ITESM- Campus Puebla

Reporte implementación de un modelo de deep learning

Inteligencia artificial avanzada para la ciencia Datos II

Alumno:

José Antonio Bobadilla García A01734433

Fecha: 04/11/2022

I. Introducción

Deep Learning es un subcampo del machine learning que estructura algoritmos en capas para crear una "red neuronal artificial" que puede **aprender y tomar decisiones inteligentes por sí misma.**

Se considera una red profunda cuando dicha red cuenta con 3 o más capas profundas para el entrenamiento. Entre más capas profundas tenga la red, ésta podrá detectar más características del input dado.

II. Problema a resolver

El problema que se quiere resolver es la clasificación de imágenes de perros y gatos. Lo que se quiere realizar es un sistema que usando redes neuronales recurrentes aprenda a detectar patrones para decidir si una imagen ingresada al sistema es un perro o un gato.

Cabe mencionar que como son imágenes, las redes neuronales no pueden procesar imágenes así tal cual, se tiene que hacer un preprocesamiento de las imágenes para poder alimentar a la red neuronal.

III. Dataset

El dataset con el que se cuenta son **12,500** imágenes anteriormente labeleadas de gatos y **12,500** imágenes de perros igualmente labeleadas.

Un ejemplo de la carpeta que contiene imágenes de perros es la siguiente:

Por el otro lado, la carpeta que contiene las imágenes de gatos es la siguiente:

IV. División de datos en prueba y entrenamiento

Lo primero que se hizo fue dividir estas imágenes en prueba y entrenamiento, tomando 80% para entrenamiento y 20% para pruebas.

Al finalizar esta división manual, se hicieron 2 datasets, uno que contenía en total 8,000 fotos para entrenamiento: 4,000 fotos de perros y 4,000 fotos de gatos y 2,000 fotos de perros y gatos respectivamente para la prueba.

El segundo dataset contenía todo el dataset original, terminando con 20,000 fotos de perros y gatos respectivamente para el entrenamiento y 5,000 para prueba.

Lo que se quería lograr con esta creación de 2 datasets es probar diferentes redes con los 2 datasets, ya que como sabemos, entre más datos tenga la red neuronal, más precisa puede llegar a ser. Entonces se quería ver si utilizando los diferentes datasets, se obtuvieran mejores resultados.

V. Tecnologías a usar

La librería a usar para la creación de la red neuronal será Keras, y la justificación de el por qué usar Keras y no alguna otra librería, es la facilidad de creación de modelos de redes neuronales, ya que se necesitan unas cuantas lineas de codigo para poder crear, entrenar, probar y validar los modelos necesarios.

VI. Equipo usado en el entrenamiento

El equipo usado para entrenar los modelos tiene las siguientes características:

- **Procesador**: Intel core i5 10th generation.
- **Memoria RAM**: 16 GB
- **Almacenamiento**: 2TB HDD y 1TB SSD
- **Tarjeta Gráfica**: NVIDIA GTX 1650 Ti
- **Sistema operativo:** Windows 10

VII. Preprocesamiento de las imágenes

Como sabemos, las redes neuronales no pueden recibir las imágenes como archivos, entonces tenemos que convertirlas a valores para que la red pueda procesarlas.

Lo primero que se hizo fue realizar un aumento de las imágenes para poder procesarlas de una mejor manera. Se utilizó la librería de Keras ImageDataGenerator.

VIII. Cargado de los datos de entrenamiento y prueba

Ya que tenemos dividido por carpetas los datos de entrenamiento y prueba ahora necesitamos cargarlas al sistema para poder entrenar el modelo que se use.

se crearon 2 variables, una para el entrenamiento y otra para pruebas, se usó la función de keras flow_from_directory() que básicamente toma un directorio y genera batches de los datos aumentados.

Con estos datos ya podríamos entrenar los modelos correspondientes.

IX. Modelo a usar

Como ya mencionamos se utilizó la librería de Keras para el modelado de la red neuronal y se eligió el modelo Sequential, el cuál nos permite crear capas de nodos encima una de otra.

La estructura básica de un modelo sequential de Keras tiene las siguientes características:

- Capas y su orden en el modelo.
- Forma de salida
- Número de parámetros (pesos) en cada capa.
- Número total de parámetros en el modelo.

