Instituto Tecnológico y de Estudios Superiores de Monterrey

Inteligencia Artificial Avanzada para la Ciencia de Datos II (Gpo 501)

Momento de Retroalimentación Individual: Implementación de un modelo de Deep Learning.

Adrián Emmanuel Faz Mercado - A01570770

Noviembre 4, 2023

Introducción

A lo largo de los últimos años, el campo de la clasificación de imágenes por medio de la tecnología ha tenido un crecimiento exponencial, y esto se ha logrado también gracias a la adopción de técnicas de Deep Learning, como lo son las Redes Neuronales Convolutivas. Las CNN son una arquitectura de redes neuronales que se utiliza principalmente para el análisis y reconocimiento de imagenes, y con el uso de filtros, permiten las características principales y patrones de las imagenes, para así realizar alguna predicción o clasificación de una imagen no conocida.

Este tipo de red neuronal se divide en una secuencia de diferentes capas, en donde cada una tiene una tarea específica de procesamiento de la imagen. Primeramente, las capas inferiores se encargan de detectar características simples, como lo son bordes, texturas o colores, mientras que las capas superiores integran esas características para identificar patrones más complejos. De igual forma, existen las capas de pooling, las cuales simplifican la información reduciendo la dimensionalidad espacial de los mapas de características, y permitiendo que las representaciones sean más manejables computacionalmente. Luego, después de las capas convolutivas y de pooling, la red utiliza una o más capas "Fully Connected", en las que se combinan toads las características aprendidas para tomar decisiones de clasificación. La red termina con una capa "Fully Connected" de salida en donde se utiliza una función de activación que corresponde al problema en específico.

El problema a resolver

En esta actividad, se buscará resolver un problema de clasificación de imágenes de diferentes deportes. El objetivo es crear y entrenar un modelo que permita clasificar imágenes entre una variedad de 15 diferentes deportes.

Para ello, se utilizará un data set obtenido de la plataforma Kaggle, el cual contiene ya las carpetas divididas en carpetas de entrenamiento, validación y testing. Los datos utilizados tienen las siguientes características:

- 15 deportes diferentes. (Estas serán las clases)
- 2235 imagenes para el entrenamiento del modelo (149 imagenes de cada deporte)
- 75 imágenes para la validación del modelo durante la fase de entrenamiento (5 imágenes de cada deporte)
- 75 imagenes para la parte de evaluación del modelo (5 imágenes de cada deporte)

Uso de Transfer Learning

Para la resolución del problema y la creación del modelo, se utilizará la técnica de Transfer Learning, la cual permite aprovechar modelos preentrenados con grandes conjuntos de datos para establecer un punto de partida en tareas de clasificación de imágenes. Esto permite que en vez de comenzar el proceso de aprendizaje desde 0, se utilicen pesos de un modelo ya entrenado en un conjunto de datos grande, como lo es ImageNet. Existen diferentes arquitecturas de CNN disponibles para aprovechar el Transfer Learning, tales cómo AlexNet, VGGNet, ResNet y MobileNet.

En este caso, para iniciar, decidí probar con la arquitectura MobileNetV2, pues es una de las arquitecturas que debido a su diseño, tiene un procesamiento más rápido y un tiempo de entrenamiento reducido, a comparación de otras arquitecturas.

Modelo inicial

Para utilizar y entrenar un modelo de aprendizaje profundo, utilicé **Keras y Tensorflow**, los cuales permiten construir y entrenar modelos de aprendizaje profundo en Python. Esta combinación de herramientas permite manipular, crear, entrenar y evaluar modelos de manera flexible y sencilla.

