Módulo 2 Implementación de un modelo de deep learning

Natalia Duarte Corzo - A01745482

Introducción

A lo largo de los años, deep learning ha tenido una expansión notable, utilizándose cada vez más con el fin de solucionar problemas de nuestra vida cotidiana y de esta manera optimizar y transformar nuestras vidas. A través de los modelos de deep learning podemos procesar datos de diversas áreas e identificar patrones o características en ellos, que nos ayuden a la resolver lo deseado.

Así pues, para esta actividad, seleccioné una base de datos de señalamientos de manos, en este caso, de las señales piedra, papel o tijera, que se utilizan para jugar, con el fin de generar un modelo capaz de identificar que señalamiento se esta realizando. Se pensó que este modelo podría después aplicarse al lenguaje de señas, con el fin de que identifiqué qué letras están generando con las manos y, de está manera identificar qué se está queriendo decir.Cabe mencionar, que esto se intentó realizar inicialmente pero, se tuviero limitaciones por la capacidad del RAM

```
#Importación de librerías
import pandas as pd
import numpy as np
import zipfile
import os
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

Carga y visualización de datos

Primeramente, se realiza la carga de datos, de una base seleccionada de la plataforma de kaggle, con el fin de identificar la calidad de imágenes y, las tres distintas categorías que existen,

siendo estas piedra, papel o tijera. Cabe mencionar que, todas las imágenes tienen el tamaño de 300x300, lo cuál será útil en los siguientes pasos.

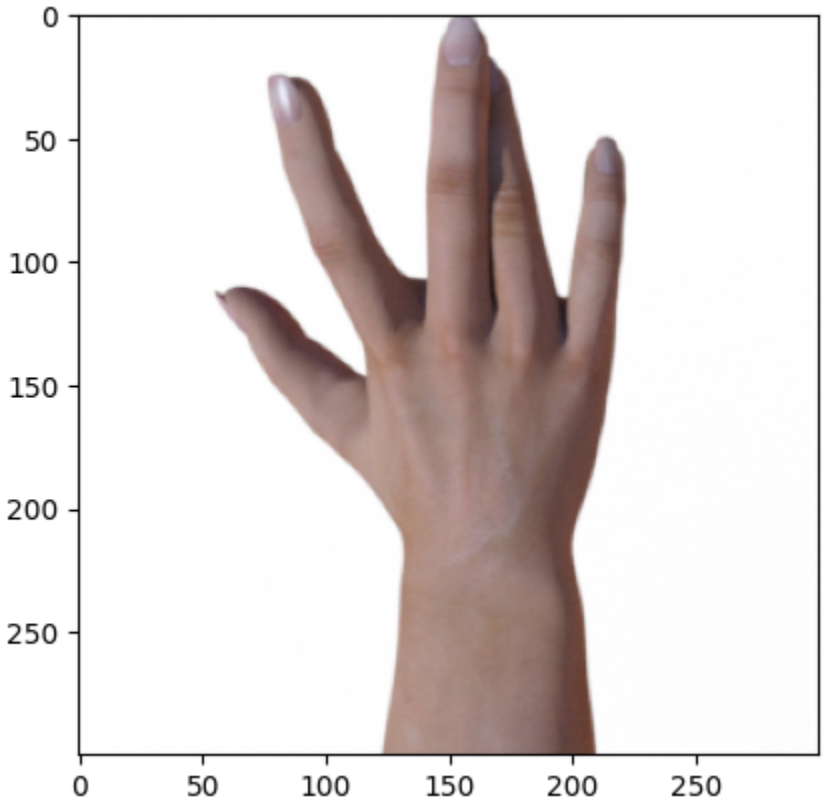
```
!mkdir -p /root/.kaggle
!cp '/content/drive/MyDrive/Semestre 7/Blumenkron/kaggle.json' /root/.kaggle/
```

```
!kaggle datasets download -d sanikamal/rock-paper-scissors-dataset
path = '/content/rock-paper-scissors-dataset.zip'
with zipfile.ZipFile(path,'r')as zip_ref:
  zip_ref.extractall('/content/')
```

