Fidel Alejandro Navarro Salazar

217202813

Seminario de Física Computacional:

Proyecto Red Neuronal CNN

Introducción

En este trabajo se entrenó una red neuronal tipo CNN para generar un modelo capaz de clasificar distintas imágenes. Para ello se utilizó la biblioteca de Tensorflow en Python para la construcción y entrenamiento del modelo.

Los datos utilizados corresponden a una muestra de fotos de perros y gatos. Estos fueron obtenidos de Kaggle [1] como parte de una competencia con datos proporcionados por Microsoft Research.

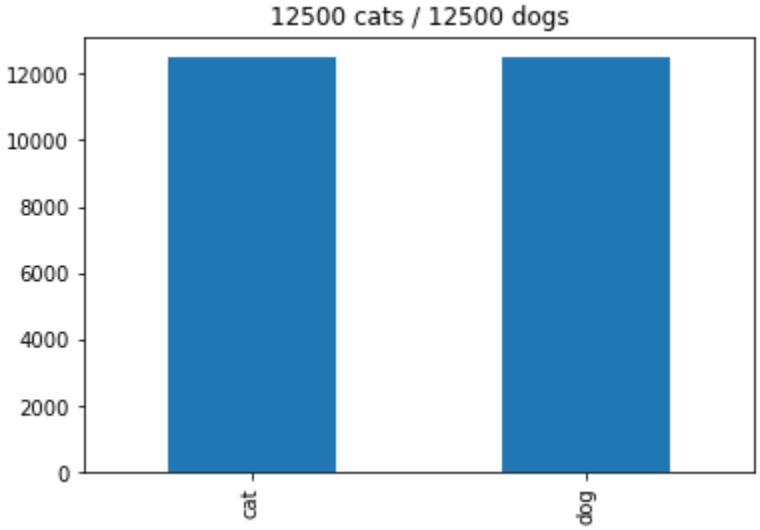

Fig. 1: Características de los datos.

Los datos que se utilizaron para el entrenamiento del modelo están conformados por 12500 fotos de perros y 12500 fotos de gatos. Los datos fueron separados en dos dataframes, uno de entrenamiento con el 80% de las imágenes y uno de prueba con 20% de las imágenes.

```
#Imagen aleatoria de nuestro dataset
load_img(train_dr+"/"+random.choice(df.filename))
```

Fig. 2: Muestra de imagen.

