Instituto Tecnológico y de Estudios Superiores de Monterrey

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

Actividad Deep Learning

Francisco Mestizo Hernández - A01731549

Noviembre 04, 2023

Introducción

Para esta actividad, se utilizarán herramientas vistas en la clase para desarrollar modelos de deep learning basandose en redes pre entrenadas y haciendo un fine tuning para el caso de uso específico. En este documento se mostrará el proceso desde elegir la red pre entrenada hasta hacerle fine tuning para hacer un clasificador de nueces.

Modelos pre entrenados y fine tuning

La técnica que se utilizará para hacer el desarrollo consiste en tomar una red que fue entrenada previamente con un set de datos muy grande y tomar los pesos que se generaron en ese entrenamiento. Después, se agregan nuevas capas para que el modelo de los resultados de las clases específicas al caso de uso. Con estas nuevas capas se hace un reentrenamiento de las capas finales de la red pre entrenada, ya que son las capas que extraen los rasgos más específicos de los resultados que se quieren obtener. A este proceso se le llama fine tuning.

Una ventaja de este método es poder aprovechar el conocimiento de diferentes dominios generado con el pre entrenamiento y adaptarlo al dominio del nuevo caso de uso. Por ejemplo, hay redes que para su entrenamiento utilizaron el set de datos de ImageNet, por lo que está entrenado para dominios como animales, paisajes, tipos de arte, tecnología, objetos, plantas, lugares y unos cuantos dominios más.

Set de datos y elección del modelo

El caso de uso que se quiere lograr es clasificar imágenes de 10 tipos diferentes de nueces. El dataset consta de aproximadamente 160 imágenes por clase para training, 5 imágenes de testing por clase y 5 imágenes de validación por clase. Este dataset es libre de uso y se puede encontrar en Kaggle (Anexo 1).

Es importante mencionar que el set de datos no es demasiado grande, por lo que el modelo pre entrenado que se elija debe tener una arquitectura sencilla, ya que un modelo con una arquitectura muy compleja podría causar un problema de overfitting. Incluso, podría ser recomendable utilizar otras técnicas de clasificación como un Random Forest, pero se intentará hacer el uso de Deep Learning ya que ese es el objetivo de la actividad.

Algunos de los modelos más sencillos que ofrece Keras para usar son MobileNetV2 y DenseNet169. Estos dos fueron entrenados con ImageNet y tienen 2 millones de parametros

aproximadamente. Para esta actividad se harán pruebas con los dos, pero para el que se hagan más pruebas será MobileNetV2.

Entrenamiento

Para hacer el entrenamiento se siguieron varios pasos, donde se fueron cambiando diferentes parámetros y estrategias hasta llegar a un modelo que diera buenos resultados.

Primero, se puede comprobar que el modelo sí puede hacer predicciones pero desconoce las clases que queremos clasificar. Por ejemplo, se probó con una imagen de un coco y el resultado fue maquillaje en polvo; o para una imagen de pistaches se obtuvo el resultado de pepinos.

```
image_path = '/content/drive/MyDrive/Tec/ColabAI/nuts/valid/pistachios/1.jpg'
features = predict_image(image_path)
features
```

```
1/1 [==============================] - 0s 47ms/step
('cucumber', 0.5369147)
```

Para hacer que el modelo clasifique las clases nuevas, se deben agregar nuevas capas al modelo pre entrenado. En este caso se añadieron 4 capas nuevas. La primera un Global Average Pooling 2D para extraer la información de la última capa convolucional de la red pre entrenada y extraer las características principales. A esta capa la sigue una capa Densa de 256, una capa Dropout con 0.5 para regularizar el modelo y una última capa densa con el tamaño 10, que son la cantidad de clases que se quieren clasificar.

A continuación se muestran los resultados del entrenamiento usando estas nuevas capas y sin modificar las capas pre entrenadas. Se utilizarán 3 epochs.

```
Epoch 1/3
19/19 [==============================] - 617s 32s/step - loss: 2.0176 - accuracy: 0.3465 - val_loss: 0.8743 - val_accuracy: 0.7600
Epoch 2/3
19/19 [==============================] - 5s 251ms/step - loss: 1.1191 - accuracy: 0.6277 - val_loss: 0.4769 - val_accuracy: 0.8800
Epoch 3/3
19/19 [==============================] - 5s 243ms/step - loss: 0.8508 - accuracy: 0.7223 - val_loss: 0.3741 - val_accuracy: 0.8800
```

Se están usando muy pocas imágenes y esto puede causar que el modelo no generalice correctamente las clases. Por esto, se utilizan técnicas de data augmentation. Las que se utilizan para este entrenamiento consisten en cambiar los valores de los píxeles a una escala de 0 a 1, cambiar la rotación, el tamaño, realizar recortes a la imagen, voltearla horizontalmente o hacerle zoom.

