MODELO 1

ESTRUCTURA BASE DE DATOS → tenemos un dataset con dos carpetas, una de imágenes de “testeo” y otra carpeta de entrenamiento. La primera carpeta tiene imágenes desordenadas de los 10 tipos de animales que hay y contiene alrededor del 20 por ciento del conjunto de imágenes del dataset. Por otro lado, tenemos la carpeta de entrenamiento, clasificada en 10 carpetas de los 10 tipos diferentes de animales y que tiene el 80 por ciento restante de las imágenes del dataset.

PREPROCESADO DE LOS DATOS → creamos 10 bucles for, uno por cada clase de animal, que recorra todo el archivo y guardar todas las imágenes en un array X, con la función append, que agregara las imágenes al final de la lista. Para prevenir errores, nos aseguramos de que todas las imágenes se pasan a RGB y que no haya problemas por si hay imágenes con canales distintos a 3. Antes de guardarlas en el array, ponemos las imágenes a 64 pixeles. Luego tenemos otro array y, encargado de diferenciar el tipo de clase animal, pensé en hacerlo con la funcion to_categorical, pero decidí que esta opción era más fácil para la posterior ejecución del resto del código. Al final normalizamos los datos de entrada.

Luego cargamos los datos del test, como no se separa por categorías, hacemos todo en un único for. X_final_img guardara las fotos de la carpeta test y X_final_test guardará los id de las respectivas fotos.

DEFINICIÓN DEL MODELO

Una vez cargado los datos, creamos la red y lo primero que hacemos es aplicar 16 kernel de 3*3 y después aplicar la función relu, transformando los valores negativos introducidos en valores positivos. Recordemos que la forma es de 64, tanto de alto como de ancho, y 3 canales,RGB. Luego aplicamos max pooling, lo más habitual es aplicarlo en un tamaño de 2*2. Con esto recorreremos cada una de las 16 imágenes e iremos preservando el valor más alto de los 4 pixeles. Al usar un 2*2, la imagen es reducida a la mitad.

Aplicamos este mismo proceso con otra capa, esta vez con 32 kernels y otra más de 64 kernels, con sus respectivos max pooling. En la transición de la capa convolucional a la capa completamente conectada, utilizamos la función Flatten. Con esto “aplanamos” la entrada, es decir, unidimensionalizar la entrada multidimensional.

La última es una capa softmax de 10 neuronas y devolverá una matriz de 10 valores de probabilidad que representan la probabilidad de que la imagen pertenezca a cada una de ellas.

Por último utilizamos la función summary para ver la arquitectura del modelo.Y la función compile, en la que especificamos la función de pérdida, en esta caso categorical_crossentropy(los valores de la etiqueta están vectorizados, un índice del vector es 1 y el resto 0), el optimizador, “adam”(su learning rate por defecto es de 0.001,

podríamos bajarlo mas pero tambien nos arriesgaremos a que el modelo fuera más lento y peor), y la métrica que es accuracy, que nos da la tasa de aciertos de la red.

ENTRENAMIENTO DEL MODELO

En la función fit, primero cogemos el array de train_X con las imágenes y el train_y con el array de las posibles categorías. Después especificamos nuestros datos de validación que necesitaremos después para estudiar las estadísticas y que definimos anteriormente con la función train_test_split en la que separamos las imágenes para test y para entrenamiento.
El batch_size indica el número de datos que usaremos para cada vuelta de los parámetros del modelo y los epochs definimos el número de veces que usaremos los datos.
En este caso para el primer modelo, he escogido un batch_size relativamente grande para que en posteriores modelos lo vayamos ajustando y bajando, pues de esa forma obtenemos mejores resultados.

RESULTADOS DEL ENTRENAMIENTO DEL MODELO

La mejor forma para estudiar los resultados son los gráficos de las estadísticas. El primer gráfico estudia las pérdidas que tiene el modelo, en el eje “y” tenemos las pérdidas, y en el eje “x” las épocas. Podemos ver que según vamos entrenando el modelo, el número de pérdidas baja. Podemos ver algún pico de subida de las pérdidas en la validación, pero por lo general, no hay mucha diferencia entre la validación y el entrenamiento, con lo que no podemos ver ningún problema de overfitting ni tampoco de underfitting ya que el modelo está bien entrenado. Por otra parte tenemos la gráfica de accuracy, donde vemos que también la validación y el entrenamiento van a la par y según avanzan las épocas el accuracy sube también, otra vez vemos picos, en este caso de bajada ya que tratamos con la accuracy, pero también podemos descartar un problema de overfitting o de underfitting.

CONCLUSIONES

Este primer modelo es un modelo bastante capaz teniendo en cuenta que es un modelo básico las gráficas obtenidas son bastante satisfactorias. El paso que en mi opinión es el más importante es la relación que existe entre el learning rate y el batch_size y que mejorandola conseguiremos que los valores de validación y entrenamiento mejoren y sean más próximos, aunque podamos estropear otra cosa, como el accuracy del modelo.