El modelo base se definió con dos capas de convolución y maxpooling, con dimensiones de 3x3 y 2x2 respectivamente y función de activación Rectificador (relu).

```
#Definimos el modelo a utilizar
#Para este ejemplo usaremos 2 capas con matrices de 3x3 de convolución y matrices de 2x2 de maxpooling.
#Posteriormente se aumentará el número de capaz y las dimensiones de las matrices
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation="relu", input_shape=(120, 120, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation="relu"),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

Fig. 3: Definición del modelo.

Para el entrenamiento del modelo se utilizó un tamaño (batch size) de 64 y 50 ciclos (epoch). A continuación, se presenta el resultado obtenido para este modelo.

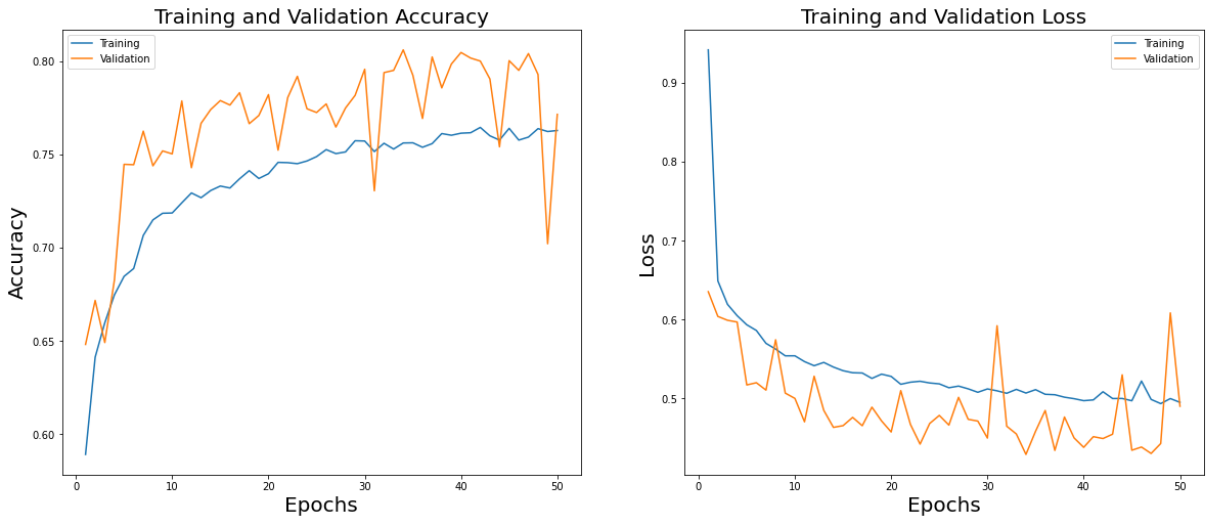

Fig. 4: Accuracy y Loss del modelo base.

De los resultados de este modelo podemos observar que no se presenta overfitting, alcanza un Accuracy de aproximadamente 0.8 y un Loss de aproximadamente 0.5

Posteriormente se realizaron distintas variaciones al modelo con la finalidad de observar como cambian los resultados del modelo para distintos parámetros y poder mejorar los resultados. Las variaciones realizadas son:

1) Aumento del número de capas de convolución y maxpooling.
2) Cambio de dimensiones de las matrices de convolución y maxpooling.
3) Uso del optimizador RMSprop.
4) Uso del optimizador Adam.
5) Cambio del tamaño (batch size).

Variación 1: Aumento de número de capas

Se añadió una capa adicional al modelo con función de activación relu y dimensiones de 3x3 y 2x2 a las matrices de convolución y maxpooling respectivamente.

```
#Definimos el modelo a utilizar
#Agregamos una capa extra
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation="relu", input_shape=(120, 120, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation="relu"),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation="relu"),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

Fig 5: Definición del modelo con una capa adicional.

Para el entrenamiento del modelo se utilizaron los mismos parámetros al modelo base. Los resultados obtenidos se presentan a continuación.

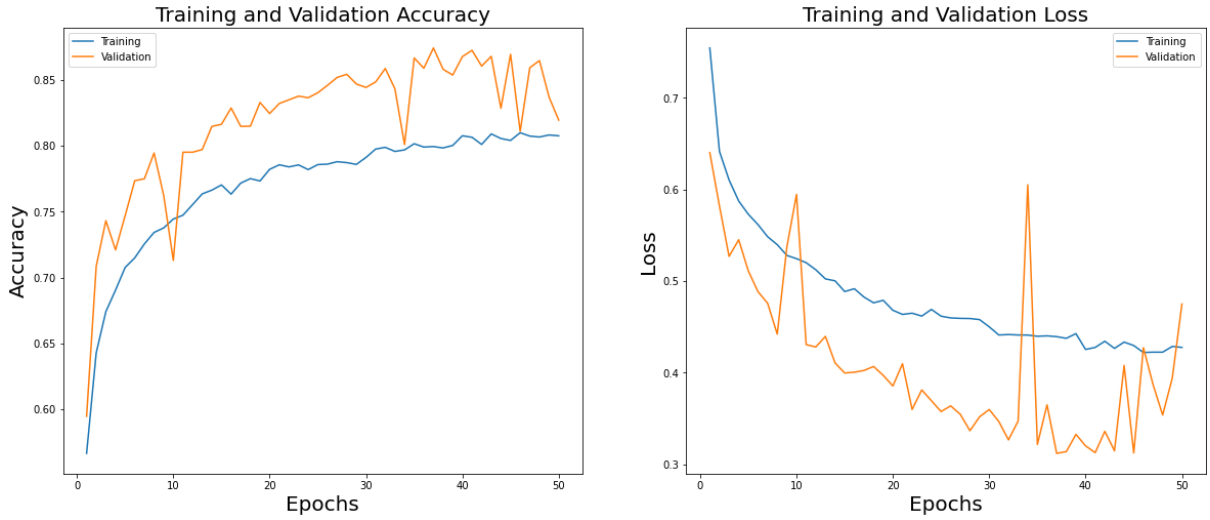

Fig. 6: Accuracy y Loss del modelo con capa adicional.