```
rock-paper-scissors-dataset.zip: Skipping, found more recently modified local
```

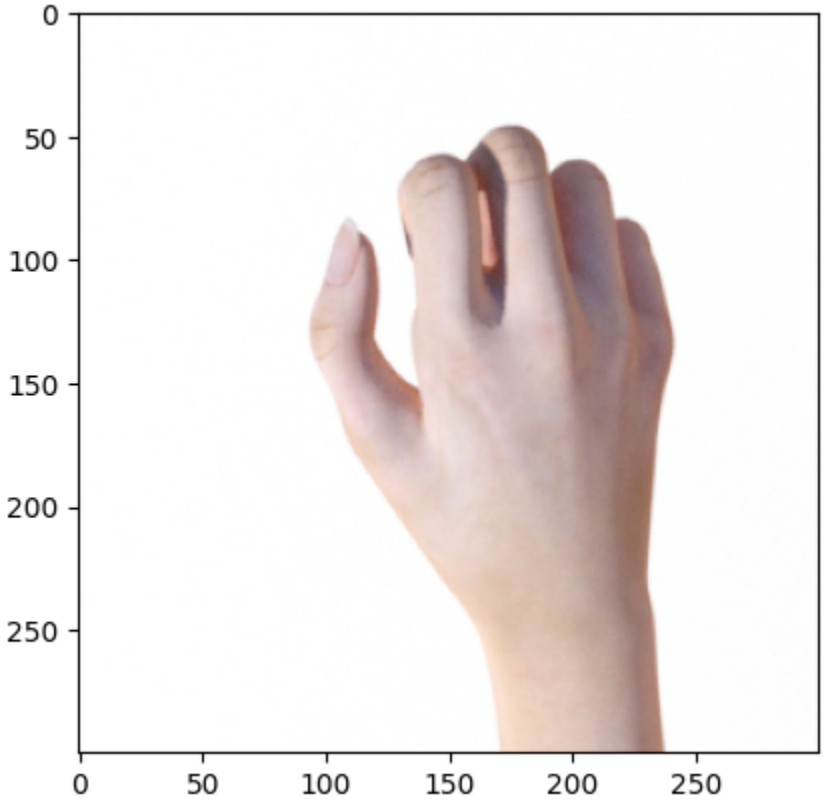
```
imagen = '/content/rock-paper-scissors/Rock-Paper-Scissors/train/paper/paper01-000.
img = mpimg.imread(imagen)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f8fdce62c50>
```



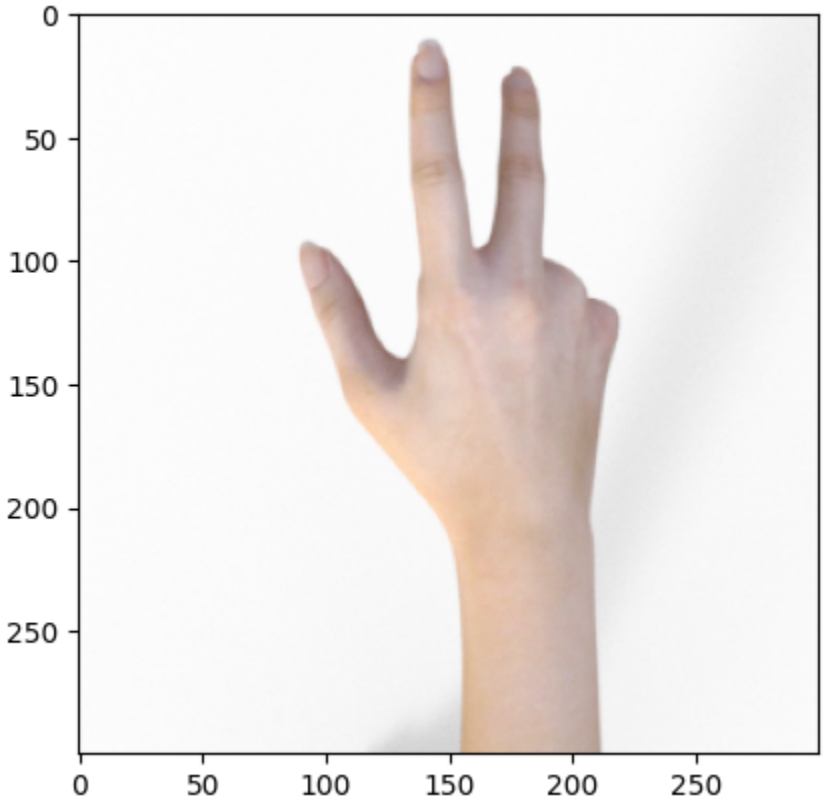

