

✓ Proyecto de Deep Learning

15/11/2024

Andre Galindo Posadas -> A00833376

Introducción

Se ha utilizado lo que es un dataset de la gente entrenando en un gimnasio, esto para predecir el nivel de experiencia de cada usuario, según sus tipos de entrenamiento. El dataset se sacó de Kaggle en la siguiente liga: <https://www.kaggle.com/datasets/valakhorasani/gym-members-exercise-dataset>.

La información original del dataset, antes de realizar la limpieza, incluía datos personales como edad, género, peso y estatura. Además, contenía diversas métricas relacionadas con el entrenamiento, como pulsaciones máximas, promedio de pulsaciones, pulsaciones en reposo, duración de las sesiones de entrenamiento en horas, calorías quemadas, tipo de entrenamiento, porcentaje de grasa corporal, ingesta de agua en litros, frecuencia semanal de entrenamiento, nivel de experiencia y el índice de masa corporal (IMC).

El objetivo es predecir la experiencia de los usuarios en el gimnasio, se seleccionaron únicamente las características relacionadas con su rendimiento y comportamiento durante el entrenamiento. Estas son pulsaciones máximas y en reposo, la duración de la sesión, el tipo de entrenamiento y la frecuencia semanal de ejercicio. Se aplicó técnicas de codificación, como también se normalizó los datos para que el modelo tuviera mejores predicciones.

El nivel de experiencia se clasifica en tres categorías: 1 (principiante), 2 (intermedio) y 3 (avanzado). Para el modelo, se utilizó una red neuronal de capas densas (Dense Layer) en una configuración de clasificación. Se ajustaron los hiperparámetros para optimizar el rendimiento del modelo, con el fin de obtener la mejor precisión posible en la predicción.

El modelo se utilizó en diferentes tipos de usuarios y darle una explicación a sus resultados y si son apegados a la realidad de sus entrenamientos. Por último se implementa un plan a futuro y como se puede adaptar en casos de la vida real y la utilidad que tiene este modelo.

Problema

Lo que se trata de analizar con el dataset es como las personas dividen sus entrenamientos y ver si lo hacen de manera efectiva, ya que se clasifican entre 3 niveles: 1 (Básico), 2 (Intermedio), 3 (Avanzado). Por lo que se trata de implementar un modelo de clasificación de Dense Layer que pueda guiar de cierta manera a las personas nuevas del gimnasio o que pueda brindar información sobre usuarios que necesiten ayuda a los entrenadores y ellos puedan guiar a estos nuevos usuarios, para que el gimnasio pueda identificar áreas de oportunidad y pueda dar un servicio más personalizado a cada miembro del mismo.

✓ Objetivos

- Crear un modelo de Dense Layer que sea eficiente y pueda realizar predicciones adecuadas
- Entender y poder diagnosticar el modelo de Dense Layer a través de diferentes pruebas de modelo
- Explicar los resultados de predicción del modelo con diferentes ejemplos

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.67.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.7)
```

```
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tens
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorf
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (

# 1 Implementación de librerías
# En esta parte del código se instalan las librerías necesarias para implementar los modelos
# y utilidades necesarias para que el código funcione adecuadamente
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import numpy as np
import pandas as pd
from pandas import ExcelWriter
import os
import time