De la figura 6 se observa que el modelo presentó un Accuracy mayor a 0.8 y un Loss menor a 0.5, mostrando mejoría al compararlo con el modelo base (figura 4). Por lo tanto, para las posteriores variaciones se conservará esta capa adicional en el modelo.

Variación 2: Cambio de dimensiones

En este modelo se cambiaron las dimensiones de las matrices de convolución y maxpooling. Previamente se utilizó una matriz de convolución de 3x3 y una matriz de maxpooling de 2x2, ahora se implementará una matrizx de 4x4 para convolución y 3x3 para maxpooling.

```
#Definimos el modelo a utilizar
#Cambiamos las matrices a utilizar
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (4,4), activation="relu", input_shape=(120, 120, 3)),
    tf.keras.layers.MaxPooling2D(3,3),
    tf.keras.layers.Conv2D(32, (4,4), activation="relu"),
    tf.keras.layers.MaxPooling2D(3,3),
    tf.keras.layers.Conv2D(64, (4,4), activation="relu"),
    tf.keras.layers.MaxPooling2D(3,3),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation="relu"),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.Dense(2048, activation="relu"),
    tf.keras.layers.Dense(1, activation="sigmoid")
```

Fig. 7: Definición del modelo con distintas matrices de convolución y maxpooling.

Los parámetros utilizados para el entrenamiento del modelo fueron similares a los utilizados en el modelo anterior (variación 2), conservando la capa adicional.

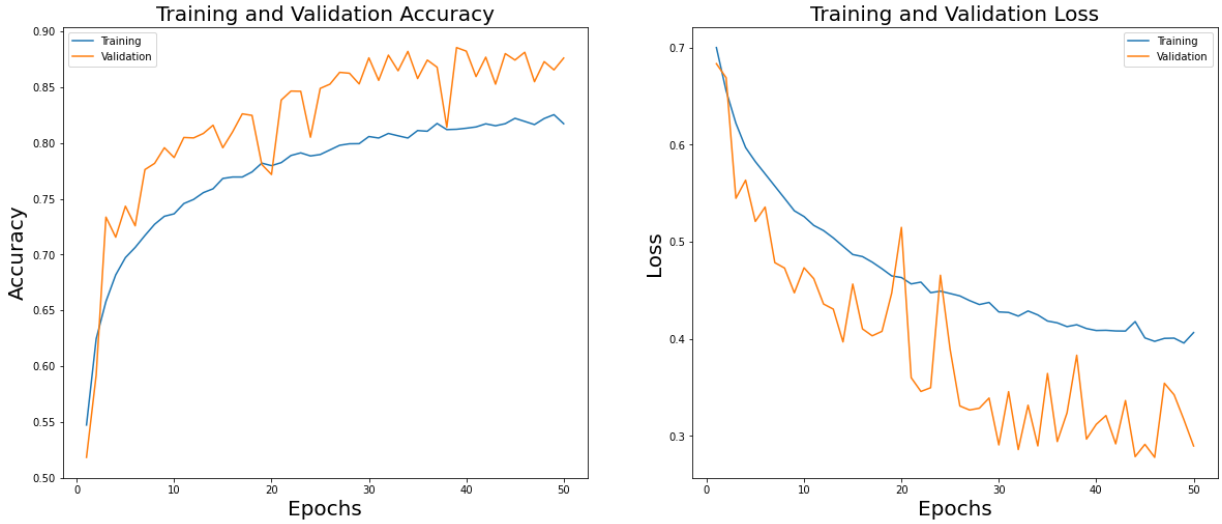

Fig. 8: Accuracy y Loss del modelo con matrices 4x4 (convolución) y 3x3 (maxpooling).