```
imagen = '/content/rock-paper-scissors/Rock-Paper-Scissors/train/rock/rock01-000.pr
img = mpimg.imread(imagen)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f8ffe066410>
```




```
imagen = '/content/rock-paper-scissors/Rock-Paper-Scissors/train/scissors/scissors0
img = mpimg.imread(imagen)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f8fdcdf2e00>
```



Creación de etiquetas

Un detalle que se identificó en esta base es que los datos de validación estaban en una sola carpeta, sin separarse por categoría, por lo que se hace un código para juntar las imagenes en su clase correspondiente (piedra, papel o tijera), con el fin de identificar cuántas imágenes se tiene en cada una de ellas

```
#Importamos las librerías que sirven para trabajar sobre directorios y manipular ar
import os
import shutil

directorio = '/content/rock-paper-scissors/Rock-Paper-Scissors/validation'
imagenes = os.listdir(directorio)#lectura de los nombres de las imágenes, pasamos t
categorias = ['rock', 'paper', 'scissors']#definimos las categorías como con los mi

for nombre in imagenes:
    image_path = os.path.join(directorio, nombre)
    if os.path.isfile(image_path):
        for categoria in categorias:
            if categoria in nombre:
                categoria_directorio = os.path.join(directorio, categoria)
                if not os.path.exists(categoria_directorio):
                    os.makedirs(categoria_directorio)
                shutil.move(image_path, os.path.join(categoria_directorio, nombre))
    else:
        print(f"{nombre} is not a file.")
```

```
paper is not a file.
rock is not a file.
scissors is not a file.
```

A continuación se utiliza ImageGenerator, de keras, para crear los generadores de datos para cada conjunto, siendo estos el conjunto de entrenamiento, de prueba y, de validación. Así pues, lo que se hace es cargar las imágenes desde su directorio, redimensionarlas y normalizarlas, para después ordenarlas por lotes y poder hacer el entrenamiento del modelo

```
#Hacemos el generador, el cual va a normalizar los píxeles de las imágenes escalánd
datagen = ImageDataGenerator(rescale=1./255)

#Definimos el tamaño del lote y, de las imagenes
batch size = 32
```

```
_
img_height, img_width = 300, 300  #tomamos en cuenta el tamaño de imagenes que se c

#Se crean generadores para cargar nuestras imágenes por lotes para los datos de ent
train_generator = datagen.flow_from_directory(
    '/content/rock-paper-scissors/Rock-Paper-Scissors/train',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'  #Definimos que es un problema de clasificación
)

datagen = ImageDataGenerator(rescale=1./255)

#Se crean generadores para cargar nuestras imágenes por lotes para los datos de pru
test_generator = datagen.flow_from_directory(
    '/content/rock-paper-scissors/Rock-Paper-Scissors/test',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)


datagen = ImageDataGenerator(rescale=1./255)

#Se crean generadores para cargar nuestras imágenes por lotes para los datos de val
val_generator = datagen.flow_from_directory(
    '/content/rock-paper-scissors/Rock-Paper-Scissors/validation',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)