MODELO 2

ESTRUCTURA BASE DE DATOS → tenemos un dataset con dos carpetas, una de imágenes de “testeo” y otra carpeta de entrenamiento. La primera carpeta tiene imágenes desordenadas de los 10 tipos de animales que hay y contiene alrededor del 20 por ciento del conjunto de imágenes del dataset. Por otro lado, tenemos la carpeta de entrenamiento, clasificada en 10 carpetas de los 10 tipos diferentes de animales y que tiene el 80 por ciento restante de las imágenes del dataset.

PREPROCESADO DE LOS DATOS → creamos 10 bucles for, uno por cada clase de animal, que recorra todo el archivo y guardar todas las imágenes en un array X, con la función append, que agregara las imágenes al final de la lista. Para prevenir errores, nos aseguramos de que todas las imágenes se pasan a RGB y que no haya problemas por si hay imágenes con canales distintos a 3. Antes de guardarlas en el array, ponemos las imágenes a 64 pixeles. Luego tenemos otro array y, encargado de diferenciar el tipo de clase animal, pensé en hacerlo con la funcion to_categorical, pero decidí que esta opción era más fácil para la posterior ejecución del resto del código.

Luego cargamos los datos del test, como no se separa por categorías, hacemos todo en un único for. X_final_img guardara las fotos de la carpeta test y X_final_test guardará los id de las respectivas fotos.

DEFINICIÓN DEL MODELO

Quise añadir otro layers adicional, Dropout, que sirve para regular el overfitting del modelo, esto lo aplico cada vez que aplicamos un MaxPooling. En cuanto a las capas ocultas he vuelto aplicar otra vez 3 pero esta vez cada una tiene el doble de kernels para intentar ajustar la precisión del modelo.

Por último he investigado la relación del batch_size con el learning rate, mi primera opción fue bajar solamente el batch_size pero me di cuenta de que esto no solo no mejoraba el modelo si no que empeoraba su funcionamiento respecto al primer modelo. Con lo que decidí bajar tanto el batch_size como el learning rate y conseguir una mayor paridad en las gráficas. Teniendo en cuenta que el accuracy entre un modelo y otro (no val_accuracy) sube casi 0.2 de puntos que es bastante para seguir manteniendo esa paridad.

ENTRENAMIENTO DEL MODELO

En cuanto al entrenamiento del modelo decidí optar por los callbacks, estuve barajando la posibilidad de añadir varios parámetros de estos mismos pero conseguí que el modelo funcionara peor con lo que añadí el más importante para mi, el Early Stopping. Su función es parar la función cuando el programa ve que la velocidad con la que aprende el programa se ve reducida. Esto lo marcamos gracias a uno de sus parámetros, patience, en el que decidimos cual es el máximo de épocas que el programa toma antes de rectificar.

RESULTADO DE ENTRENAMIENTO DEL MODELO

La mejor forma para estudiar los resultados son los gráficos de las estadísticas. El primer gráfico estudia las pérdidas que tiene el modelo, en el eje “y” tenemos las pérdidas, y en el eje “x” las épocas. Podemos ver que según vamos entrenando el modelo, el número de pérdidas baja y lo más importante, no hay mucha diferencia entre la validación y el entrenamiento, con lo que no podemos hablar de un problema de overfitting o de underfitting. Por otra parte tenemos la gráfica de accuracy, donde vemos que también la validación y el entrenamiento van a la par y según avanzan las épocas el accuracy sube también, con lo que podemos descartar también un problema de overfitting o de underfitting aparente. Podemos notar que al final de las dos gráficas las líneas se separan pero no es un problema a destacar.

Una vez estudiado más a fondo le puse un numero mucho mas alto de épocas para ver un rendimiento a gran escala y me di cuenta que que si se producía overfitting ya que la recta de validación se queda más arriba que la de entrenamiento en las perdidas y mas baja que la de entrenamiento en la accuracy. El lea

CONCLUSIONES

La conclusión final es que a pesar de buscar muchas formas de mejorar el modelo, el primer modelo, al ser más simple es más eficaz.

Al probar varias combinaciones de modelos, pensé que para que el modelo mejore necesitaba bajar el learning rate pero eso empeoraba mi accuracy. Así que probé a bajar el batch_size para ver si mejoraba y conseguí perfeccionarlo hasta el punto que equipararse con el accuracy del primer modelo. Pero cuando empecé a tocar el resto de valores como los callbacks el modelo perdió eficacia.La principal diferencia entre el primer modelo y el segundo se ve en la simpleza, y seguramente al no haber utilizado mejor los parámetros utilizados en el segundo modelo.