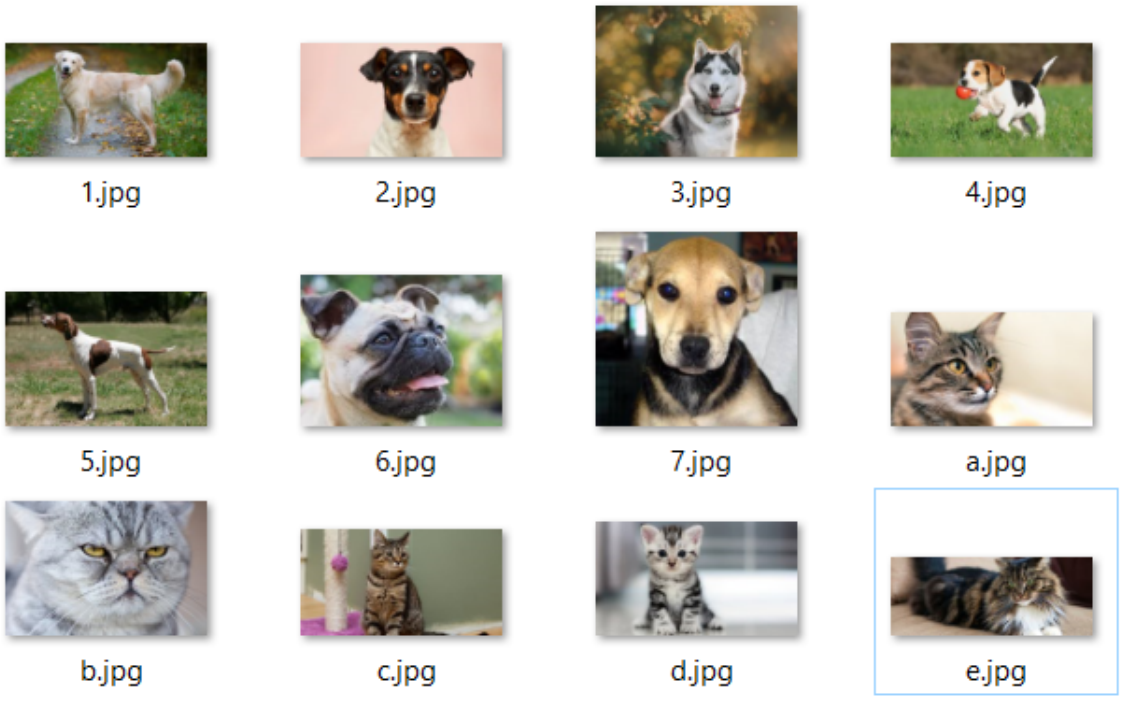
En el caso de nuestro modelo o de los modelos secuenciales a implementar tendrían la siguiente estructura:

- Capas profundas: A definir
- Forma de salida: 1 nodo (0 u 1)
- Capa de entrada: tamaño y dimensiones de imagen.

Se usó esta estructura para definir las diferentes redes y seleccionar la que tenga un mejor rendimiento.

X. Prueba del modelo

Para probar el modelo se utilizarán las siguientes imágenes sacadas de google:



Se encuentra un archivo llamado **predict_all_models.ipynb** el cuál prueba todos los modelos con dichas imágenes y saca la predicción de cada una.

XI. Primer Approach

Como primer intento de una red neuronal se añadió al modelo la capa de entrada, la cual contiene el tamaño de la imagen la cual es de 64x64.

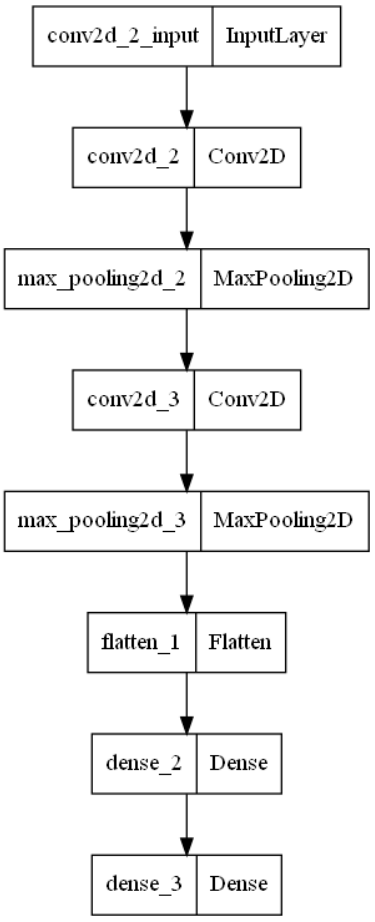
Posteriormente se añadió una capa de Max pooling con una forma de 2,2.

Después de esto se añadió una capa profunda de 32 nodos con una función de activación ReLu y un Max pooling igualmente de forma 2,2.

Después de esto se añadió una capa de flatten para añadir después una última capa profunda de 128 nodos con función de activación ReLu y una última capa de 1 nodo con una función de activación sigmoid.