# Puedes utilizar el generador directamente en model.fit()
#model.fit(train_generator, epochs=10)
```

```
Found 2520 images belonging to 3 classes.
Found 372 images belonging to 3 classes.
Found 33 images belonging to 3 classes.
```

CNN

Para esta actividad se decidió uitilzar redes convolucionales, que sirven para procesar las imágenes, sin tener que definir de manera manual los filtroso realizar operaciones manuales. Para lograr esto se utiliza de la librería de keras: keras.layers.Conv2D, con el fin de poder definir nuestras capas y, entrenarlas como parte de nuestra red.

Primer modelo

Primer modelo

El primer modelo, tiene 3 capas convolucionales, en las cuales vamos aumentando el tamaños de su filtro inciando con 16 y, duplicandolo hasta llegar a 64, lo cual nos ayuda a identifica características de nuestras imágenes. Así mismo, en utilizamos MaxPooling2D con el fin de reducir la dimensionalidad de las características identificadas y, utilizamos 'relu' como nuestra función de activaciión. Como se puede observar más adelante, se utiliza Flatten para pasar la dimensionalidad de los mapas generados de 2D a 1D.

Finalmente, antes de entrenar al modelo se debe de compilar, para lo que usa el optimizador adam, la función de pérdida 'categorical_crossentropy', debido a que nuestro problema es de clasificación multiclase y definimos que queremos usar la precisión para ir evaluando el modelo mientras lo entrenamos.

```
model = Sequential([
    Conv2D(16, kernel_size=3, padding='same', activation="relu", input_shape=(img_w
    MaxPooling2D(2),
    Conv2D(32, kernel_size=3, padding='same', activation="relu"),
    MaxPooling2D(2),
    Conv2D(64, kernel_size=3, padding='same', activation="relu"),
    MaxPooling2D(2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
```

El siguiente paso, es hacer el entrenamiento de nuestro primero modelo, en el cual, como se observa usamos los datos de entrenamiento y los de validación para evaluar el modelo y, definimos que habrán 10 épocas (el modelo "pasa" 10 veces sobre los datos para entrenarse).

```
epochs = 10
history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=val_generator
)
```

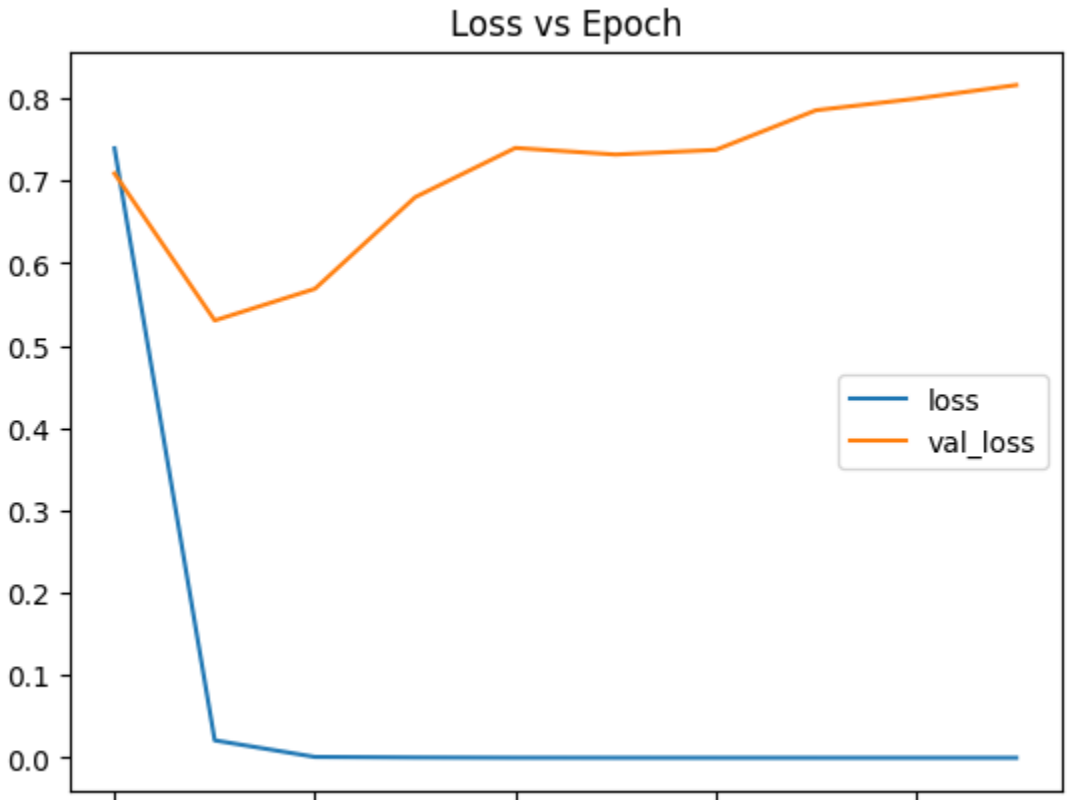
```
Epoch 1/10
79/79 [==============================] - 518s 6s/step - loss: 0.7391 - accurac
Epoch 2/10
79/79 [==============================] - 388s 5s/step - loss: 0.0212 - accurac
```

