase *SVC* de la librería *scikit learn*. Además se han importado desde la librería funciones que permiten obtener las métricas descritas en la sección 5. Se presenta a continuación una breve descripción de las funciones que necesitará utilizar.

- `classifier.set_params(kernel='linear', C=C)`: le permitirá fijar un kernel lineal para la experiencia y fijar la variable de slack C. Puede cambiar la función de kernel a cualquiera de los siguientes disponibles {`'linear'`,`'poly'`,`'rbf'`,`'sigmoid'`,`'precomputed'`} de ser necesario. Se recomienda comenzar con `'linear'`.
- `classifier.fit(X_train, y_train)`: entrena al clasificador `classifier` usando los ejemplos en la matriz de caractarísticas `X_train` con su respectivo vector de etiquetas `y_train` (categoría a las cuales corresponde cada dato).
- `y_pred = classifier.predict(X_test)`: retorna un vector con las categorías en las cuales el clasificador predice que va cada ítem de `X_test`.
- `score = accuracy_score(y_test, y_pred)`: retorna un puntaje comparando las categorías en `y_test` con las categorías obtenidas de las predicciones (`y_pred`).
- `matrix = confusion_matrix(y_test, y_pred)`: retorna una matriz de confusión obtenida de comparar las categorías predichas con las categorías del set de datos. Para visualizar esta matriz usted deberá ocupar la función disponible en Ayudantía-robotica-cognitiva.ipynb.
- `plot_confusion_matrix(matrix, target_names, norm=False)`: muestra la matriz de confusión mat, si se quiere normalizar la matriz hay que setear norm como `True`, en otro caso se deja como `False`. Se le debe entregar una lista `target_names` con los nombres de las categorías.

Una función de gran utilidad es `train_test_split` que nos permitirá separar los datos de nuestra matriz de características `X` y nuestro vector de etiquetas `y` en submatrices con las que podemos trabajar nuestros sets de entrenamiento, validación y test. El parámetro `test_size` debe contener un valor entre cero y uno, representará la fracción de datos a separar. El parámetro `random_state` recibe números enteros, y actúa como semilla aleatoria para inicializar la separación.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=ts, random_state=num)
```

6.1.2 Pillow

La librería *Pillow* es bastante popular y simple para el manejo de imágenes en *Python*.

- `img = Image.open(filename)`: carga una imagen utilizando la librería *Pillow* con nombre `filename`.
- `img = Image.fromarray(nparray)`: transforma un arreglo de *numpy* en una imagen de *Pillow*.

6.1.3 OpenCV

La librería *OpenCV* es un software libre desarrollado originalmente por Intel con fines de uso para visión por computador. Se trata de una librería más sofisticada, que utilizaremos principalmente para el trabajo con videos.

- `img = cv2.imread(filename)`: carga una imagen utilizando la librería OpenCV. La imagen cargada queda en formato BGR.
- `img = cv2.line(img, (x1, y1), (x2, y2), color, thickness)`: dibuja una línea recta desde `(x1, y1)` hasta `(x2, y2)` con color (una tupla de 3 elementos de 0 a 255) y con grosor igual a `thickness`.
- `img = cv2.circle(img, (x, y), radius, color, thickness)`: dibuja un círculo en `(x, y)` con radio `radius`, color `color`, y con grosor igual a `thickness`.
- `img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`: transforma la imagen de espacio de color BGR a RGB.
- `video = cv2.VideoCapture(filename)`: crea un objeto video de *OpenCv*. Se pueden extraer las imágenes cuadro por cuadro al iterar sobre este objeto.
- `ret, frame = video.read()`: retorna una imagen de cuadro en el instante actual en `frame`. La imagen leída estará en formato BGR. El booleano `ret` será falso cuando no ha podido leer el cuadro.
- `fps = round(video.get(cv2.CAP_PROP_FPS))`: retorna un valor flotante del *framerate* en el que está grabado el video. Se debe redondear para obtener un valor útil de trabajo.
- `totalFrames = round(video.get(cv2.CAP_PROP_FRAME_COUNT))`: retorna un valor flotante de la cantidad de cuadros que tiene el video. Se debe redondear para obtener un valor útil de trabajo.

6.1.4 OpenPifPaf

Como esta es una librería desarrollada con aprendizaje profundo, para su uso utilizaremos las GPUs de Colab. El primer paso es su instalación:

