Informe Modelo CNN para identificación de acciones humanas

El reconocimiento de actividades humanas, o HAR por sus siglas en inglés, es un campo amplio de estudio que se ocupa de identificar el movimiento o la acción específica de una persona basándose en datos de sensores. Los movimientos suelen ser actividades típicas realizadas en interiores, como caminar, hablar, estar de pie y sentarse.

¿Por qué es importante?

El reconocimiento de actividades humanas desempeña un papel significativo en la interacción entre personas y en las relaciones interpersonales. Dado que proporciona información sobre la identidad de una persona, su personalidad y estado psicológico, es difícil de extraer. La capacidad humana de reconocer las actividades de otra persona es uno de los principales temas de estudio de las áreas científicas de visión por computadora y aprendizaje automático. Como resultado de esta investigación, muchas aplicaciones, incluyendo sistemas de videovigilancia, interacción humano-computadora y robótica para la caracterización del comportamiento humano, requieren un sistema de reconocimiento de múltiples actividades.

Objetivo

El objetivo del modelo es dada una imagen de entrada determinar la acción presente dentro de la misma.

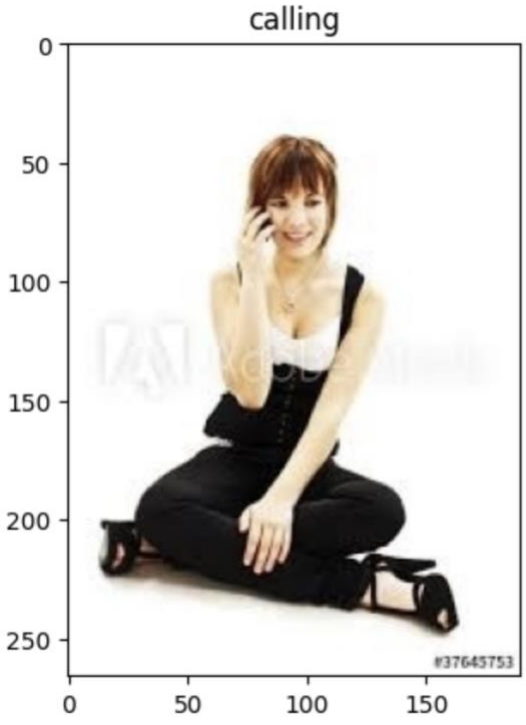
Datos

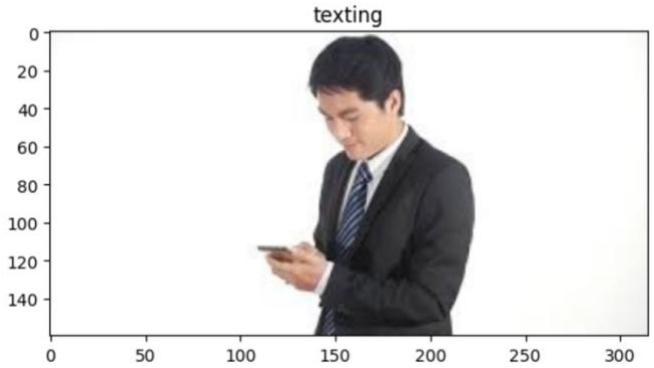
El data set usado es de Kaggle, contenido en el siguiente link: https://www.kaggle.com/datasets/meetnagadia/human-action-recognition-har-dataset

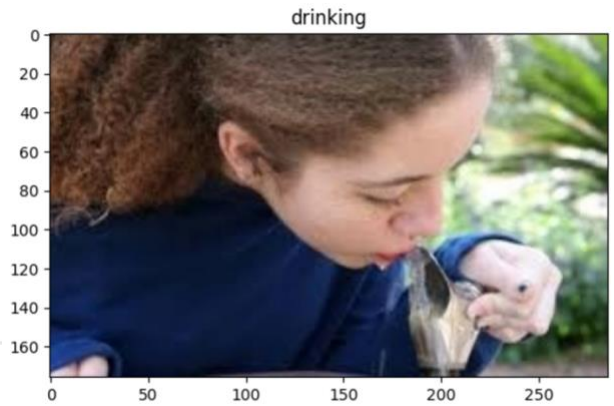
Train data size: 12600 archivos
Test data size: 5400 archivos
Clases: 15 clases de acciones para clasficar.
Distribución: 840 imágenes por cada acción
Tamaño en disco: 311 MB.