```
Epoch 3/10
79/79 [==============================] - 392s 5s/step - loss: 0.0011 - accura
Epoch 4/10
79/79 [==============================] - 350s 4s/step - loss: 4.5034e-04 - ac
Epoch 5/10
79/79 [==============================] - 381s 5s/step - loss: 1.4255e-04 - ac
Epoch 6/10
79/79 [==============================] - 360s 5s/step - loss: 8.1349e-05 - ac
Epoch 7/10
79/79 [==============================] - 328s 4s/step - loss: 4.9357e-05 - ac
Epoch 8/10
79/79 [==============================] - 325s 4s/step - loss: 3.4139e-05 - ac
Epoch 9/10
79/79 [==============================] - 326s 4s/step - loss: 2.5478e-05 - ac
Epoch 10/10
79/79 [==============================] - 323s 4s/step - loss: 1.9004e-05 - ac
```

Como se puede observar a continuación, evaluamos la precisión de nuestro modelo, la cual, estuvo arriba de 70% en los datos de validación, lo cual es un buen resultado, pero, aún tiene áreas de oportunidad para mejorar la precisión. No obstante, dado que para los datos de entrenamientos obtenemos un accuracy del 100%, puede que estemos teniendo un overfitting.

```
# Load the history into a pandas Dataframe
df = pd.DataFrame(history.history)
# Make a plot for the loss
df.plot(y=["loss", "val_loss"], title="Loss vs Epoch",ylim=(0,1))
```

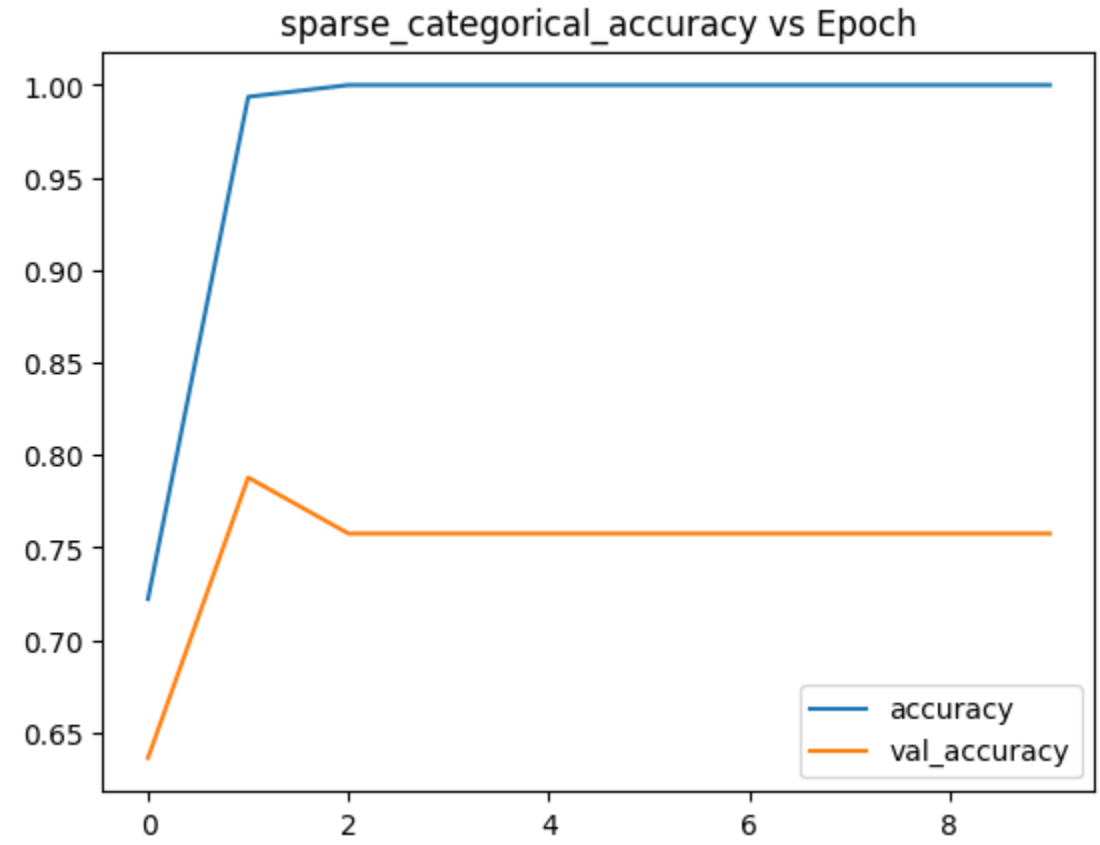
```
<Axes: title={'center': 'Loss vs Epoch'}>
```