De la figura 8 se observa que el modelo presenta un Accuracy mayor a 0.8 para los datos de entrenamiento y mayor a 0.85 para los datos de prueba. Por otro lado, presenta un los menos a 0.4 para los datos de entramiento y menor a 0.3 para los datos de prueba. Esta variación mostro mejoría en los resultados del modelo; por lo tanto, se conservarán las dimensiones de las matrices en variaciones posteriores.

Variación 3: Optimizador RMSprop

Utilizando los mismos parámetros que el modelo anterior (variación 2) se implementó el optimizador RMSprop con una tasa de aprendizaje (learning rate) de 0.001.

```
#Agregamos el Optimizer RMSprop con un learning rate de lr=0.001
model.compile(optimizer=RMSprop(lr=0.001),
              loss="binary_crossentropy",
               metrics = ["accuracy"])
```

Fig. 9: Implementación del optimizador RMSprop.

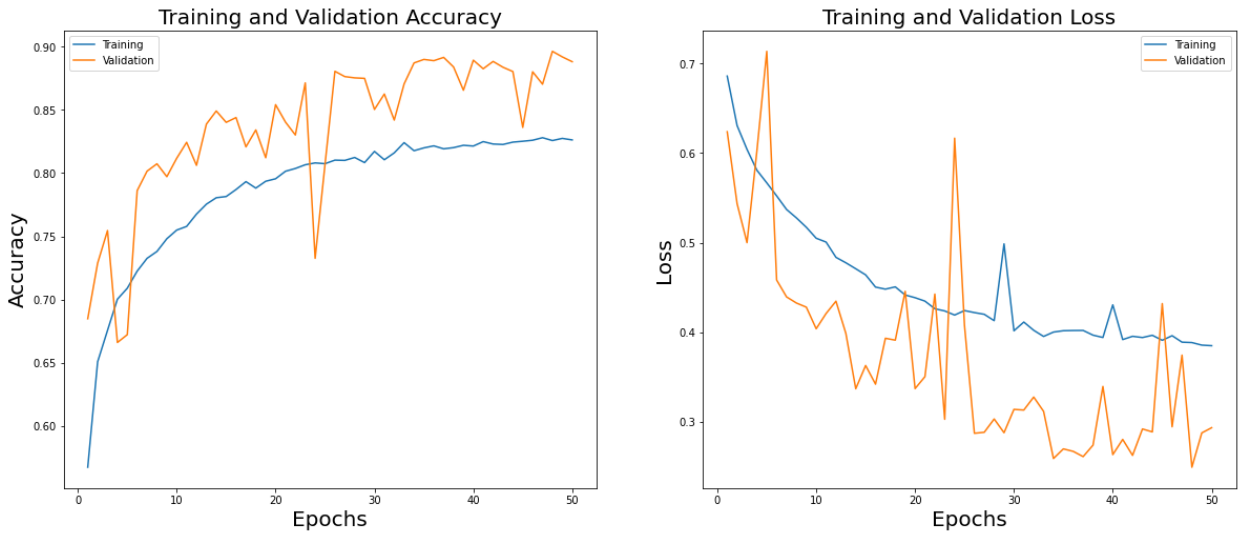

Fig. 10: Accuracy y Loss del modelo con optimizador RMSprop con learning rate de 0.001.

De la figura 10 observamos que el modelo presenta un Accuracy mayor a 0.8 para el entrenamiento y mayor a 0.85 para la prueba. Además, presenta un Loss menor a 0.4 para el entrenamiento y menor a 0.3 para la prueba. Comparando los resultados con el modelo de la variación 2 la única diferencia apreciable es que al implementar el optimizador RMSprop las variaciones en las gráficas son más marcadas.