Algunos ejemplos


drinking
0
25
50
75
100
125
150
175
0
50
100
150
200
250


calling
0
50
100
150
200
250
0
50
100
150


texting
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300


drinking
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

Modelo

Para afrontar el problema se utilizó un modelo CNN secuencial, donde se hace uso de un modelo pre-entrenado llamado EficientNetB7. Este modelo es conocido por su rendimiento y eficiencia, combinando alta exactitud con bajo poder computacional.

Características Clave de EfficientNetB7

Escalabilidad: Los modelos EfficientNet escalan todas las dimensiones de la red (profundidad, ancho y resolución) de manera sistemática utilizando un método de escalado compuesto. EfficientNetB7 es el modelo más grande de la familia EfficientNet, ofreciendo una mayor precisión pero requiriendo más recursos computacionales.

Rendimiento: EfficientNetB7 alcanza una alta precisión en conjuntos de datos de referencia como ImageNet. Es uno de los modelos de mejor rendimiento en términos de precisión, manteniendo un costo computacional relativamente bajo en comparación con otros modelos de última generación.

Arquitectura:

- Escalado Compuesto: EfficientNet utiliza un método de escalado compuesto que escala uniformemente el ancho, la profundidad y la resolución de la red.
- Bloques MBConv: Utiliza bloques de Convolución Invertida de Cuello de Botella Móvil (MBConv), que son eficientes y efectivos para capturar características.

Pesos Pre-entrenados: EfficientNetB7 viene con pesos pre-entrenados en ImageNet, lo que lo hace adecuado para el aprendizaje por transferencia. Utilizar estos pesos pre-entrenados puede acelerar significativamente el proceso de entrenamiento y mejorar el rendimiento del modelo en tu conjunto de datos.

Tamaño del Modelo: EfficientNetB7 es la variante más grande en la familia EfficientNet, con 66 millones de parámetros. Requiere más memoria y potencia computacional en comparación con variantes más pequeñas como EfficientNetB0 o B1.

El modelo resumen del modelo se encuentra a continuación:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb7 (Functional  (None, 2560)              64097687
 )

 flatten (Flatten)           (None, 2560)              0

 dense (Dense)               (None, 256)               655616

 dense_1 (Dense)             (None, 15)                3855

=================================================================
Total params: 64757158 (247.03 MB)
Trainable params: 659471 (2.52 MB)
Non-trainable params: 64097687 (244.51 MB)
_________________________________________________________________
```

Pre-Procesado de datos

En la primera iteración solo se hizo resize de los datos. Se normalizaron las imágenes a 240x240 pixeles. Esto se divide entre el X_train (array de las imágenes preprocesadas) y Y_train (los respectivos labels).

En el resultado de esta prueba se evidencia overfitting con los datos de prueba ya que se obtiene una precisión de cerca del 98% pero muchas de las imágenes test salen erróneas lo que nos advierte una mala generalización por parte del modelo.

Para otra iteración del mismo grupo de training se crean dos generadores con subset de training y otro de validación, con el que se puede disminuir el overfitting y mejorar el rendimiento del modelo con las imágenes de prueba. Además, se incluyen atributos de rotación, width shift, height_shift y flip horizontal.

```
datagen = ImageDataGenerator(validation_split=0.2,
                             rotation_range=20,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             horizontal_flip=True)
```