La estructura de la red quedó de la siguiente manera.

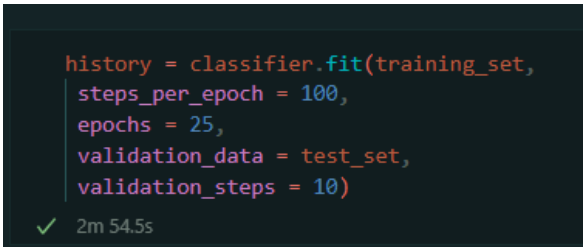
Estructura de la red:




```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 62, 62, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 31, 31, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 29, 29, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 6272)              0

 dense_2 (Dense)             (None, 128)               802944

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
None
```

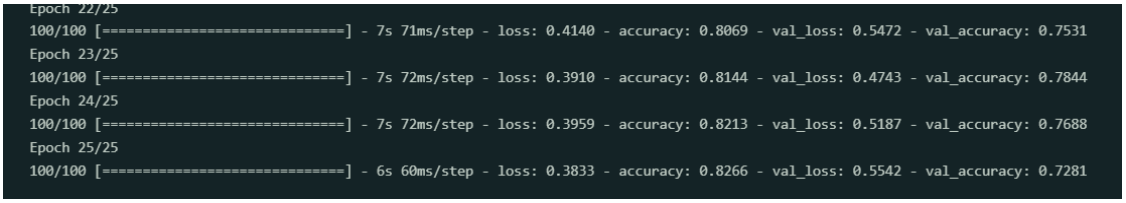
Se configuraron los siguientes hiperparametros con solamente 25 épocas y 100 pasos por época.

Hiperparámetros:



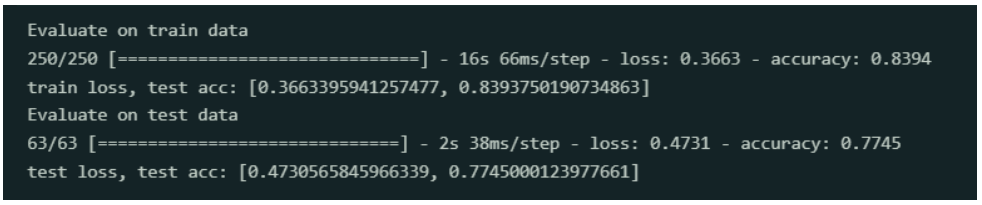
Después de aproximadamente 8 min de entrenamiento obtenemos los siguientes resultados y procedemos a validar el modelo con los datos de prueba y entrenamiento.

Entrenamiento:



Podemos observar que para este modelo obtenemos un accuracy del 83% con los datos de entrenamiento y un accuracy de 77% en los datos de prueba. Al tener una precisión menor en los datos de prueba, podemos ver como el modelo tiene overfitting.

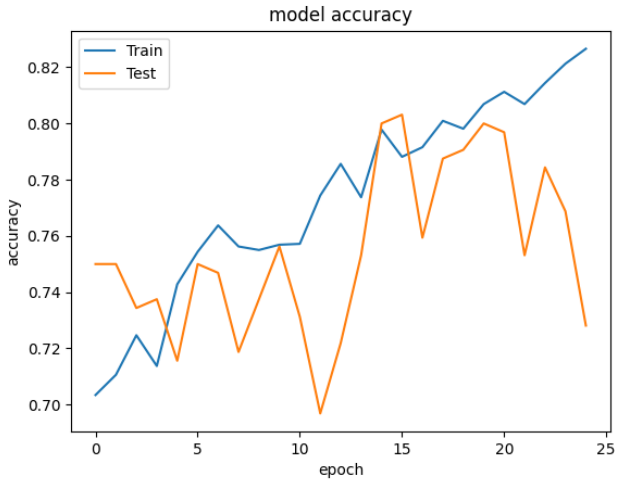
Validation:



Realizamos algunas gráficas de la precisión del modelo y los errores.

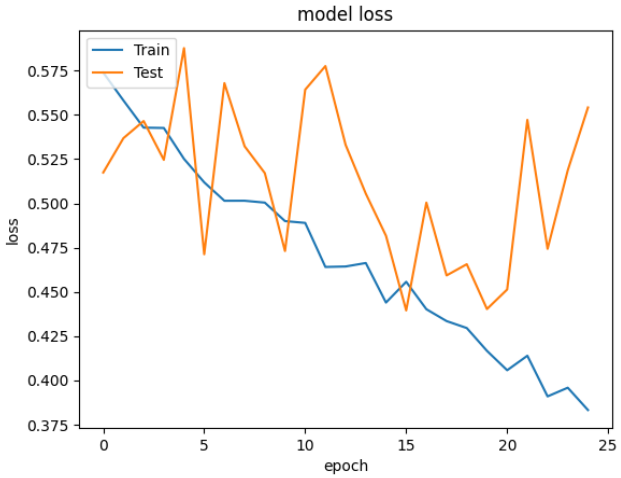
Podemos observar cómo efectivamente el modelo presenta una precisión muy dispersa en los datos de prueba.