Primeramente, decidí cargar el modelo de MobileNetV2 y probar directamente mis datos de prueba con él, para así analizar la manera en la que trabajaba y las predicciones que podría dar. Dado que este es un modelo que ya estaba cargado, solamente fue cuestión de realizar las predicciones de mis imagenes de prueba y obtuve resultados de la siguiente manera:

```
Imagen 0: Clase predicha - bow, Probabilidad - 0.9959996938705444
Imagen 1: Clase predicha - horizontal_bar, Probabilidad - 0.2348284274339676
Imagen 2: Clase predicha - bow, Probabilidad - 0.6798624396324158
Imagen 3: Clase predicha - bow, Probabilidad - 0.9813103675842285
Imagen 4: Clase predicha - bow, Probabilidad - 0.961994469165802
Imagen 5: Clase predicha - ballplayer, Probabilidad - 0.6180654764175415
Imagen 6: Clase predicha - ballplayer, Probabilidad - 0.7382985949516296
Imagen 7: Clase predicha - ballplayer, Probabilidad - 0.11827856302261353
Imagen 8: Clase predicha - baseball, Probabilidad - 0.3671359121799469
Imagen 9: Clase predicha - fountain, Probabilidad - 0.888058602809906
Imagen 10: Clase predicha - basketball, Probabilidad - 0.9935752749443054
Imagen 11: Clase predicha - basketball, Probabilidad - 0.9053314328193665
Imagen 12: Clase predicha - amphibian, Probabilidad - 0.07215804606676102
```

En las predicciones, podemos observar que las clasificaciones fueron de las 1000 clases con las que previamente fue entrenado el modelo de MobileNetV2. El modelo intenta clasificar las imagenes en base a las 1000 que ya conoce, y en realidad no está completamente equivocado, pues por ejemplo:

- En las imágenes que corresponden a “archery”, algunas se clasificaron como “bow”, es decir a un arco, el cual probablemente se identificó en las imagenes.
- En las que corresponden a “baseball”, algunas se clasificaron cómo “ballplayer”, pues se identificaron a hombres con una pelota.
- En “basketball”, algunas de las imagenes si se identificaron como “basketball”, pues al parecer es una clase que sí conocía el modelo ya.

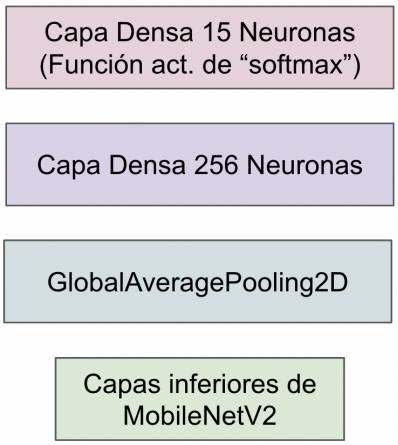
Como se pudo observar, el modelo sí es capaz de clasificar las imagenes entre las clases que ya conoce y a “asignar” la clase que tiene la mayor probabilidad.

Modelo entrenado con capas nuevas (Aproximación Inicial)

Ahora, para asegurar que el modelo sea capaz de predecir el deporte al que corresponde a cada imagen, se entrene con las imagenes nuevas, y que solamente tenga como posible salida las 15 clases existentes, es necesario crear un nuevo modelo, en el cual se utilizará como base la arquitectura de MobileNetV2, pero que también tendrá unas capas extras en la parte superior, incluyendo una capa

densa al final para que las clasificación final sea unicamente entre las 15 clases con las que se entrenó el modelo.

La arquitectura que se siguió en este primer modelo es la siguiente:



Primero, se cargó el modelo de MobileNetV2 con los pesos de imagenet, y el parámetro de “include_top” se inicializó cómo falso, porque las últimas capas son específicas para las clases del conjunto de datos de ImageNet. En este caso, se necesitan reemplazar esas capas con otras nuevas que se entrenarán específicamente para las nuevas clases. De esta forma, se puede aprovechar el conocimiento preentrenado sobre como procesar imagenes en general, pero se personaliza la salida para nuestro conjunto de datos.

```
# Se define la base del modelo con las capas inferiores del modelo de MobileNetV2, no se incluye la "parte superior" de la red pueds es la que contiene las capas densas
# No se incluye la parte superior porque vamos a personalizarla para que pueda clasificar y entrenarse con mis datos y clases.
conv_base = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Se inicia un modelo nuevo secuencial.
model = Sequential()

# Agregamos la base convolucional de MobileNetV2 como la base del modelo, exceptuando la parte superior.
model.add(conv_base)
```

Después de cargar e inicializar el modelo con el modelo de MobilenetV2 como base, se agregaron las siguientes capas, las cuales ya forman parte de la personalización a nuestro conjunto de datos:

```
# Agregamos la base convolucional de MobileNetV2 como la base del modelo, exceptuando la parte superior.
model.add(conv_base)

# Congelamos la base para que no se actualicen los pesos durante el entrenamiento y se mantengan las características que ya ha aprendido.
conv_base.trainable = False

# Se agrega una capa de Global Average Pooling para transformar el último conjunto de mapas de características en un vector.
model.add(GlobalAveragePooling2D())

# Se agrega una capa "Fully Connected" o Densa nueva con 256 neuronas y la función de activación "relu"
model.add(Dense(256, activation='relu'))

# Finalmente, se agrega una última capa densa (Softmax para que se pueda calcular la probabilidad sobre las diferentes clases para la clasificación)
model.add(Dense(NUM_CLASSES, activation='softmax'))
```