```
!pip install openpifpaf==0.10.1
```

Los comandos de shell se declaran con un signo de exclamación en los jupyter notebooks.
Una vez instalada vamos a inicializar una red neuronal profunda. Como *OpenPifPaf* es una librería bien empaquetada haremos uso de unas cuantas funciones y métodos para trabajar con ella. Antes de eso, debemos cargar la red:

```
import torch
use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
print("usando " + "GPU" if use_cuda else "CPU")
oppnet, _ = openpifpaf.network.factory(checkpoint='resnet50')
oppnet = oppnet.to(device)
decode = openpifpaf.decoder.factory_decode(oppnet,
                                           seed_threshold=0.5)
processor = openpifpaf.decoder.Processor(oppnet, decode, #detector que utilizaremos
                                         instance_threshold=0.2,
                                         keypoint_threshold=0.3)
```

- `data = openpifpaf.datasets.PilImageList(lista)`: prepara un data set a partir de una lista de imágenes para *PifPaf*. Cada imagen en la lista debe ser un objeto inicializado con *Pillow*.
- `loader = torch.utils.data.DataLoader(data, batch_size=1, pin_memory=True)`: Utiliza *PyTorch* para crear una clase que transformará las imgágenes a tensores.
- `fields_batch = processor.fields(images_batch)`: Genera un mapa de gradientes de nombre `fields_batch` desde `images_batch`. Recurre a la clase inicializada `processor`. La variable `images_batch` proviene de `loader` al iterar con un bucle `for`. Ej:

```
for images_batch, _, __ in loader:
    #debemos enviar las imagenes a la GPU antes de procesarlas
    images_batch = images_batch.cuda()
    #Aca abajo podemos seguir con el resto de nuestro codigo
```

- `keypoint_sets, scores = processor.keypoint_sets(fields_batch[0])`: extrae los puntos de los esqueletos encontrados en una imágen como una lista de arreglos `keypoint_sets` con un nivel de confianza `score`. El tensor `keypoint_sets[n][17][3]`, o matriz de matrices, posee tres dimensiones. La primera corresponde a la cantidad de matrices de esqueletos encontrados, aquí `n`, mientras que las otras dos dimensiones ya han sido discutidas (ver arriba).
- `keypoint_painter = openpifpaf.show.KeypointPainter(color_connections=True, linewidth=2)`: Prepara un objeto `keypoint_painter` para desplegar las imágenes con el esqueleto correspondiente dibujado en ellas.
- `predictions = processor.annotations(fields_batch[0])`: Prepara un objeto de predicciones para la imagen comentada con mapa de gradientes `fields_batch`. Tal imágen se puede desplegar utilizando:

```
with openpifpaf.show.image_canvas(im) as ax:
  keypoint_painter.annotations(ax, predictions)
```

6.2 Preguntas

Utilizando las funciones anteriormente descritas programe un *jupyter notebook* utilizando la herramienta de Google Colab para entrenar y probar la implementación de un SVM. Se le proveerá algo de código extra en algunas partes para facilitar el trabajo:

1. Visite la dirección web de donde descargaremos el set de datos `http://vision.stanford.edu/Datasets/OlympicSports/`. Obtenga los links de descarga para las clases [*hammer*, *discus*, *shot_put*, *long_jump*]. Utilice el comando de bash `wget` para descargar cada archivo y el comando `unzip` para descomprimir los archivos. A continuación haga uso del script visto en la ayudantía para transformar los archivos desde el formato *.seq* al formato *.mp4* con cada uno de los archivos de la carpeta. Despliegue un corto clip de video de cada clase utilizando la función `to_gif` provista en los códigos. Señale correctamente a qué clase corresponde cada gif.

2. Utilizando los mismos videos seleccionados desarrolle un script que los lea con *OpenCV*. Seleccione al azar tres cuadros dentro del video: uno del primer tercio del video, otro del segundo tercio y otro del último tercio. Transforme cada cuadro desde el formato BGR al formato RGB. La interfaz actual de *OpenCV* guarda las imágenes de forma nativa en arreglos de numpy. Utilice el método `fromarray` de *Pillow* para convertirlas en imágenes de esa librería. Despliegue dichas imágenes usando el objeto `keypoint_painter` como se mencionó arriba.

3. Haga una reflexión sobre cuál cree que será el comportamiento del clasificador usado en esta tarea dadas las condiciones de cada deporte. Dé razones por las cuales podría identificar correctamente a las personas y cuáles podrían ser posibles limitantes. Refierase también al efecto entre la diferencia/similitud entre las imágenes en los tres momentos del video.

4. Cargue cada elemento de video obtenido. Extraiga tres imágenes al azar de cada video, ahora utilice las funciones de *OpenPifPaf* para obtener la matriz `keypoint_sets[n][17][3]`. Tenga en cuenta que las imágenes pueden contener a más de una persona (o a veces ninguna) dependiendo de la captura del video.

5. Indentifique al deportista. Por ende debe existir, como máximo, un "esqueleto" por imagen. En caso de que las tres imágenes no contengan personas, probar con otras tres imágenes al azar en la pregunta 2. Si el problema persiste indicarlo y comunicar al ayudante. Desarrolle algún criterio para identificar y filtrar la matriz (del "esqueleto") correcta y explíquelo. Además, despliegue las imágenes en las que está trabajando y marque el "esqueleto" elegido. Se solicita que al menos la mitad de las imágenes cuenten con un deportista seleccionado, si no le logra identificar al deportista pero en la pregunta anterior tampoco se hizo, esta imagen no entra en el conteo.

6. Haga otro script donde se elijan imágenes que tengan al deportista y al público o staff. Así como identificó al deportista en la pregunta anterior, haga lo mismo identificando a la persona (no el deportista) que esté más cercano al deportista. De hipótesis de las razones del comportamiento del reconocimiento , de respaldo (en código o argumentación de por qué debería ser así).

7. Transforme las matrices de puntos de esqueletos de las tres imágenes (donde se identificó a deportistas) en una única fila que represente una muestra de cada video. Defina un valor de clase para cada deporte. Concatene cada muestra o fila en una matriz de características `X`, sobre esa posición concatene el valor de la clase correspondiente a ese video en el vector de etiquetas `y`. Luego, desarrolle un criterio de separación de la matriz `X` y el vector `y` en tres submatrices para tener los sets de entrenamiento, validación y test. Utilice la función `train_test_split` para hacer las separaciones correspondientes. Justifique su elección de porcentajes en base a la teoría aprendida en clases, documentacíon y lo que ha aprendido del set de datos.

8. Entrene un clasificador SVM valores de slack $C = [0.1, 0.6, 1, 3, 7]$, y kernels = [$'linear'$, $'poly'$, $'rbf'$, $'sigmoid'$]. Explique cómo cada uno de los kernels clasifican los valores de forma teórica (sea breve). Reporte el valor de exactitud $accuracy$ para cada par de valores de hiperparámetros ordenándolos en una tabla con entradas $kernels/slack$. Para esto utilice el set de validación para medir exactitud. Reporte la matriz de confusión normalizada para el mejor par de hiperparámetros según se presenta en la tabla. Explique la matriz de confusión y las posibles razones del valor de exactitud ($accuracy$) ¿Tiene alguna relación con las posibles razones que indicó en la pregunta 3?

9. Reporte la exactitud y la matriz de confusión sobre el set de test. Analice la matriz de confusión reportada, ¿Que clase (deporte) es el que cuenta con mejor predicción? ¿Qué diferencia la posición de los deportistas en ese deporte frente a los demás y en qué afecta eso en el actuar del clasificador?

10. Analice las dos matrices de confusión obtenidas en los pasos anteriores junto a los valores de precisión del set de entrenamiento ¿Qué puede concluir sobre la relación y parentesco entre los set de entrenamiento, validación y testeo? ¿Qué opina sobre el posible rendimiento de su programa si trabajara con un dataset más grande?

El entregable es un archivo *jupyter notebook* (de extensión ".ipynb") con su nombre y apellido, número de alumno, y el desarrollo de la actividad. Fijese que la opción "Omitir el resultado de las celdas al guardar este notebook" está deshabilitada y que la aceleración de hardware está configurada en "GPU". Puede hacer esto en el menú Edición⇒ Configucación del notebook. No se corregirán los notebooks que no tengan los resultados disponibles.
Se les pide encarecidamente que agreguen comentarios a lo largo del código, así como párrafos conectivos y explicativos del desarrollo de cada pregunta. Además, las funciones y variables creadas por ustedes deben tener nombres autoexplicativos para facilitar la compresión del código.

References

[1] S. Kreiss, L. Bertoni, A. Alahi, *PifPaf: Composite Fields for Human Pose Estimation*, CVPR, 2019.