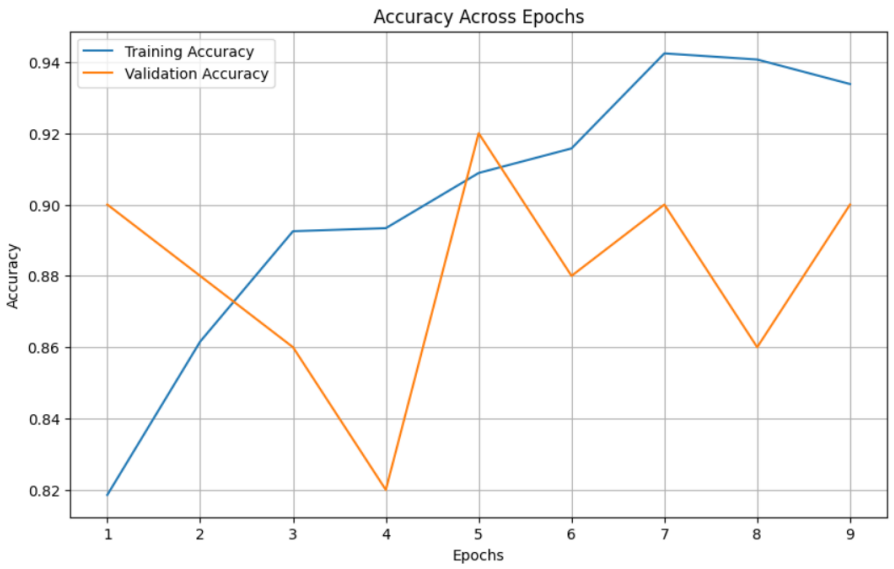
Estos cambios se realizan únicamente al set de datos de entrenamiento, ya que se quiere que se generalicen las características de los objetos y que sea más robusto en casos reales. Pero para los sets de prueba y validación no se realiza porque las imágenes ya están a cómo se encontraría en la vida real. Los resultados que se obtienen con este cambio son los siguientes:

```
Epoch 1/3
19/19 [==============================] - 26s 1s/step - loss: 2.0641 - accuracy: 0.3267 - val_loss: 0.9781 - val_accuracy: 0.7200
Epoch 2/3
19/19 [==============================] - 20s 1s/step - loss: 1.2808 - accuracy: 0.5907 - val_loss: 0.6267 - val_accuracy: 0.8200
Epoch 3/3
19/19 [==============================] - 21s 1s/step - loss: 1.0765 - accuracy: 0.6397 - val loss: 0.4317 - val accuracy: 0.8800
```

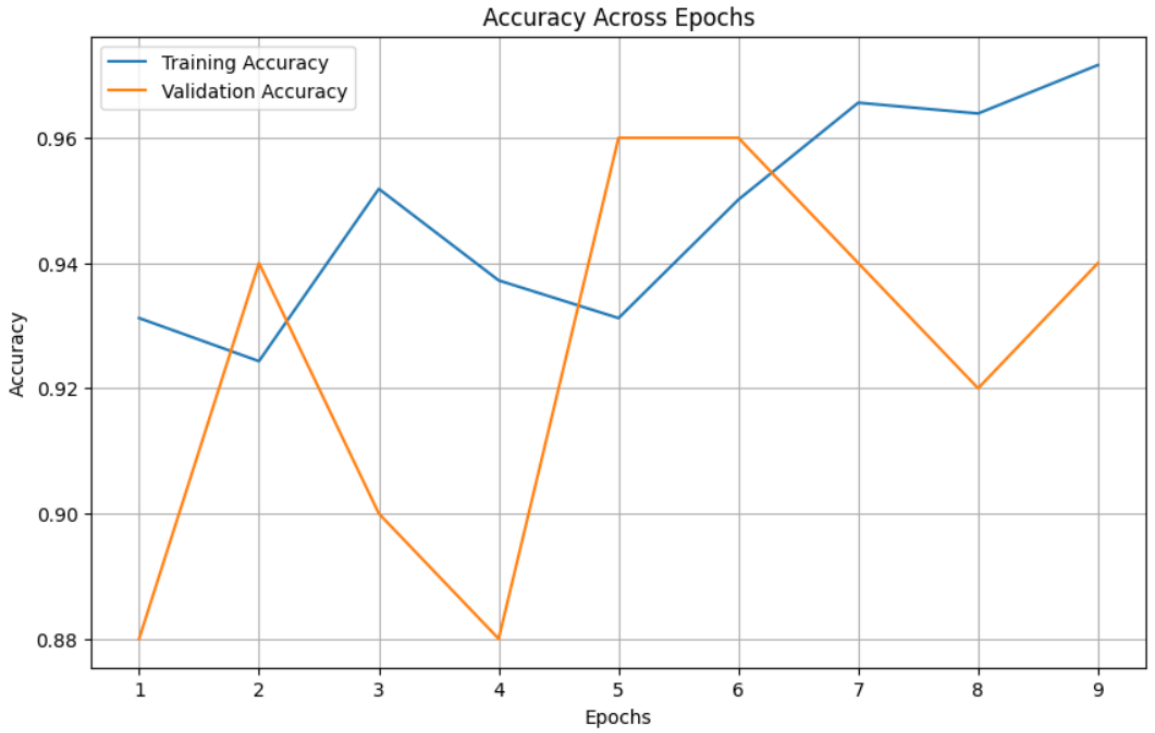
Ahora, se puede probar el desempeño del modelo volviendo a entrenar las últimas capas del modelo. Hasta ahora, los resultados obtenidos habían sido con todas las capas del modelo base congeladas. Se descongelarán las últimas 4 capas para comprobar si el modelo obtiene una mejoría. Esto implica que se estrenarán 743306 parámetros.

```
=================================================================================================
Total params: 2588490 (9.87 MB)
Trainable params: 743306 (2.84 MB)
Non-trainable params: 1845184 (7.04 MB)
_________________________________________________________________________________________________
Epoch 1/3
19/19 [==============================] - 28s 1s/step - loss: 1.0134 - accuracy: 0.6664 - val_loss: 0.4415 - val_accuracy: 0.8200
Epoch 2/3
19/19 [==============================] - 22s 1s/step - loss: 0.6725 - accuracy: 0.7730 - val_loss: 0.3212 - val_accuracy: 0.9000
Epoch 3/3
19/19 [==============================] - 20s 1s/step - loss: 0.4817 - accuracy: 0.8555 - val_loss: 0.3519 - val_accuracy: 0.8600
```