# 2 Lectura del Dataset
# En esta parte del código, se lee la base de datos, se asignan las columnas y luego se enseña
# una pequeña cantidad de los datos
dataframe = pd.read_csv('gym_members_exercise_tracking.csv')
dataframe.columns = ['Age', 'Gender', 'Weight (kg)', 'Height (m)', 'Max_BPM', 'Avg_BPM',
                    'Resting_BPM', 'Session_Duration (hours)', 'Calories_Burned', 'Workout_Type', 'Fat_Percentage', 'Water_Intake (liters)', 'Workout_Frequency (days/week)', 'Experience_Level', 'BMI']
dataframe.head()
```

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	Fat_Percentage	Water_Intake (liters)	Workout_Frequency (days/week)	Experience_Level	BMI
0	56	Male	88.3	1.71	180	157	60	1.69	1313.0	Yoga	12.6	3.5			
1	46	Female	74.9	1.53	179	151	66	1.30	883.0	HIIT	33.9	2.1			
2	32	Female	68.1	1.66	167	122	54	1.11	677.0	Cardio	33.4	2.3			
3	25	Male	53.2	1.70	190	164	56	0.59	532.0	Strength	28.8	2.1			
4	38	Male	46.1	1.79	188	158	68	0.64	556.0	Strength	29.2	2.8			

Pasos siguientes:

Generar código con dataframe

☒ Ver gráficos recomendados

New interactive sheet

✓ Explicación de la limpieza del DataSet

En respecto el porque se escogieron los datos es por métricas de entrenamiento que ya se han utilizado anteriormente, una de las cosas más principales es las pulsaciones máximas de entrenamiento, ya que esto no indica realmente una experiencia alta o baja, pero si como responde el cuerpo cuando se le exige una prueba física. Un ejemplo de su uso es en los deportes de alto rendimiento, ya que mientras más tiempo puedas sostener una frecuencia máxima estas mejor preparado para el entrenamiento.

El tipo de entrenamiento también va a afectar a como responde el cuerpo ya que los requisitos que cada ejercicio son diferentes unos entre otros, pero estos pueden tener un efecto directo en como influye al cuerpo humano. Al final una persona que hace cardio puede tener diferentes pulsaciones y más elevadas que una persona que hace fuerza. Lo mismo influiría las calorías quemadas por entrenamiento, por ejemplo los HIIT y Yoga pueden quemar más que otros tipos de entrenamientos como la fuerza.

Ahora la frecuencia con las que se entrena tanto el tiempo activo como las veces que se realiza este entrenamiento a la semana puede influir, ya que las personas más capacitadas tienen un plan inteligente y de alta intensidad para que logren mantener por más tiempo su frecuencia máxima. Mientras que las nuevas personas sus cuerpos tiene que adaptarse a los estímulos de entrenamiento y a través de eso ir programando más intensidad.

Todo esto se une con los tiempos de descanso, ya que esta parte también es importante para prevenir lesiones, por lo que hay que entender también como esta el cuerpo en su momento de descanso y cuales son las pulsaciones normales sin exigencias deportivas

Todo esto unido son los datos que se utilizaron para que el modelo entrenara y pudiera predecir la experiencia de cada uno de sus usuarios. Se descartarán información como el género, edad y peso, ya que tratamos de medir la eficiencia del entrenamiento por lo que el peso no influye, ya que las personas que entrenan fuerza son personas más pesadas, y no por eso entrenan mal, junto con las edades ya que eso puede influir en las pulsaciones pero de nuevo todo esto se mide al momento de entrenar por lo que es algo que se puede ir descartando, en el caso de el género hay mucha variedad, ya que puede que estemos midiendo a mujeres de entrenamiento de alta intensidad y a hombres que son su primera vez, esto también puede pasar al revés, por lo que se descarta ese dato. Junto con otros que no afectaban directamente al entrenamiento

```
# 3 Limpieza del dataset
# En esta parte del código lo que se hace es escalar los datos y codificarlo, aparte
# de asignar lo que es los features y el label

# Se asignan las columnas que se van a escalar y la que se va a codificar
num_features = ['Max_BPM', 'Resting_BPM', 'Session_Duration (hours)', 'Calories_Burned', 'Workout_Frequency (days/week)']
cat_features = ['Workout_Type']
```

```
# Se asigna el label
y = dataframe['Experience_Level']

# Crear un scaler y ajustarlo a las características numéricas
scaler = StandardScaler()
# Se escalan los datos que anteriormente establecimos
X_num_scaled = scaler.fit_transform(dataframe[num_features])

# Convertir a DataFrame para fácil visualización y luego para concatenarlo con los
# datos codificados
X_num_scaled_df = pd.DataFrame(X_num_scaled, columns=num_features)

# Se crea el codificador de datos
encoder = OneHotEncoder(sparse_output=False)
# Se codifica los datos del tipo de entrenamiento
X_cat_encoded = encoder.fit_transform(dataframe[cat_features])

# Obtener nombres de las columnas de One-Hot Encoding
cat_feature_names = encoder.get_feature_names_out(cat_features)
# Luego se convierte en DataFrame los datos codificados
X_cat_encoded_df = pd.DataFrame(X_cat_encoded, columns=cat_feature_names)

# Luego se concatenan las dos columnas tanto los datos ya escalados
# como son los datos codificados, para enseñar lo que es el antes y después del Dataset
X_transformed_df = pd.concat([X_num_scaled_df, X_cat_encoded_df], axis=1)

# Mostrar un parte del Dataset ya limpió, como su forma
print(X_transformed_df.head())
print(X_transformed_df.shape)

# Dividir el dataset en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_transformed_df, y, test_size=0.2, random_state=42)