- GlobalAveragePooling2D
    - Esta capa reduce los mapas de características a un valor promedio y ayuda a minimizar el sobreajuste de las predicciones

- Capa Densa de 256 Neuronas
    - Se agrega una capa de 256 Neuronas, la cual es una capa “fully connected” y en ella el modelo comienza a aprender relaciones no lineales entre las características que se extraen. Utiliza una función de activación de “relu”, la cual ayuda a mantener no linealidad en el aprendizaje.

- Capa Densa de 15 Neuronas con función de Activación “SoftMax”
    - Capa de salida del modelo, la cual tiene un número de neuronas igual al número de clases posibles. La función de activación “softmax” se utiliza para obtener un vector de probabilidades de las clases, en donde se obtiene la probabilidad de que la imagen pertenezca a cada una de las posibles clases. Técnicamente, la clase que tenga la probabilidad más alta es a la que pertenece y la clase predicha por el modelo.

Además, antes de compilar y comenzar a entrenar el modelo, también se realizó un proceso de “Data Augmentation”, el cual consiste en incrementar la cantidad y diversidad de imagenes de entrenamiento para modelos de aprendizaje automático. Se generan nuevas imagenes a partir de las existentes a través de diferentes transformaciones. Las transformaciones que se realizaron para ampliar el set de entrenamiento fueron las siguientes:

- Normalización de valores de pixeles de la imagen a un rango entre 0 y 1.
- Rotación de imagenes
- Traslaciones horizontales y verticales
- “Zoom in” o “Zoom out” de imagenes
- Las imagenes se voltean de manera aleatoria

Luego, se compiló el modelo con un learning rate de 0.00001 y con la medida de “accuracy” para re entrenarse a si mismo junto con el set de validación.

```
# Compilación del Modelo. Se utiliza el optimizador "Adam" y como métrica la "accuracy".
model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Se comenzó a entrenar el modelo utilizando 10 épocas y la configuración previamente mencionada, y finalmente, se obtuvieron los siguientes resultados de “accuracy” en los datos de entrenamiento y los de validación:

- Accuracy entrenamiento: **0.5692**
- Accuracy validación: **0.6562**