Variación 4: Optimizador Adam

Utilizando los mismos parámetros que el modelo de la variación 2 se implementó el optimizador Adam con un learning rate de 0.001.

```
#Agregamos el Optimizer Adma con un learning rate de lr=0.001
model.compile(optimizer=Adam(lr=0.001),
              loss="binary_crossentropy",
               metrics = ["accuracy"])
```

Fig. 11: Implementación del optimizador Adam.

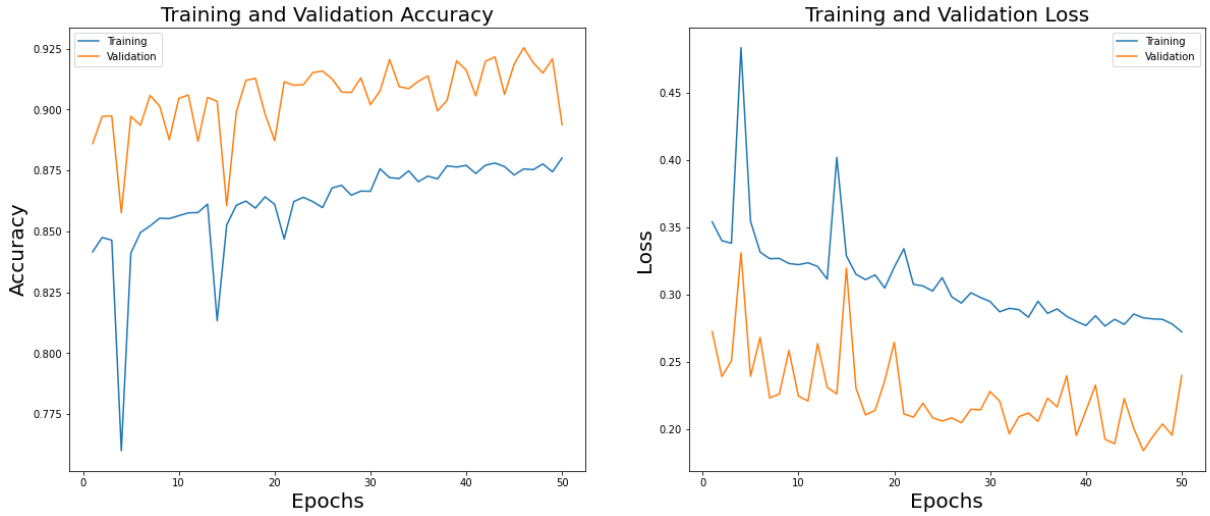

Fig 12: Accuracy y Loss del modelo con optimizador Adam con learning rate de 0.001.

Podemos observar en la figura 12 que el Accuracy del modelo es mayor a 0.87 para el entrenamiento y mayor a 0.9 para la prueba. El modelo presenta un Loss menor a 0.3 en el entrenamiento y menor a 0.25 para la prueba. Al comparar el modelo con los resultados obtenidos por la variación 2 y 3 observamos una mejor en los valores alcanzados del Accuracy y Loss. Adicionalmente, se mejoró el tiempo de entrenamiento, la variación 4 presentó un tiempo de ejecución de 1.69 horas, mientras que la variación 2 y 3 presentaron un tiempo de ejecución de 1.76 y 1.75 horas respectivamente.

Debido a la mejoría de los resultados se conservará el modelo de la variación 4 para las próximas variaciones.

Variación 5: Batch Size

Previamente se utilizó un tamaño (batch size) de 64 para las variaciones 1, 2, 3 y 4, para este caso se utilizará un tamaño de 32. La finalidad de reducir el tamaño es para observar cómo se comportaría el modelo si reducimos el número de imágenes procesadas por ciclo.

Para el entrenamiento del modelo se hizo uso de los mismos parámetros de la variación 4.