Accuracy:



Para el cálculo del error podemos observar como igualmente tiene un comportamiento disperso para el test.

Loss:



Prueba del modelo:

probando el modelo con las imágenes sacadas de google los resultados son los siguientes:

- De las 7 imágenes de perros, 7 las detectó correctamente.
- De las 5 imágenes de gatos, 0 las detectó correctamente.

XII. Posibles mejoras

Como observamos que el modelo cuenta con overfitting podríamos ya sea disminuir las épocas del modelo, o aumentar la cantidad de datos para entrenamiento y prueba, o en su defecto agregar más capas profundas para que la red detecte más características en las imágenes.

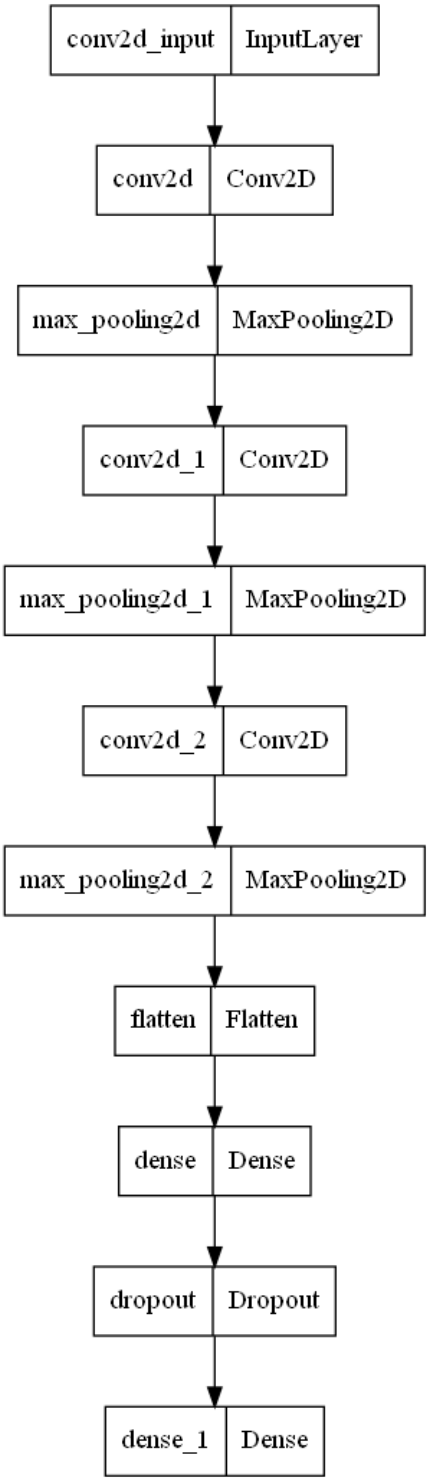
XIII. 2do Modelo

Para este 2do modelo lo que se hizo fue aumentar una capa convolucional profunda 2D de 64 nodos y una función de activación de ReLu, así como una capa de dropout la cuál establece aleatoriamente las unidades de entrada en 0,, lo que ayuda a evitar el overfitting. También se modificaron hiperparametros que se comentarán más a detalle en la sección de hiper parámetros.

A partir de este modelo se incluyó la capa de dropout en los modelos.

Estructura de la red:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 64)                802880

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 831,585
Trainable params: 831,585
Non-trainable params: 0
_________________________________________________________________
```


conv2d_input
InputLayer
conv2d
Conv2D
max_pooling2d
MaxPooling2D
conv2d_1
Conv2D
max_pooling2d_1
MaxPooling2D
conv2d_2
Conv2D
max_pooling2d_2
MaxPooling2D
flatten
Flatten
dense
Dense
dropout
Dropout
dense_1
Dense

Para este modelo se agregaron y modificaron algunos hiperparametros.

Primeramente para tener un buen cálculo de los pasos por época se dividió el número de datos de entrenamiento sobre la cantidad de batches por entrenamiento, en este caso 32, quedando **(número de datos)/(número de batches)**. Investigando un poco esto mejoraría el modelo.

Así mismo dividimos también los **validation_steps** sobre el número de datos de prueba sobre el número de batches.

Se añadió el parámetro de **workers** que básicamente es cuántos procesos se estarán corriendo en el modelo en paralelo, y en este caso se definieron 12 workers. Esto nos ayudó a que el entrenamiento fuera un poco más rápido.

También se añadió el parametro **max_queue_size** que nos ayuda a definir el tamaño máximo de la cola de entrenamiento interna que se utiliza para "precachear" muestras. En este caso definimos este valor máximo como 100.

Hiperparámetros:

```
history = classifier.fit(training_set,
    steps_per_epoch = 8000/32,
    epochs = 25,
    validation_data = test_set,
    validation_steps = 2000/32,
    workers=12,
    max_queue_size=100)
```

Entrenamiento:

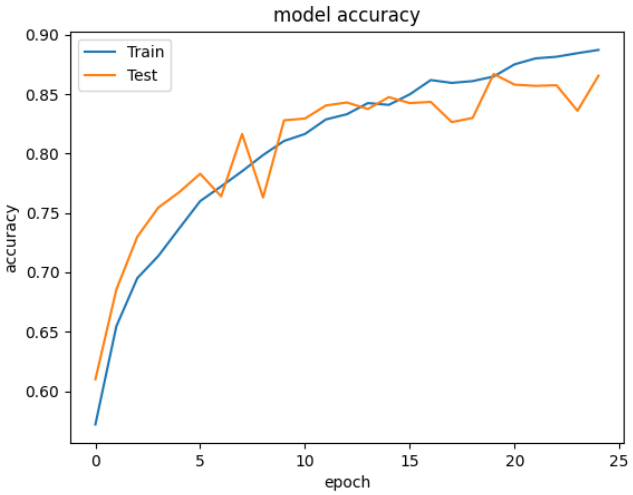
```
250/250 [==============================] - 15s 56ms/step - loss: 0.2863 - accuracy: 0.8801 - val_loss: 0.3733 - val_accuracy: 0.8570
Epoch 23/25
250/250 [==============================] - 14s 53ms/step - loss: 0.2751 - accuracy: 0.8815 - val_loss: 0.3620 - val_accuracy: 0.8575
Epoch 24/25
250/250 [==============================] - 13s 50ms/step - loss: 0.2712 - accuracy: 0.8845 - val_loss: 0.4039 - val_accuracy: 0.8360
Epoch 25/25
250/250 [==============================] - 14s 52ms/step - loss: 0.2615 - accuracy: 0.8873 - val_loss: 0.3529 - val_accuracy: 0.8655
```

Podemos observar en base a la validación de los datos de prueba y entrenamiento que sigue habiendo overfitting pero en base a las gráficas mostradas más abajo, el comportamiento es más normal para los datos tanto de prueba como entrenamiento.

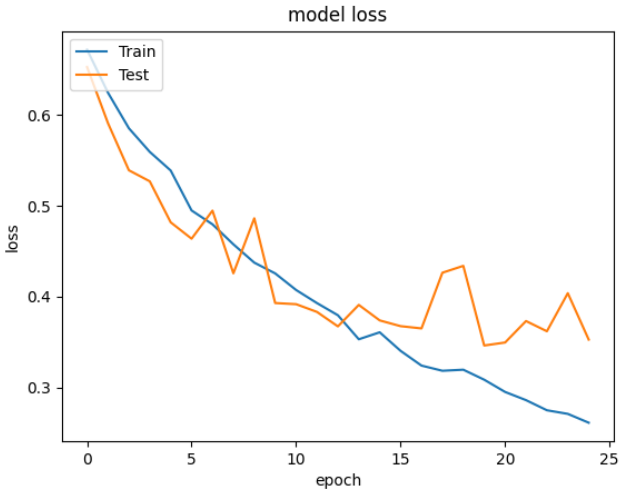
Validation:

```
Evaluate on train data
250/250 [==============================] - 31s 123ms/step - loss: 0.2320 - accuracy: 0.9056
train loss, test acc: [0.2320248931646347, 0.9056249856948853]
Evaluate on test data
63/63 [==============================] - 3s 40ms/step - loss: 0.3529 - accuracy: 0.8655
test loss, test acc: [0.3528532087802887, 0.8654999732971191]
```

Accuracy:



Loss:



Prueba del modelo:

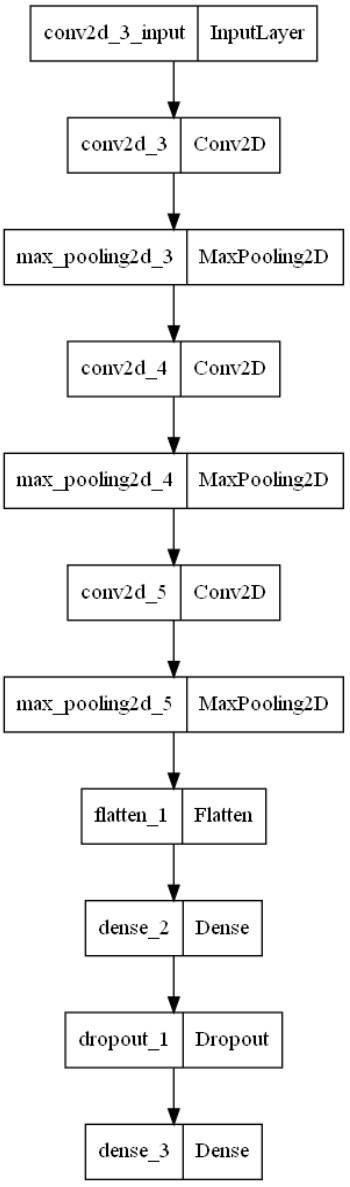
probando el modelo con las imágenes sacadas de google los resultados son los siguientes:

- De las 7 imágenes de perros, 4 las detectó correctamente.
- De las 5 imágenes de gatos, 5 las detectó correctamente.

XIV. Tercer Modelo

Para este modelo se utilizó el dataset completo, que consta de 20,000 imágenes de entrenamiento y 5000 para pruebas. Posiblemente utilizando más datos el modelo mejore en la precisión.

Estructura de la red:


conv2d_3_input
InputLayer
conv2d_3
Conv2D
max_pooling2d_3
MaxPooling2D
conv2d_4
Conv2D
max_pooling2d_4
MaxPooling2D
conv2d_5
Conv2D
max_pooling2d_5
MaxPooling2D
flatten_1
Flatten
dense_2
Dense
dropout_1
Dropout
dense_3
Dense

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 61, 61, 32)        9248

 max_pooling2d_4 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 12544)             0

 dense_2 (Dense)             (None, 64)                802880

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 831,585
Trainable params: 831,585
Non-trainable params: 0
_________________________________________________________________
```

Hiperparámetros:

```python
history = classifier.fit(training_set,
    steps_per_epoch = 20000/32,
    epochs = 25,
    validation_data = test_set,
    validation_steps = 2000/32,
    workers=12,
    max_queue_size=100)
```

Entrenamiento:

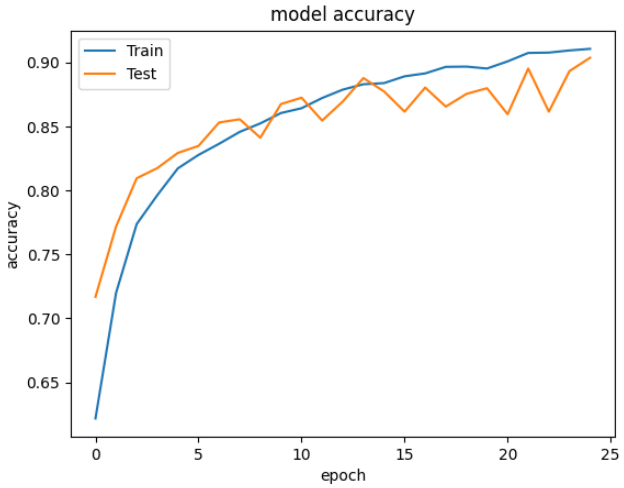
```
625/625 [==============================] - 29s 46ms/step - loss: 0.2391 - accuracy: 0.9010 - val_loss: 0.3572 - val_accuracy: 0.8596
Epoch 22/25
625/625 [==============================] - 29s 45ms/step - loss: 0.2256 - accuracy: 0.9075 - val_loss: 0.2763 - val_accuracy: 0.8953
Epoch 23/25
625/625 [==============================] - 35s 54ms/step - loss: 0.2231 - accuracy: 0.9078 - val_loss: 0.4004 - val_accuracy: 0.8616
Epoch 24/25
625/625 [==============================] - 35s 54ms/step - loss: 0.2220 - accuracy: 0.9095 - val_loss: 0.2748 - val_accuracy: 0.8934
Epoch 25/25
625/625 [==============================] - 33s 51ms/step - loss: 0.2168 - accuracy: 0.9107 - val_loss: 0.2421 - val_accuracy: 0.9038
```

Para este modelo si se obtuvo una precisión para el train de 92.2% y para el test de 89.7% con una diferencia de 2.5%, ya que la diferencia es baja, se puede decir que el modelo sigue teniendo un poco de overfitting pero cada vez los valores del accuracy de train y test son más cercanos entre sí. Por esta razón este modelo es el mejor hasta ahora.