```
Found 2235 images belonging to 15 classes.
Found 75 images belonging to 15 classes.
Epoch 1/10
69/69 [==============================] - 39s 518ms/step - loss: 2.9515 - accuracy: 0.0921 - val_loss: 2.5833 - val_accuracy: 0.1250
Epoch 2/10
69/69 [==============================] - 36s 521ms/step - loss: 2.6351 - accuracy: 0.1425 - val_loss: 2.4015 - val_accuracy: 0.2188
Epoch 3/10
69/69 [==============================] - 35s 506ms/step - loss: 2.4563 - accuracy: 0.2220 - val_loss: 2.2457 - val_accuracy: 0.3125
Epoch 4/10
69/69 [==============================] - 36s 522ms/step - loss: 2.2940 - accuracy: 0.3091 - val_loss: 2.1159 - val_accuracy: 0.3750
Epoch 5/10
69/69 [==============================] - 38s 545ms/step - loss: 2.1550 - accuracy: 0.3663 - val_loss: 1.9300 - val_accuracy: 0.4844
Epoch 6/10
69/69 [==============================] - 34s 486ms/step - loss: 2.0231 - accuracy: 0.4281 - val_loss: 1.7951 - val_accuracy: 0.5000
Epoch 7/10
69/69 [==============================] - 34s 489ms/step - loss: 1.9061 - accuracy: 0.4784 - val_loss: 1.6628 - val_accuracy: 0.5781
Epoch 8/10
69/69 [==============================] - 33s 476ms/step - loss: 1.8072 - accuracy: 0.5079 - val_loss: 1.6052 - val_accuracy: 0.5625
Epoch 9/10
69/69 [==============================] - 33s 482ms/step - loss: 1.7063 - accuracy: 0.5556 - val_loss: 1.4606 - val_accuracy: 0.6562
Epoch 10/10
69/69 [==============================] - 34s 499ms/step - loss: 1.6148 - accuracy: 0.5692 - val_loss: 1.4237 - val_accuracy: 0.6562
```

Los resultados obtenidos no son tan buenos pero aun así, debido a que era la primera aproximación, procedí a evaluar el modelo con los datos de prueba, los cuales aun no habían sido vistos por el modelo.

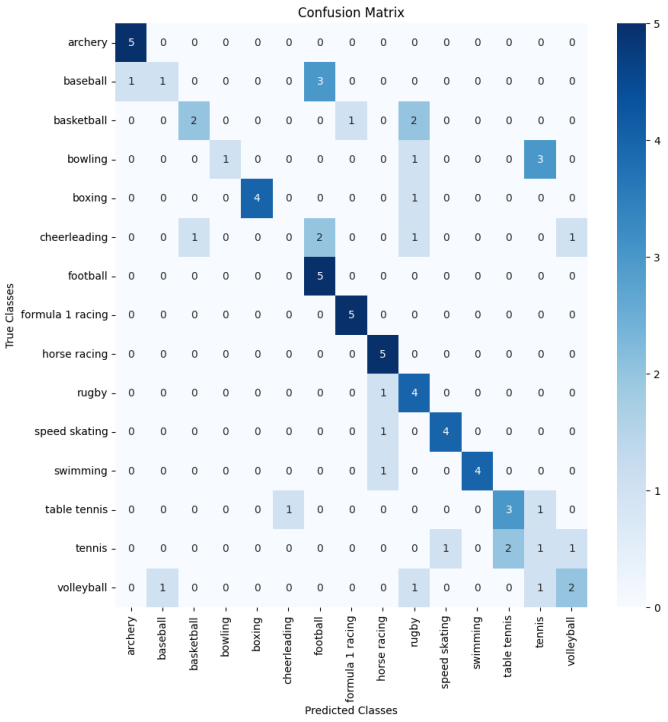
Evaluación del modelo inicial

```
# Calcular el porcentaje de accuracy del modelo.
test_loss, test_accuracy = model.evaluate(test_generator)
print('Test accuracy:', test_accuracy)
```

```
3/3 [==============================] - 0s 59ms/step - loss: 1.3723 - accuracy: 0.6133
Test accuracy: 0.6133333444595337
```

Se obtuvo un 61.33% de accuracy en los datos de prueba, el cual fue un porcentaje muy cercano al que se obtuvo tanto en entrenamiento como en validación, por lo que podemos decir que vamos por buen camino y que no existe hasta el momento overfitting, lo cual sucedería si obtuvimos un valor mucho más bajo que en training, pues significaría que se está aprendiendo los datos de entrenamiento y no está generalizando.

La matriz de confusión de los datos de prueba se vio de la siguiente manera:



En ella se puede observar que si bien la mayoría de los datos se clasificaron correctamente, hay algunas clases que no obtuvieron ningún resultado correcto, cómo lo es “cheerleading”. Además, se observan varios elementos fuera de la diagonal principal, lo cual indica que son datos que no fueron clasificados correctamente.

Fine-tuning y mejora del modelo

Ya que se obtuvieron los primeros resultados del modelo inicial, es posible comenzar a realizar ajustes en diferentes aspectos del modelo para que tenga un mejor desempeño en todas las áreas, tanto en entrenamiento, validación como en los datos de prueba.

Para ello, realicé muchos experimentos modificando diferentes parámetros, agregando capas e incluso modificando la cantidad de épocas y el tamaño del "batch". Sin embargo, para no enlistar todas las modificaciones que realicé para llegar a mi modelo final, enlistaré algunos comentarios y observaciones que realicé al estar cambiando el modelo:

- Comencé agregando un Dropout de 0.5 para apoyar en la parte de regularización del modelo, pues este funciona de tal manera que durante el entrenamiento, se "apagan" la mitad de las salidas de las neuronas en la capa a la que se aplica, esto para evitar que las neuronas en la red desarrollen una dependencia entre sí durante el entrenamiento, lo que puede llevar a que exista quizás overfitting y coadaptación. Sin embargo, me di cuenta que al agregar este Dropout, en realidad mis resultados eran peores, estaba obteniendo un porcentaje mucho menor de "accuracy", y decidí mejor eliminarlo, pues lo que probablemente sucedía era que no lograba aprender suficiente de los datos por el hecho de que la mitad de las neuronas se apagaban aleatoriamente. Además, en el primer intento no había indicios de sobreajuste, por lo que después concluí que quizás no era necesario agregar esa parte extra de Dropout en este caso.

- De igual manera, para continuar con el fine tuning del modelo, decidí probar a descongelar algunas capas del modelo de MobileNetV2, pues inicialmente se habían congelado todas y no se habían modificado. El descongelar capas permite que algunas capas se vuelvan a entrenar con los datos nuevos, y puede dar mejores resultados dependiendo el caso.

- Realicé pruebas modificando la cantidad de "epochs", pero decidí mantener la cantidad inicial, pues al tener menos epochs, a veces no alcanzaba a terminar de converger, y si se agregaban muchas más, terminaba de converger desde antes y no tenía caso tener extras.

Después de varios intentos, los cambios finales que se realizaron en comparación al modelo inicial fueron los siguientes:

1. Se descongelaron las últimas 3 capas de la arquitectura MobileNetV2, para que estas puedan ser reentrenadas con los datos nuevos de las imagenes de deportes.
2. Se modificó el valor del learning rate, y en vez de ser 0.00001, se cambió a 0.0001.

Lo demás se mantuvo igual, pero con estos 2 cambios se pudo obtener un desempeño mucho más bueno.

Estos fueron los resultados del modelo con los cambios realizados al momento de estar entrenandolo:

```
Found 2235 images belonging to 15 classes.
Found 75 images belonging to 15 classes.
Epoch 1/10
69/69 [==============================] - 39s 508ms/step - loss: 2.2582 - accuracy: 0.3191 - val_loss: 1.5044 - val_accuracy: 0.5469
Epoch 2/10
69/69 [==============================] - 35s 508ms/step - loss: 1.4277 - accuracy: 0.6455 - val_loss: 0.9714 - val_accuracy: 0.7344
Epoch 3/10
69/69 [==============================] - 34s 497ms/step - loss: 1.0158 - accuracy: 0.7490 - val_loss: 0.6688 - val_accuracy: 0.7812
Epoch 4/10
69/69 [==============================] - 35s 502ms/step - loss: 0.7704 - accuracy: 0.8034 - val_loss: 0.4687 - val_accuracy: 0.8594
Epoch 5/10
69/69 [==============================] - 34s 495ms/step - loss: 0.6435 - accuracy: 0.8311 - val_loss: 0.3594 - val_accuracy: 0.8594
Epoch 6/10
69/69 [==============================] - 36s 517ms/step - loss: 0.5884 - accuracy: 0.8434 - val_loss: 0.3178 - val_accuracy: 0.9062
Epoch 7/10
69/69 [==============================] - 36s 519ms/step - loss: 0.5125 - accuracy: 0.8597 - val_loss: 0.2918 - val_accuracy: 0.9062
Epoch 8/10
69/69 [==============================] - 36s 520ms/step - loss: 0.4862 - accuracy: 0.8647 - val_loss: 0.2731 - val_accuracy: 0.9062
Epoch 9/10
69/69 [==============================] - 34s 495ms/step - loss: 0.4420 - accuracy: 0.8797 - val_loss: 0.2068 - val_accuracy: 0.9531
Epoch 10/10
69/69 [==============================] - 38s 556ms/step - loss: 0.4296 - accuracy: 0.8729 - val_loss: 0.1733 - val_accuracy: 0.9531
```

Se puede observar que esta vez, el valor de “accuracy” de los datos de entrenamiento y validación fueron los siguientes:

- Accuracy Entrenamiento: **0.8729**
- Accuracy Validación: **0.9531**

Los nuevos valores indican que esta vez el modelo alcanzó una “accuracy” mucho más alta, tanto en entrenamiento como en validación, ambos obteniedno un desempeño por arriba del 85%.

Finalmente, también se probó el modelo con los datos reservados para testeo, pues estos no fueron vistos al momento de estar entrenando:

```
# Calcular el porcentaje de accuracy del modelo.
test_loss, test_accuracy = model.evaluate(test_generator)
print('Test accuracy:', test_accuracy)
```

