Nombre: Lilian Rebeca Carrera Lemus.
Carnet: 20008077
Curso: Text Mining and Image Recognition

Laboratorio No. 3 - Reporte

Descripción: Para este laboratorio deberá utilizar Tensorflow/Keras para desarrollar un clasificador binario el cual permita realizar la clasificación de las observaciones proporcionadas (dataset de gatos y perros).

Como métrica de referencia deberá generar un accuracy mínimo de 0.8, la idea del laboratorio es que usted entienda cómo funcionan las diferentes capas que forman una red neuronal convolucional y como esta puede ajustarse o modificarse para mejorar los resultados de una predicción.

Entregables: Para este laboratorio deberá entregar el código realizado y un documento sencillo donde explique los pasos que implementó para su red convolucional, tanto la fase de preprocesamiento como la red neural completamente conectada. En este reporte deberá incluir todas las consideraciones de configuración e hiperparámetros que usted seleccionó para producir su mejor resultado. Por ultimo deberá incluir sus resultados comparando el set de prueba contra los resultados de la red.

Pasos para la Resolución del Problema

Para este problema, el código se encuentra en el archivo Lab3-CNN_Clasificador.ipynb, el cual es el notebook de Python que contiene la solución completa, así mismo en este notebook también se encontrarán las pruebas de predicción para las imágenes proporcionadas.

El objetivo es diseñar un algoritmo de aprendizaje automático, para un problema de clasificación entre dos especies animales como lo son perros y gatos. Dada la naturaleza del problema, se debe hacer uso de redes neuronales convolucionales (CNN).

El modelo de red implementado está divido en dos partes, la primera es la red convolucional, que cuenta con dos capas convolucionales, que realizan la extracción de features y detectan patrones en las imágenes y la segunda es la capa completamente conectada (fully connected), la cual se encarga de realizar la clasificación con base a los patrones detectados en la red convolucional.

Los pasos que se llevaron a cabo para resolver el problema planteado son:

1. ***Definición del Modelo de Red Convolucional***
    a. ***Capas convolucionales***

    Para agregar todas las capas del modelo inicialmente se definió una estructura base, la cual se ejecutará en forma secuencial.

    Posteriormente se procedió a la agregación y definición de dos capas convolucionales para la extracción de características, las cuales detectan patrones en las imágenes de entrada, entre más capas convolucionales hayan más patrones diferentes serán detectados en las imágenes, pero hay que tener en cuenta que no debemos excedernos porque sino nuestro modelo puede sufrir un sobreajuste debido al aumento del capacity o complejidad del modelo.

    La primera capa convolucional tiene como parámetros, la cantidad de filtros = 32, lo cual significa que se realizarán 32 convoluciones. El tamaño de los filtros es de 3*3, siguiendo el estándar de números impares, el tamaño de las imágenes de entrada se estableció en 64*64*3, debido a que son imágenes a color. Como función de activación se seleccionó ReLu, para que devuelva solo números positivos y cambié los números negativos por 0, esto debido a la naturaleza del problema (ya que son imágenes). Por default si no definimos el padding utilizará “same”, por lo cuál el tamaño de las imágenes filtradas resultantes será el mismo que el de las imágenes de entrada.

    Se agregó una segunda capa convolucional para robustecer el modelo, en la cuál solo cambió la cantidad de filtros y se redujo a 24.

    b. ***Capas de MaxPooling***

    Para cada una de las capas convolucionales, se agregó su respectiva capa de MaxPooling, con el objetivo de reducir la dimensionalidad de la imagen resultante de la capa convolucional con la mejor versión de la misma, es decir con los valores máximos, o patrones en donde la activación fue mayor.

    El parámetro definido para estas capas es el pool_size, el cual es una tupla de 2 enteros, y define el tamaño de ventana sobre el cual tomar el máximo. En este caso (4, 4), por lo cual tomará el valor máximo sobre una ventana de agrupación de 4x4.

c. ***Capa de Flattening***

El objetivo de esta capa es aplanar el resultado de maxpooling anterior y lo vuelve un vector para que este sea la entrada de la siguiente red neural completamente conectada. Esta capa no tiene parámetros.

2. ***Definición Red Neural completamente Conectada***

Esta red cuenta con dos capas, la capa completamente conectada y la capa de salida, las cuales se encargan de realizar la clasificación con base a los patrones detectados en la red convolucional.

a. ***Capa Fully Connected***

En la capa completamente conectada no definimos un tamaño de entrada fijo, ya que no es la primera capa, por lo cual tomará como entrada el tamaño del vector obtenido como resultado de la capa de flattening. Los parámetros definidos en esta capa son la cantidad de neuronas que tendrá (units), que en este caso son 128 y la función de activación elegida fue ReLu. Se definió está función de activación, ya que por la naturaleza del problema necesitamos que nuestra salida no tenga números negativos.

Para definir la cantidad de neuronas se puede aplicar la siguiente fórmula (40 entradas + cantidad de salidas)/2 o bien trabajar con potencias de 2.

b. **Capa de Salida**

La definición de esta capa depende del problema a resolver. Como en este caso la salida es clasificación binaria solo necesitamos una neurona de salida.

Los parámetros configurados son, la función de activación elegida fue Sigmoid, ya que, esta es mejor en este caso porque es más sencillo de interpretar la salida como una probabilidad. Si el problema fuera de regresión podría usarse ReLu o si fuera de clasificación multiclase podría ser softmax.