# Se cambia el Label para que los valores empiecen de 0 a 2
y_train = to_categorical(y_train - 1)
y_test = to_categorical(y_test - 1)
```

```

Max_BPM  Resting_BPM  Session_Duration (hours)  Calories_Burned  \
0  0.010081    -0.303555                1.264598        1.495690
1  -0.076726    0.515749                0.127098       -0.082284
2  -1.118414   -1.122858                -0.427068       -0.838243
3  0.878155    -0.849757               -1.943735       -1.370351
4  0.704540    0.788850                -1.797902       -1.282278

Workout_Frequency (days/week)  Workout_Type_Cardio  Workout_Type_HIIT  \
0          0.743295                0.0                0.0
1          0.743295                0.0                1.0
2          0.743295                1.0                0.0
3         -0.352502                0.0                0.0
4         -0.352502                0.0                0.0

Workout_Type_Strength  Workout_Type_Yoga
0          0.0          1.0
1          0.0          0.0
2          0.0          0.0
3          1.0          0.0
4          1.0          0.0
(973, 9)
```

✓ Justificación de la Escalación de los datos

El proceso de escalación de los datos se realizo, porque genera muchas ventajas a la hora de entrenar un modelo de Deep Learning. Algunos ejemplos e esto son:

- En este caso donde las características del entrenamiento varían en las métricas de los datos, ya que algunas cosas son RPM o calorías. se escala para que el modelo no le de importancia a una característica, para que no genere sesgos y realice medidas más exactas
- Los datos escalados ayudan a que el modelo pueda converger más rápido y de una manera más estable. En pocas palabras el modelo se calibra para que de respuestas más eficientes y exactas.
- Los datos al estar en un mismo rango, reduce la varianza, por lo que se apoya lo anterior y los modelos convergen más rápido
- Para ReLU, los valores cercanos a cero son ideales, ya que ayuda a mantener la activación sin saturación.

```
# 4 Creación del modelo
# Se crea una función donde se puede ir manipulando los hiperparámetros.
# Además como se establecen los hiperparametros si no se envían datos
def create_model(input_shape, optimizer='adam', neurons=64, num_layers=2):
    # Se crea un modelo de Dense Layer
    model = Sequential()

    # Se agrega la primera capa neuronal con los datos personalizados
    model.add(Dense(neurons, activation='relu', input_shape=(input_shape,)))

    # Se agregan las demás capas del modelo según lo establecido en las num_layer
    for _ in range(num_layers - 1):
        # Se agrega las capas intermedias
        model.add(Dense(neurons, activation='relu'))

    # Se agrega la última capa con la salida de las 3 categorías de experiencia de entrenamiento
```



```
# 6 Resultados de las pruebas del modelo
# En esta parte del código lo que se hizo fue verificar cual era el modelo más óptimo para poder
# diagnosticarlo y sacar la mejor precisión posible

# Ruta del archivo
archivo_excel = 'excel_resultados.xlsx'

# Verificar si el archivo ya existe
if os.path.exists(archivo_excel):
    # Si el archivo existe, lo cargamos
    df_excel = pd.read_excel(archivo_excel, sheet_name="Resultados")
else:
    # Si no existe, creamos un DataFrame vacío
    excel_resultados = {
        "learning_rate": [],
        "batch_size": [],
        "neurons": [],
        "epochs": [],
        "layers": [],
        "accuracy": [],
        "execution_time": []
    }
    df_excel = pd.DataFrame(excel_resultados)



# Agregar una nueva fila con los resultados actuales
df_excel.loc[len(df_excel)] = [lr, bz, neu, epo, nlay, accuracy * 100, timer]

# Guardar el DataFrame actualizado en el archivo Excel
df_excel.to_excel(archivo_excel, sheet_name="Resultados", index=False)

print("Nuevos datos agregados a excel_resultados.xlsx")
```

 Nuevos datos agregados a excel_resultados.xlsx