```
3/3 [==============================] - 0s 59ms/step - loss: 0.2341 - accuracy: 0.9067
Test accuracy: 0.9066666960716248
```

En la evaluación del modelo con los datos de testeo, se obtuvo un desempeño del 90.67% de accuracy, lo cual indica que también el modelo es bueno para clasificar los datos que no se han visto y que tiene una buena capacidad para generalizar.

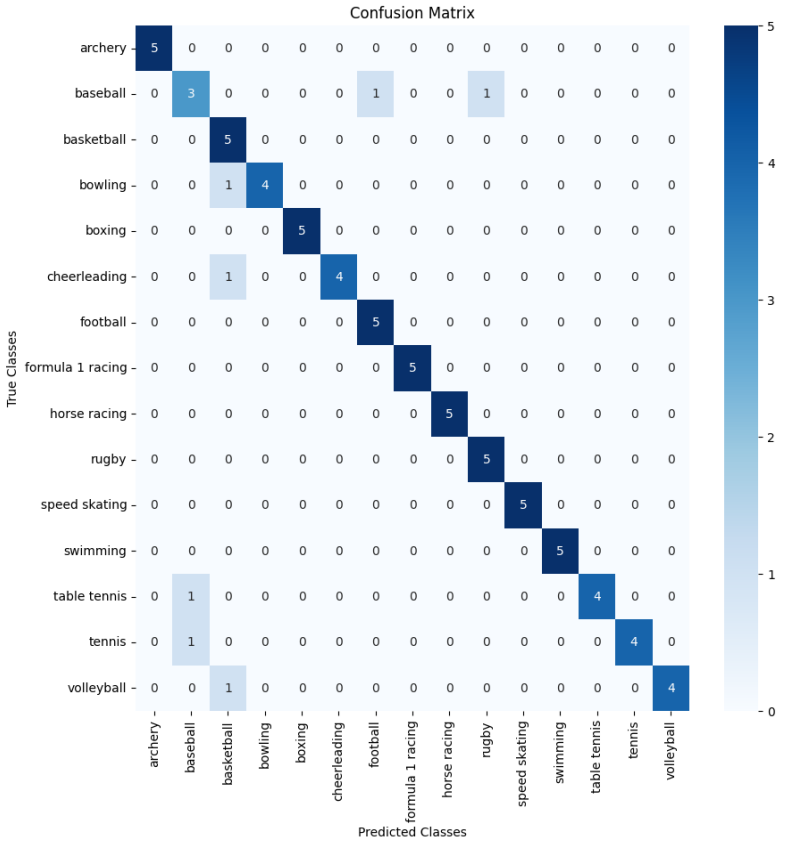
```
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Verificar cuántas predicciones fueron correctas
corrects = np.sum(predicted_classes == true_classes)
print(f"Correct predictions: {corrects} out of {len(true_classes)}")
```

```
3/3 [==============================] - 1s 63ms/step
Correct predictions: 68 out of 75
```

De las 75 imagenes de prueba, 68 fueron clasificadas correctamente, y solo 7 incorrectamente.

La matriz de confusión de este nuevo test se ve de la siguiente manera:



En ella, se puede observar que en la línea diagonal ya está mucho más presente el color azul fuerte, indicando que la gran mayoría de las clases tuvieron clasificaciones correctas. A los lados podemos observar que solamente 7 imagenes no fueron clasificadas correctamente.

Conclusión

En conclusión, los modelos de Deep Learning, especialmente cuando se combinan con técnicas como el transfer learning, nos permiten abordar y resolver problemas complejos, como lo es la clasificación de imagenes. El transfer learning nos permite aprovechar los conocimientos adquiridos por modelos previamente entrenados en grandes datasets, facilitando y acelerando el desarrollo de modelos eficaces con menos datos y en menor tiempo.

Si bien el resultado que se obtuvo al finalizar el fine tuning y el ajuste de parámetros fue bueno, sin duda alguna es posible mejorar el desempeño a través de la modificación de parámetros y la experimentación con nuevas capas y otro tipo de arquitecturas. El Deep Learning cuenta con un gran potencial para impulsar avances significativos en la inteligencia artificial, ofreciendo la capacidad de entender y procesar datos de maneras que imitan la percepción humana.

Liga al Google Collab donde está el código:

https://colab.research.google.com/drive/14TZKdqH3sKMYmkkrffNFTc5dKlFFS8R_?usp=sharing