```
0  2  4  6  8
```

```
# Make a plot for the accuracy
df.plot(y=["accuracy", "val_accuracy"], title="sparse_categorical_accuracy vs Epoch
```

```
<Axes: title={'center': 'sparse_categorical_accuracy vs Epoch'}>
```



Modelo mejorado

Primeramente, para crear crear un mejor modelo, tomé como primer acercamiento el añadir más capas. En total, como se puede observar hay tres capas convolucionales con MaxPooling, como en el modelo anterior se tiene una capa de flatten y, una capa densa, una de dropout, la cual tiene como función ayudar a evitar el overfitting de nuestros datos.

Por otro lado, se modificó el optimizador a RMSprop y, se cambió el learning rate a .0001, lo cual ayudará con la velocidad del entrenamiento de nuestros datos, que como vimos anteriormente toma mucho tiempo incluso con un modelo de baja complejidad.

Por último se añadió EarlyStopping, el cuál sirve que cuando se identifiqué que la pérdida del conjunto de validación no mejora, detendrá el entrenamiento y, tomará al mejor resultado.

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import RMSprop

model_improved = Sequential()
model_improved.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, im
model_improved.add(MaxPooling2D((2, 2)))
model_improved.add(Conv2D(64, (3, 3), activation='relu'))
model_improved.add(MaxPooling2D((2, 2)))
model_improved.add(Conv2D(128, (3, 3), activation='relu'))
model_improved.add(MaxPooling2D((2, 2)))
model_improved.add(Flatten())
model_improved.add(Dense(128, activation='relu'))
model_improved.add(Dropout(0.5))
model_improved.add(Dense(3, activation='softmax'))

optimizer = RMSprop(learning_rate=0.0001)

model_improved.compile(optimizer=optimizer, loss='categorical_crossentropy', metric

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights

history = model_improved.fit(train_generator, epochs=10, validation_data=val_genera
```