3. ***Compilar la Red***

En este paso es donde definiremos los elementos de cómo vamos a resolver el problema, por lo cual los parámetros de configuración son, el optimizador que se utilizará para resolver el problema de minimización de la función de costo, la función de costo que se usará (o función de error), la cual se usará para evaluar si mi solución es adecuada y las métricas, las cuales nos indican como está funcionando el algoritmo.

Para el optimizador se eligió “Adam”, ya que este se recomienda para este tipo de problemas, por lo que es más eficiente que GD. Debido a que es un problema de clasificación binaria, como función de costo se utilizará "binary_crossentropy" y la métrica de evaluación en este caso será "accuracy", de acuerdo a lo indicado en el problema.

4. ***Preprocesamiento de las imágenes de Entrada***

   El siguiente paso es formatear los datos para el entrenamiento y las pruebas.

   a. ***Transformación de imágenes***

      Las transformaciones en las imágenes se refieren a hacer variaciones en dichas imágenes para que sean mejores después para predecir y reducir el overfitting, es decir enriquecer la entrada. Por lo cual en este caso se definen un rango para los giros aleatorios y otro para los zooms aleatorios, un reescalado de las imágenes (números entre 0 y 1) y como verdadero si queremos que las imágenes se volteen horizontalmente.

   b. ***Formatear las imágenes***

      El objetivo de este paso es, como su nombre lo indica, darle el mismo formato a todas las imágenes, ya que todas tienen diferentes tamaños y características. Los parámetros son el tamaño de las mismas que se definió como 64*64 de acuerdo a lo definido en la red también como entrada, un tamaño de batch de 32 y un modo de clase = binario, ya que el problema es binario.

5. ***Configuración del Entrenamiento***

   El siguiente paso es definir los mejores parámetros para el entrenamiento, de modo que se cumpla con que el error baje y el acurracy esté de acuerdo a los límites definidos en el problema.

   Se realizará el entrenamiento y validación de una vez, y las configuraciones para el entrenamiento son: primero se definen los set de datos para entrenamiento ya para validación proporcionados y en este caso se definieron 5 epochs para el entrenamiento, usando 8000 imágenes en cada uno y para la validación son los mismos 5 epochs, usando 2000 imágenes en cada uno.

   Con esta elección de parámetros se obtiene un acurracy de entrenamiento 94.36, el cual es mayor al solicitado y un acurracy de validación de 84.

Configuración del Entrenamiento

[24]:
```
#El siguiente paso es definir los mejores parámetros para el entrenamiento, de modo que se cumpla con que el error baje
#y el acurracy esté de acuerdo a los límites definidos en el problema
#Se realizará el entrenamiento y validación de una vez
#Para el entrenamiento en este caso se definieron 5 epochs, usando 8000 imágenes en cada uno
#Para La validación son los 5 epochs, usando 2000 imágenes en cada uno.
cnn.fit(train_set,
        steps_per_epoch=8000,
        epochs=5,
        validation_data=test_set,
        validation_steps=2000,
        verbose=1)
```

```
Epoch 1/5
8000/8000 [==============================] - ETA: 0s - batch: 3999.5000 - size: 32.0000 - loss: 0.4336 - acc: 0.7930WARNING:tensorflow:From C:\Users\Rebeca\Anaconda3\lib\site-packages\tensorflow\python\keras\engine\training_v1.py:2048: Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
8000/8000 [==============================] - 935s 117ms/step - batch: 3999.5000 - size: 32.0000 - loss: 0.4336 - acc: 0.7930 - val_loss: 0.4151 - val_acc: 0.8125
Epoch 2/5
8000/8000 [==============================] - 959s 120ms/step - batch: 3999.5000 - size: 32.0000 - loss: 0.2987 - acc: 0.8692 - val_loss: 0.4167 - val_acc: 0.8265
Epoch 3/5
8000/8000 [==============================] - 978s 122ms/step - batch: 3999.5000 - size: 32.0000 - loss: 0.2225 - acc: 0.9061 - val_loss: 0.5286 - val_acc: 0.8201
Epoch 4/5
8000/8000 [==============================] - 1066s 133ms/step - batch: 3999.5000 - size: 32.0000 - loss: 0.1731 - acc: 0.9282 - val_loss: 0.5399 - val_acc: 0.8299
Epoch 5/5
8000/8000 [==============================] - 975s 122ms/step - batch: 3999.5000 - size: 32.0000 - loss: 0.1400 - acc: 0.9436 - val_loss: 0.6232 - val_acc: 0.8390
```

t[24]: `<tensorflow.python.keras.callbacks.History at 0x21f5b1ec188>`

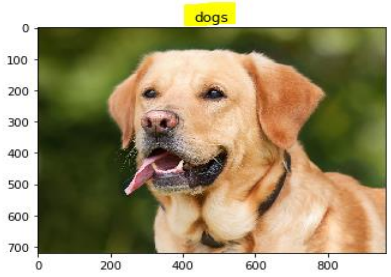
Con estos parámetros en el entranamiento se logró un acurracy de 94.36, el cuál es mayor al indicado.

6. *Predicciones de Prueba*

Para realizar las predicciones de prueba individuales, se definió una función en la cual se lee la imagen de entrada indicada, se determina un tamaño de 64*64 y se obtienen las clases predichas por el modelo entrenado en el paso anterior para después mostrar los resultados.

Probamos el modelo, con las imágenes proporcionadas para predicción

[35]:
```
predecir('single_prediction/cat_or_dog_1.jpg', cnn)
```



[36]:
```
predecir('single_prediction/cat_or_dog_2.jpg', cnn)
```