```
df_excel.head()
```

	learning_rate	batch_size	neurons	epochs	layers	accuracy	execution_time
0	0.001	64.0	128.0	65.0	2.0	84.615386	1.731541e+09
1	0.001	64.0	64.0	50.0	3.0	84.102565	1.731541e+09
2	0.001	32.0	32.0	35.0	5.0	88.717949	1.731541e+09

Pasos siguientes:

[Generar código con df_excel](#)

☒ [Ver gráficos recomendados](#)

[New interactive sheet](#)

✓ Explicación de resultados

- Configuración 1: En esta configuración, el modelo tiene 2 capas y cuenta con 128 neuronas por capa. Esta configuración logra una precisión del 84.61%, pero la menor profundidad limita su capacidad de capturar relaciones complejas en los datos. La combinación de un batch_size de 64 y pocas capas, genera un modelo que se estabiliza, pero que tiene una capacidad limitada de aprendizaje, por lo que sus resultados son los más bajos
- Configuración 2: En esta configuración se redujo las neuronas por capa siendo 64 por nivel, junto con un batch_size de 64, que sigue siendo sólido, pero siguen sin salir del porcentaje del 84.10% por lo que el resultado sigue siendo eficiente pero no logro aprender de manera eficiente, el hecho también de que hay varios epoch y a veces sube y vuleve a bajar puede ser que no este ayudando al modelo a entrenar.
- Configuración 3: En esta configuración que fue la mejor se aumentaron las capas profundas ya que fueron 5 y se usan menos neuronas. Se altero el batch_size para que el aprendizaje fuera más eficiente a la hora de actualizar los pesos, y se redujo el número de epoch para que no fuera redundante en los resultados finales. Al final este modelo tiene un equilibrio, entre la profundidad que tiene junto con el aprendizaje que genera

```
# 7 Prueba del modelo con diferentes usuarios
# En este código se establecieron datos de diferentes compañeros para saber el nivel
# de cada uno de ellos en el gimnasio y poder entender si los resultados tienen sentido

# Se establece los datos en el DataFrame de los diferentes usuarios
MyDF = pd.DataFrame({
    'Max_BPM': [170, 160, 120, 130],
    'Resting_BPM': [75, 60, 60, 70],
    'Session_Duration (hours)': [2, 1.5, 1.5, 1],
    'Calories_Burned': [1200, 700, 800, 600],
    'Workout_Type': ['Strength', 'HIIT', 'Strength', 'Strength'],
    'Workout_Frequency (days/week)': [6, 2, 4, 5]
})

# Se normalizan los datos establecidos de los usuarios
MyDF[['Max_BPM', 'Resting_BPM', 'Session_Duration (hours)', 'Calories_Burned',
    'Workout_Frequency (days/week)']] = scaler.transform(MyDF[['Max_BPM', 'Resting_BPM',
    'Session_Duration (hours)', 'Calories_Burned',
    'Workout_Frequency (days/week)']])

# Se codifica el tipo de entrenamiento de cada uno de los usuarios
workout_type_encoded = encoder.transform(MyDF[['Workout_Type']])
```

```
# Se quita la columna de entrenamiento anterior
MyDF = MyDF.drop('Workout_Type', axis=1)

# Se establece el dataframe con los nombres de las columnas codificadas
workout_type_encoded_df = pd.DataFrame(workout_type_encoded, columns=encoder.get_feature_names_out(['Workout_Type']))
# Y se concatenan los Dataframes normalizados y codificados
MyDF = pd.concat([MyDF, workout_type_encoded_df], axis=1)

# Se les da al modelo y se enseñan los resultados de predicción
prediction = model.predict(MyDF)
predicted_class = prediction.argmax(axis=1)