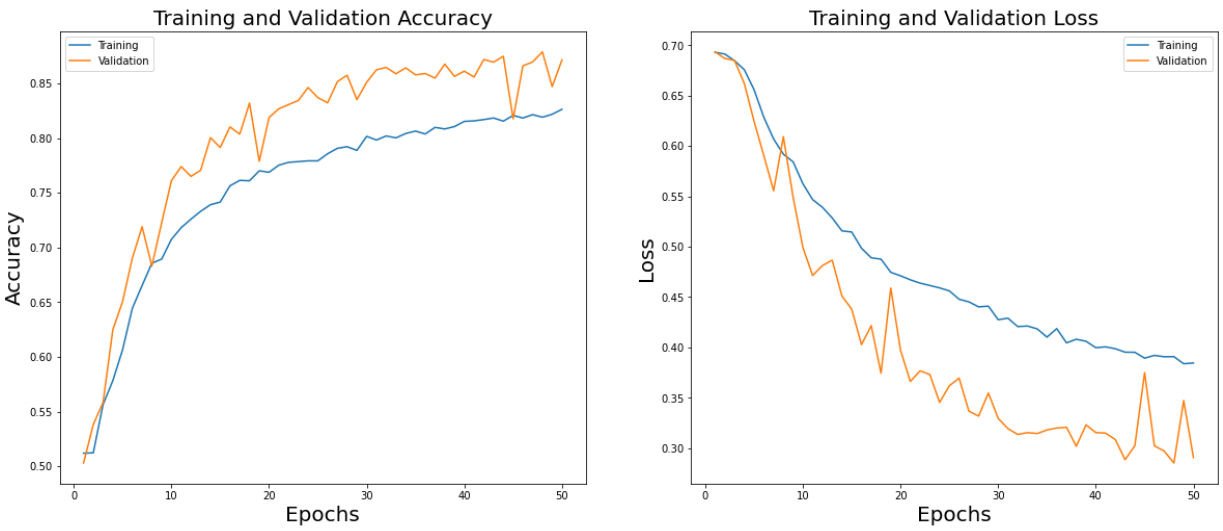

Fig. 13: Accuracy y Loss del modelo con batch size de 32.

En la figura 13 se observa que el Accuracy del modelo es mayor a 0.8 para el entrenamiento y prueba, mientras que el Loss es menor a 0.35 para el entrenamiento y menor a 0.3 en la prueba. Comparando los resultados con los obtenidos en la variación 4 observamos que se obtuvo un Accuracy menor para el entrenamiento y prueba, y un aumento en el Loss en la variación 5. Sin embargo, la variación 5 muestra un comportamiento del Accuracy y Loss más continuo.

Para finalizar utilizaremos el modelo obtenido (variación 4) para clasificar las fotos tomadas de distintas mascotas. Podemos observar de la figura 14 que le modelo fue capaz de predecir que la primera imagen correspondía a un gato con 96.86 porcentaje de confianza y que la segunda imagen corresponde a la de un perro con 90.09 porcentaje de confianza.

Esta imagen probablemente corresponde a un GATO con un 96.86 porcentaje de confianza.

Esta imagen probablemente corresponde a un PERRO con un 90.09 porcentaje de confianza.

Fig. 14: Uso del modelo para clasificar imágenes de perros y gatos.

Conclusión

Por medio de las distintas variaciones fue posible determinar los parámetros y la arquitectura para generar un modelo adecuado capaz de clasificar con precisión alta distintas imágenes de perros y gatos. Al final el modelo utilizado es el definido en la variación 4:

1) Batch size de 64
2) Tres capas de convolución y maxpooling con dimensiones 4x4 y 3x3 respetivamente
3) Optimizador Adam con learning rate de 0.001

Este modelo presentó el Accuracy más alto y Loss más bajo, como se muestra en la figura 12. Además, fue capaz de clasificar adecuadamente distintas imágenes de prueba, como se observa en la figura 14.

A pesar de ser un modelo adecuado para la clasificación de imágenes se recomendaría para futuros modelos aumentar el numero de epochs ya que en ninguna de las variaciones se observó que los valores de Accuracy y Loss convergieran. En este trabajo no fue posible aumentar el epoch por cuestiones de tiempo.

Referencias

1) Kaggle. "*Dogs vs. Cats*". (2013, 25 septiembre). Kaggle. https://www.kaggle.com/c/dogs-vs-cats