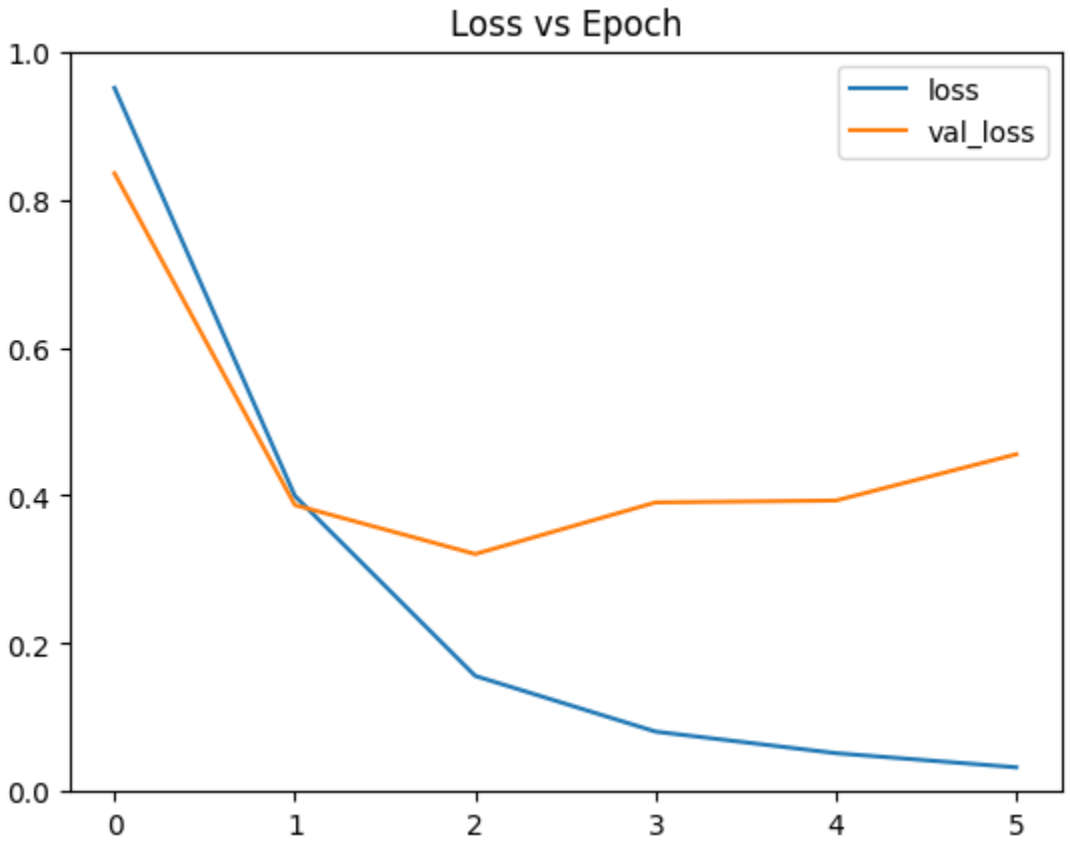
```
Epoch 1/10
79/79 [==============================] - 679s 9s/step - loss: 0.9507 - accura
Epoch 2/10
79/79 [==============================] - 667s 8s/step - loss: 0.3993 - accura
Epoch 3/10
79/79 [==============================] - 668s 8s/step - loss: 0.1561 - accura
Epoch 4/10
79/79 [==============================] - 672s 8s/step - loss: 0.0808 - accura
Epoch 5/10
79/79 [==============================] - 670s 9s/step - loss: 0.0517 - accura
Epoch 6/10
79/79 [==============================] - 670s 8s/step - loss: 0.0326 - accura
```

Como se puede observar en los resultados del segundo modelo, este nos da mejores resultados gracias a los cambios realizados. Primeramente, para los datos de entrenamiento, ya no se llego a una precisión del 100%, no obstante puede que, dado que es de 99% pueda seguir habiendo un overfitting. Por otro lado, la precisión de los datos de validación, tuvo resultados mejores, siendo la más alta de 87%

```
# Load the history into a pandas Dataframe
df = pd.DataFrame(history.history)
```