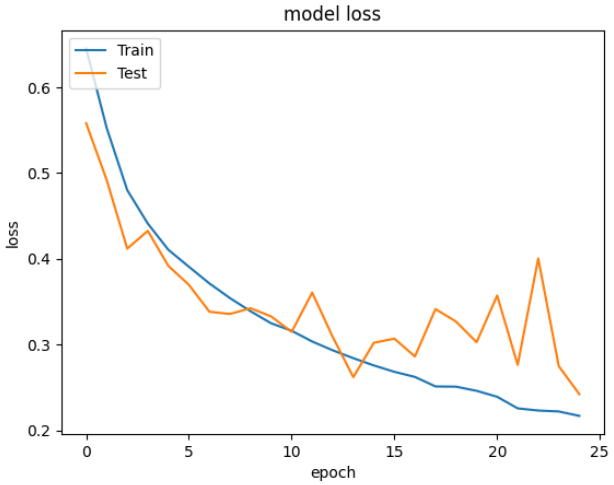
Validation:

```
Evaluate on train data
625/625 [==============================] - 76s 122ms/step - loss: 0.1743 - accuracy: 0.9282
train loss, test acc: [0.17427031695842743, 0.9282000064849854]
Evaluate on test data
157/157 [==============================] - 5s 34ms/step - loss: 0.2671 - accuracy: 0.8972
test loss, test acc: [0.2671208083629608, 0.8971999883651733]
```

Accuracy:



Loss:



Prueba del modelo:

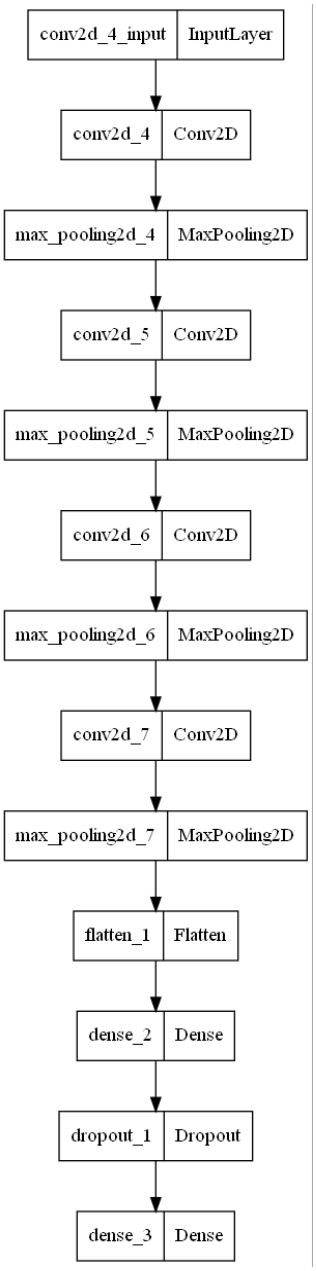
probando el modelo con las imágenes sacadas de google los resultados son los siguientes:

- De las 7 imágenes de perros, 1 las detectó correctamente.
- De las 5 imágenes de gatos, 2 las detectó correctamente.

XV. Cuarto Modelo

Se decidió realizar un último modelo añadiendo otra capa profunda de 32 nodos y una función de activación relu para ver si el modelo mejoraba.

Estructura de la red:


conv2d_4_input
InputLayer
conv2d_4
Conv2D
max_pooling2d_4
MaxPooling2D
conv2d_5
Conv2D
max_pooling2d_5
MaxPooling2D
conv2d_6
Conv2D
max_pooling2d_6
MaxPooling2D
conv2d_7
Conv2D
max_pooling2d_7
MaxPooling2D
flatten_1
Flatten
dense_2
Dense
dropout_1
Dropout
dense_3
Dense

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 61, 61, 32)        9248

 max_pooling2d_5 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d_6 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 12, 12, 64)        18496

 max_pooling2d_7 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 2304)              0

 dense_2 (Dense)             (None, 64)                147520

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 185,473
Trainable params: 185,473
Non-trainable params: 0
_________________________________________________________________
None
```

Hiperparámetros:

```
history = classifier.fit(training_set,
    steps_per_epoch = 20000/32,
    epochs = 25,
    validation_data = test_set,
    validation_steps = 2000/32,
    workers=12,
    max_queue_size=100)
```

Entrenamiento:

```
Epoch 21/25
625/625 [==============================] - 33s 52ms/step - loss: 0.2046 - accuracy: 0.9151 - val_loss: 0.2118 - val_accuracy: 0.9182
Epoch 22/25
625/625 [==============================] - 33s 51ms/step - loss: 0.1922 - accuracy: 0.9216 - val_loss: 0.2318 - val_accuracy: 0.9023
Epoch 23/25
625/625 [==============================] - 33s 52ms/step - loss: 0.1908 - accuracy: 0.9213 - val_loss: 0.2559 - val_accuracy: 0.9082
Epoch 24/25
625/625 [==============================] - 33s 51ms/step - loss: 0.1846 - accuracy: 0.9241 - val_loss: 0.2478 - val_accuracy: 0.8978
Epoch 25/25
625/625 [==============================] - 33s 52ms/step - loss: 0.1880 - accuracy: 0.9240 - val_loss: 0.2049 - val_accuracy: 0.9127
```

Validation:

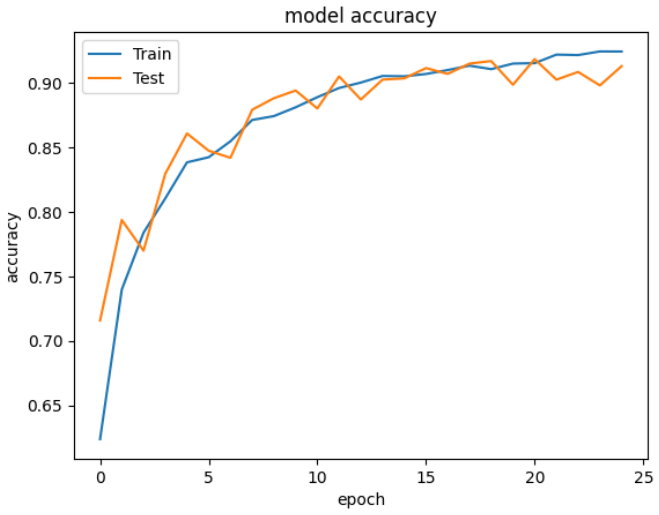
Podemos ver como la evaluación del modelo tanto en train como en test es bastante parecida con una diferencia de 1.55%. Este modelo ahora se convierte en el mejor modelo con un accuracy mayor y un overfitting muy bajo.

Efectivamente la capa profunda que se agregó ayudó a que el modelo tuviera un overfitting menor y por ende una mayor precisión.

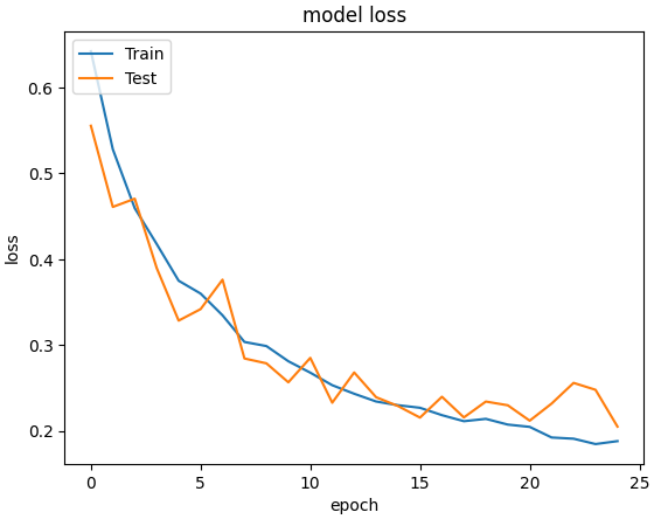
```
Evaluate on train data
625/625 [==============================] - 75s 120ms/step - loss: 0.1645 - accuracy: 0.9333
train loss, test acc: [0.1645161360502243, 0.9332500100135803]
Evaluate on test data
157/157 [==============================] - 6s 36ms/step - loss: 0.2069 - accuracy: 0.9178
test loss, test acc: [0.20693928003311157, 0.9178000092506409]
```

También podemos ver como el comportamiento del accuracy como el error es muy similar tanto en train como en test por lo que se considera un muy buen modelo.

Accuracy:



Loss:



Prueba del modelo:

probando el modelo con las imágenes sacadas de google los resultados son los siguientes:

- De las 7 imágenes de perros, 3 las detectó correctamente.
- De las 5 imágenes de gatos, 5 las detectó correctamente.

XVI. Conclusiones

Al desarrollar diferentes modelos de cnn modificando los hiperparámetros, añadiendo más capas profundas y añadiendo más datos de entrenamiento se pudo mejorar el modelo con muy poco overfitting y con gran precisión.