```
[ ]  train_generator = datagen.flow(iii, y_train, subset='training')
     validation_generator = datagen.flow(iii, y_train, subset='validation')
```

Entrenamiento del modelo

Se decide hacer fit del modelo con las capas del modelo pre-entrenado congeladas, y usar las mismas capas de la primera iteración, pero con atributos extras como la validation_data, steps_per_epochs y validation_steps.

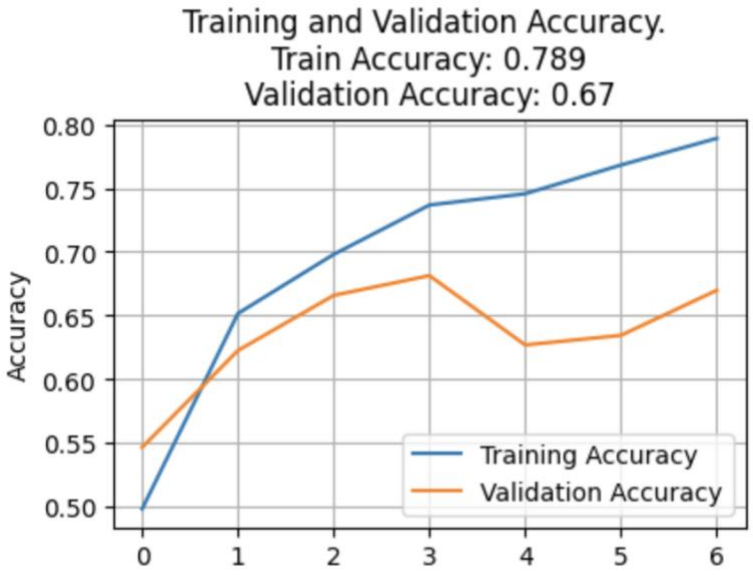
Además, se redujo el tamaño de las imágenes para lograr que procesen más rápido porque aun haciendo uso de aceleradores de hardware, cada epoch tenía una duración aproximada de 5 minutos.

Luego de estos primeros 10 epochs, se hacen entrenables las ultimas capas del modelo, y se vuelve a hacer fit. Cada epoch toma un tiempo mayor que los primeros 10, pero la diferencia de precisión entre los datos de prueba y los datos test disminuye bastante.

Se evidencia un aumento en el consumo de RAM en el momento de hacer el fine tunning, al punto de sacar un error por total de memoria usada durante algunas iteraciones.

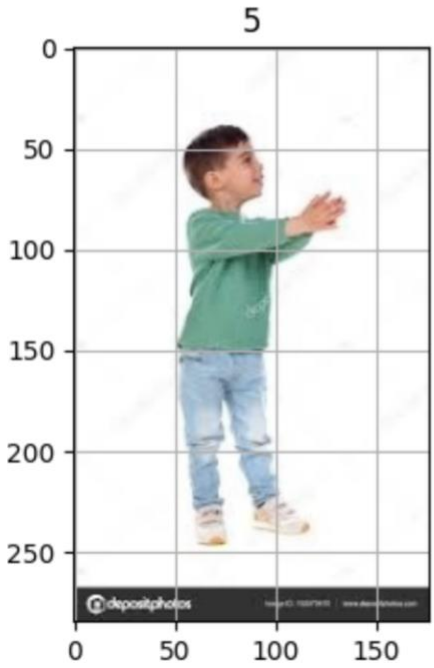
Después de este proceso se obtienen los siguientes resultados:

```
loss: 0.6544 - accuracy: 0.7890 - val_loss: 1.1617 - val_accuracy: 0.6698
```



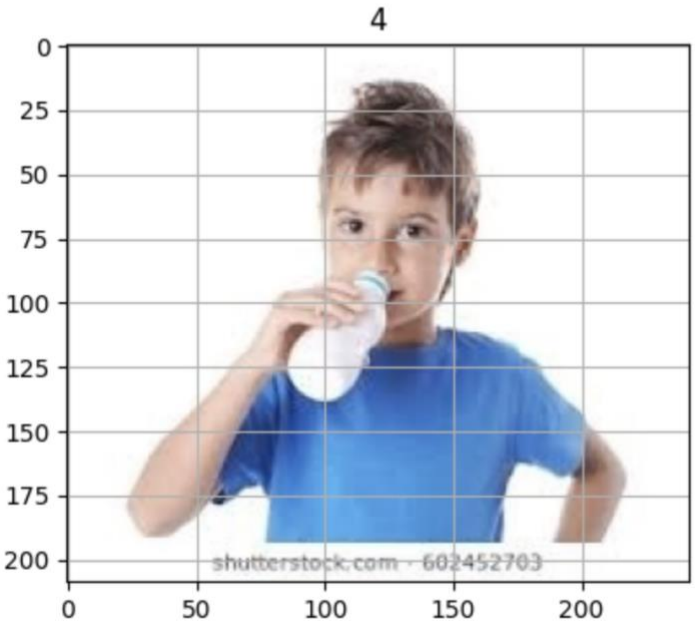
Algunos resultados:

```
1/1 [==============================] - 6s 6s/step
probability: 32.96618461608887%
Predicted class :  clapping
```

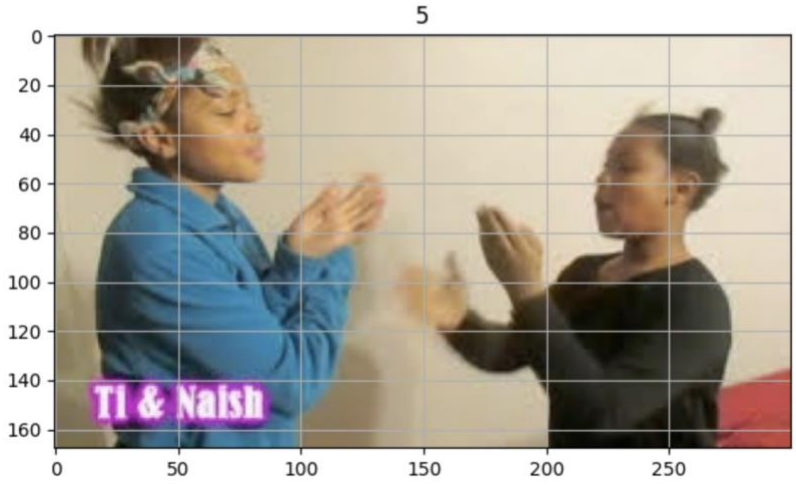


Variables

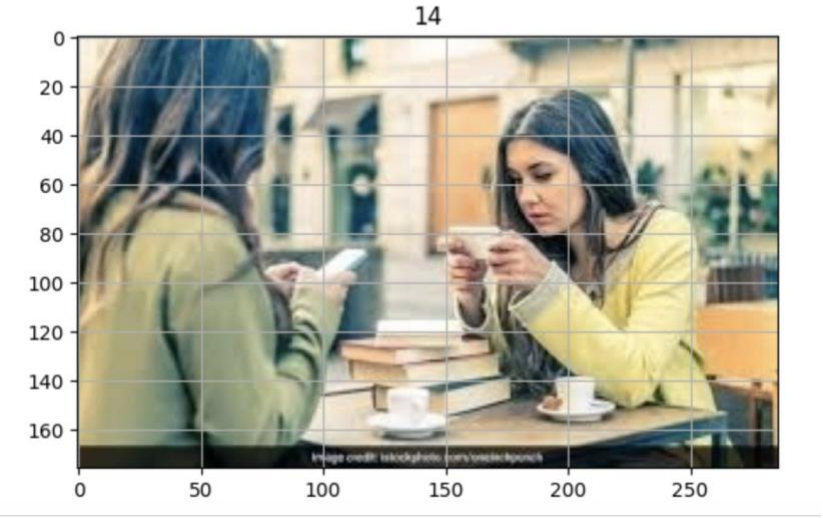
```
1/1 [==============================] - 0s 44ms/step
probability: 79.1300356388092%
Predicted class :  drinking
```




```
1/1 [==============================] - 0s 47ms/step
probability: 94.33802366256714%
Predicted class :  clapping
```



```
1/1 [==============================] - 0s 47ms/step
probability: 50.872933864593506%
Predicted class :  texting
```



Algunas posibles mejoras:

El modelo a pesar de tener una precisión menor parece estar mejor generalizado. Se considera satisfactorio el resultado, se adjunta una función para elegir una imagen aleatoria dentro del grupo de prueba.

Al ver las gráficas de la pérdida y la precisión se puede evidenciar que aumentar algunos epochs puede mejorar el modelo siempre y cuando se tenga el poder de cómputo disponible. También se pueden aumentar el número de capas que se descongelan para hacer el fine tunning.

Evitar el overfitting es esencial para obtener una buena generalización del modelo, por lo que también se pueden agregar más parámetros al preprocesado de datos.