# Impresión de los resultados
print("Experiencia de Andre Galindo (Strength): ", predicted_class[0])
print("Experiencia de Raphael Chavéz (HIIT): ", predicted_class[1])
print("Experiencia de Raphael Chavéz (Strength): ", predicted_class[2])
print("Experiencia de Pablo Martínez (Strength): ", predicted_class[3])
```

1/1 ————— 0s 36ms/step
Experiencia de Andre Galindo (Strength): 2
Experiencia de Raphael Chavéz (HIIT): 0
Experiencia de Raphael Chavéz (Strength): 1
Experiencia de Pablo Martínez (Strength): 1

✓ Explicación de predicciones

- Estudio de Andre Galindo: En mi caso, llevo 4 años entrenando sabiendo aprender de todos mis entrenadores, por lo que en estos momentos entiendo como funciona mi cuerpo y como ir administrando mis entrenamientos, tanto de fuerza, como el balance que hago en el caso del crossfit, se como balancear mis entrenamientos y adaptarme a todo lo que hago, por lo que el tiempo que llevo entrenando, todo lo que he aprendido, junto con mis pulsaciones y frecuencias de entrenamiento se mantener mi frecuencia cardiaca por un tiempo más alto que el promedio, por lo que la predicción del modelo tiene sentido
- Estudio de Raphael Chavéz: En las dos predicciones donde el sujeto es mi compañero Raphael, tiene mucho sentido ya que las clases de HIIT las hace en clases extracurriculares por lo que solo puede tomar dos veces por semana, por lo que la baja frecuencia de entrenamiento si bien es muy baja la limitaciones de aprendizaje son muy claras por lo que esta en principiantes en ese aspecto por lo que tiene sentido, por otro lado la intensidad cambia cuando esta entrenando fuerza, ya que es más frecuencias y quema más calorías teniendo un poco menos en las pulsaciones máximas, y por eso lo asigna a la parte de intermedio.
- Estudio de Pablo Martínez: En el caso de mi compañero Pablo, tiene más frecuencia de entrenamiento, ya que lo hace 5 veces por semana, por menos tiempo, pero de manera más eficiente por lo que también lo pone en intermedios, mantiene una buena cantidad de pulsaciones máximas y quema un buen promedio de calorías por sesión de entrenamiento. La predicción tiene sentido

Conclusión

En este proyecto se puede ver que el modelo de Dense Layer dio resultados bastante buenos, la manera de mejorar el modelo fue afectando tanto los epochs así como la cantidad de neuronas y capas de profundidad, pero en si la mayoría de modificaciones que se hicieron resultaron ser bastante eficientes, ya que el promedio de los resultados fue de un 80% a 88% de precisión. Por lo que se puede decir que este modelo se ajusto de manera eficiente a lo que es el dataset y la clasificación de los datos, en este caso los mejores hiperpárametros del modelo fueron:

- Learning Rate = 0.001
- Batch Size = 32
- Neuronas = 32
- Epoch = 35
- Capas densas = 5.

Aunque el objetivo principal del proyecto no era maximizar la precisión del modelo, se buscó un equilibrio entre precisión y eficiencia en el uso de recursos. Se encontró que el mejor rendimiento se obtenía al:

- Reducir el número de neuronas por capa mientras se aumentaba la profundidad. Esta combinación permitía al modelo capturar relaciones complejas sin necesidad de usar una gran cantidad de recursos.
- Utilizar un batch_size bajo (32), lo que facilitó una actualización más frecuente de los pesos, permitiendo un aprendizaje más detallado del modelo.
- Limitar el número de epoch (35) para evitar que el modelo se confundiera y comenzara a oscilar en precisión. Se observó que un número alto de épocas provocaba subidas y bajadas en la precisión de un epoch a otra, lo que sugiere que el modelo podría estar "aprendiendo en exceso" ciertos patrones y perdiendo generalización.

Este ajuste balancea la capacidad del modelo de aprender patrones complejos y la estabilidad en sus resultados, asegurando un rendimiento sólido sin redundancia en el entrenamiento.

En este proyecto vi como implementar de manera eficiente lo que es un modelo de Dense Layer y como tratar de ir diagnosticando el modelo, para poder modificar los hiperpárametros y sacar el mejor resultado posible, en este caso los valores ayudaron y empeoraron no de manera tan significativa pero de todas maneras se saco el modelo más óptimo y justo los retos fue tratar de entender el modelo y ver por donde ir para optimizarlo.

Referencias Bibliograficas

- Crego, A. P. (2023, 11 octubre). Cómo se mide el rendimiento deportivo: métricas | Unisport. Unisport. <https://unisport.es/como-se-mide-rendimiento-deportivo/>
- Comunicacion. (2024, 30 enero). Principios del entrenamiento deportivo y tipos. Ciencias Deportivas. <https://cienciasdeportivas.com/entrenamiento-deportivo-puntos-principales/>
- Romero, R., & Romero, R. (2023, 16 agosto). Estas son las métricas que debes tomar en cuenta - Men's Health Latam. Men's Health Latam - Men's Health México, la guía completa para el hombre. Fitness, nutrición, estilo de vida, mente, moda y estilo, sexo y relaciones, relojes y motores. Todo en un mismo lugar. <https://menshealthlatam.com/estas-son-las-metricas-que-debes-tomar-en-cuenta/>
- Navarro, S. (2024, 8 noviembre). Normalizar los datos en Deep Learning [2024] | KeepCoding. KeepCoding Bootcamps. <https://keepcoding.io/blog/normalizar-los-datos-en-deep-learning/#:~:text=Normalizar%20los%20datos%20en%20deep%20learning%20es%2C%20b%C3%A1sicamente%2C%20transformar%20los,la%20generalizaci%C3%B3n%20de%20los%20modelos.>