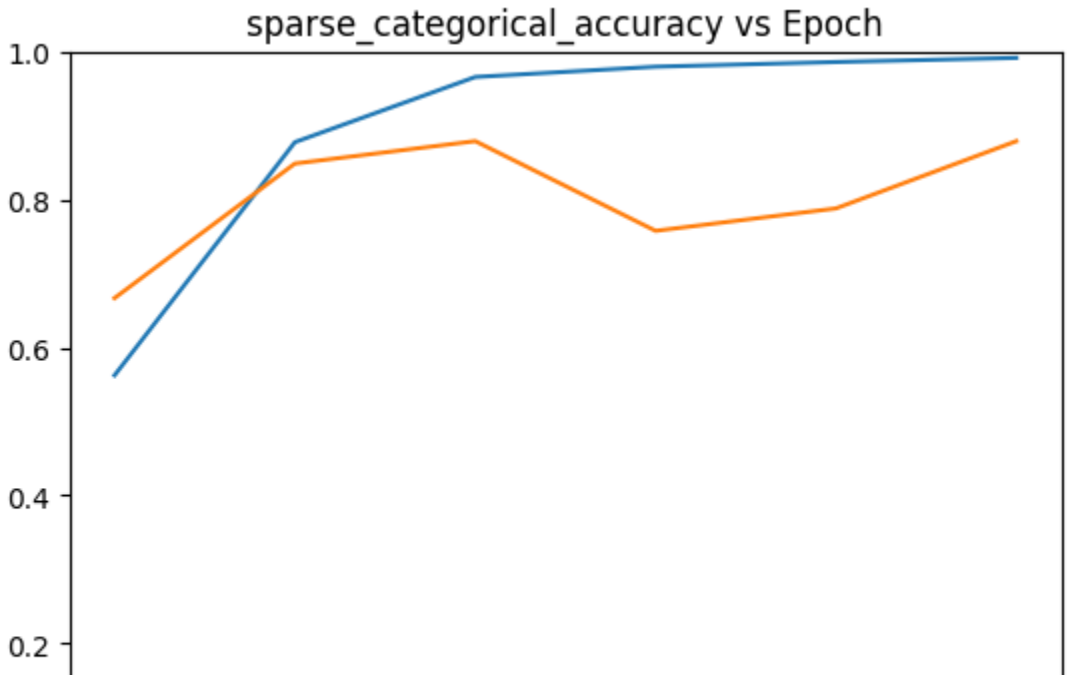
```
# Make a plot for the loss
df.plot(y=["loss", "val_loss"], title="Loss vs Epoch",ylim=(0,1))
```

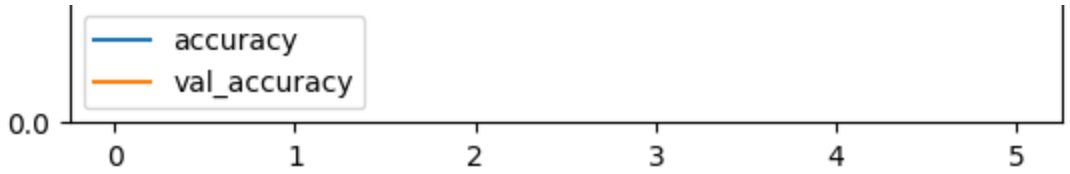
```
<Axes: title={'center': 'Loss vs Epoch'}>
```




```
# Make a plot for the accuracy
df.plot(y=["accuracy", "val_accuracy"], title="sparse_categorical_accuracy vs Epoch
```

```
<Axes: title={'center': 'sparse_categorical_accuracy vs Epoch'}>
```





Conclusiones

Después de completar esta actividad, se puede afirmar que se cumplió con el objetivo de desarrollar un modelo inicial para clasificar las imágenes de piedra, papel y tijera utilizando una red neuronal convolucional. Con base a los resultados, se trabajó en la creación de un modelo mejorado, el cual demostró una mejora al abordar problemas de sobreajuste con la implementación de early stopping e inclusión de una capa de dropout y, al mismo tiempo, mostró una mejor precisión gracias a las capas adicionales y el uso de un nuevo optimizador.

Este modelo puede tener una aplicación más realista como clasificación de signos del lenguaje de señas para la interpretación de las expresiones de las personas, lo cual mejoraría significativamente la comunicación con individuos con discapacidades en este ámbito. En esta actividad no se logró realizar esto, debido a la capacidad de la memoria RAM, que impidió hacer una clasificación de 27 clases, siendoe estas una para cada letra del abecedario.