Con el fine tuning se obtienen resultados mejores a los vistos anteriormente, pero parece que el modelo podría mejorar aún más si se aumentara la cantidad de epochs que realiza, así que se probarán los resultados aumentando a 9 epochs.



Con las modificaciones hechas el modelo parece ser bastante estable ya que tiene muy buenos resultados para el set de training como para el de validación, con un accuracy de más de 0.9 para los dos. Por último, se hará el último cambio para tratar de mejorar el modelo un poco más. Se probará implementar una reducción del learning rate para intentar que el modelo haga overfitting sobre los datos de entrenamiento.



Así, se llegó a un modelo con un accuracy de 97% en el entrenamiento y de 94% en el set de validación.

Es importante mencionar que para evaluar un modelo no solamente se debe de tomar en cuenta la métrica del accuracy. Por eso se muestran algunas otras métricas obtenidas para el modelo final. Uno de los objetivos es disminuir el loss para la validación y el entrenamiento.

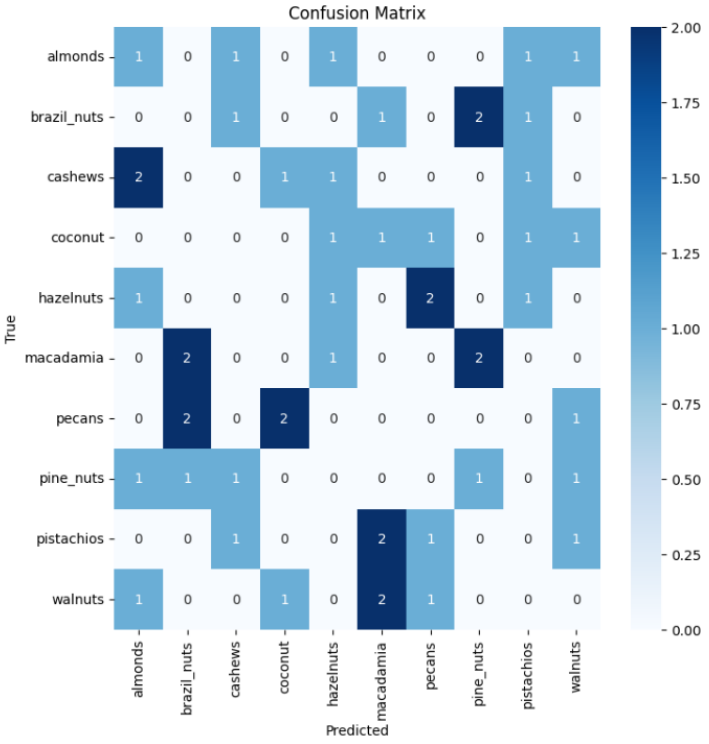
```
Final Training Loss: 0.11654035747051239
Final Validation Loss: 0.27209243178367615
```

También se obtuvieron valores altos para las métricas de precision y recall

```
Final Training Accuracy: 0.9613069891929626
Final Validation Accuracy: 0.8999999761581421
Final Precision: 0.9662629961967468
Final Recall: 0.9604471325874329
```

Evaluación de resultados

Finalmente, se probará el modelo con el set de datos que no se ha utilizado, esto con el objetivo de ver que tan bien funciona en casos de la vida real. Para ver los resultados se muestra una matriz de confusión.



A pesar de los resultados de las métricas, se obtuvieron resultados poco alentadores con la matriz de confusión. Se sabe que el modelo no está haciendo overfitting por las métricas obtenidas, pero los resultados de la matriz de confusión pueden estar dados por un error en el procesamiento de los datos antes de pasarlos al modelo.

Anexos

1. *Tree nuts image classification*. https://www.kaggle.com/datasets/gpiosenka/tree-nuts-image-classification
2. *Google colab con el desarrollo del modelo* https://drive.google.com/file/d/1vyOmhEpMBlZy_Xkn8rn_xsP22hrEjL6I/view?usp=sharing