Clasificación automática de arquitecturas

Isabel Arrans Vega
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
isaarrveg@alum.us.es | isabelarrans@gmail.com

Matthew Bwye Lera
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
matbwyler@alum.us.es | xamtehwt@gmail.com

***Resumen*— Con este trabajo, pretendemos diseñar una inteligencia artificial capaz de clasificar rápidamente edificios de entre 25 clases distintas de estilos arquitectónicos, haciendo uso de redes neuronales.**

Nuestras redes neuronales tienen distintos puntos fuertes y débiles; algunas son capaces de entrenarse muy rápidamente, pero tienen sobreajuste, lo que lo hace menos confiable para cualquier tipo de imagen, mientras que otras tardan mucho en entrenarse, pero tienen menos problemas con el sobreajuste, lo que lo hace más fiable.

Palabras Clave—Inteligencia Artificial, Redes Neuronales, Convolución, Sobreajuste, Conjunto de entrenamiento, Aprendizaje, Clasificación

I. INTRODUCCIÓN

Este proyecto se realiza en un ámbito académico, en la Universidad de Sevilla. En el grado de Informática - Ingeniería del Software, asignatura de Inteligencia Artificial, los alumnos debemos agruparnos por parejas, solicitar algún proyecto y realizarlo y documentarlo. En este caso, realizaremos un trabajo para la clasificación de imágenes mediante redes neuronales convolucionales.

Dado nuestro conocimiento en la materia, se espera que nos documentemos en internet acerca de la resolución de esta tarea y que se referencie de forma apropiada. Las expectativas son que aprendamos a desarrollar software que haga uso de inteligencias artificiales y que adquiramos las competencias necesarias para poder hacer más trabajos de este tipo en el futuro.

En este trabajo, haremos uso de redes neuronales; esto son una serie de algoritmos y operaciones que tratan de simular una red neuronal humana, utilizando la teoría de grafos para representarse. El objetivo de nuestro código es ser capaz de entrenarse con un conjunto de entrenamiento y luego clasificar de forma apropiada una serie de imágenes de edificios entre 25 categorías que representan los estilos arquitectónicos. Proporcionaremos una serie de redes neuronales con parecidos entre sí y distinciones entre sí, esperando que una buena variedad de formas de análisis.

Como nuestras redes neuronales trabajan sobre análisis de imágenes, deben hacer uso de operaciones de convolución, que facilitan el análisis y la comparativa entre imágenes. Por este motivo, las redes neuronales que diseñemos recibirán el nombre de redes neuronales convolucionales.

II. PRELIMINARES

En esta sección haremos una breve introducción de las técnicas que hemos empleado, y también mencionaremos una serie de trabajos relacionados que nos han ayudado a alcanzar las soluciones que hemos propuesto.

A. Métodos empleados

Los métodos empleados son los siguientes:

- Redes neuronales convolucionales: Se trata de un método que recibe parámetros de entrada, normalmente en forma de imágenes en caso de ser una red neuronal convolucional, luego estos parámetros influyen sobre una serie de operaciones que se influyen las unas a las otras, que forman las capas intermedias, y, por último, recibimos una serie de parámetros de salida, que nos indicarán la respuesta del algoritmo. Este método se caracteriza por tener capacidad de aprendizaje dado un conjunto de entrenamiento, que le permite luego saber clasificar cualquier dato de entrada basándose en su aprendizaje previo.
- Convolución: Es un método de filtrado de imagen [1], que altera los píxeles de las imágenes en función de los otros píxeles que existen a su alrededor, que va a facilitar a nuestro algoritmo la clasificación de imágenes, reduciendo el nivel de detalles en que se tendrá que fijar. Se aplicará este algoritmo varias veces para hacer distintos análisis de las imágenes, donde cada una de las veces que apliquemos convolución, se reducen los detalles de las imágenes.
- Aumento de datos: Un inconveniente típico de este tipo de proyectos es el sobreajuste, es decir, que el algoritmo clasifique muy bien lo que recibe del conjunto de entrenamiento, pero no consigue clasificar bien elementos externos a él. Para mejorar esta situación, decidimos aumentar el número de datos que compara. La forma más fácil que encontramos para esto fue replicar y rotar las imágenes que el algoritmo analiza, y que analice también estas versiones [2] [3]. El principal inconveniente de hacer esto es que el entrenamiento es más largo, ya que tiene que trabajar con más imágenes.
- Dropout: Similar al método explicado antes, este se utiliza para reducir el sobreajuste. Este método lo que hace es descartar una proporción concreta de las

imágenes en cada época del entrenamiento [2] [4], que ayuda a que el entrenamiento utilice distintos datos de entrenamiento en cada una de las épocas, haciéndolo más variado y evitando que se sobreajuste al conjunto de entrenamiento.

B. *Trabajos Relacionados*

Aquí recorreremos un poco los distintos trabajos que hemos encontrado que nos han sido de ayuda.

- Documentación de la asignatura de Inteligencia Artificial.
- Práctica 2 de la asignatura de Inteligencia Artificial.
- Tutorial de clasificación de imágenes de la página principal de Tensorflow.
- Tutorial de clasificación de imágenes de la página principal de Keras.
- Documentación de los métodos de Tensorflow y Keras.

III. Metodología

Al comienzo de nuestro trabajo, partimos de la base que teníamos tras haber realizado la Práctica 2 de la asignatura. Observamos, en ella, los pasos necesarios para poder entrenar una red neuronal y repasar conceptos teóricos que iban a ser necesarios durante el transcurso del trabajo.

Por otro lado, leímos en profundidad lo que se esperaba que construyéramos, ya habiendo repasado los conceptos. Sabiendo qué objetivos debíamos alcanzar y teniendo una primera idea de los pasos necesarios para ello, comenzamos a idear qué estructura podrían tener las redes empleadas en los modelos que nos pedían. Todo esto aun desconociendo cómo trabajar con imágenes.

En primera instancia decidimos que dos modelos claramente distintos podían diferenciarse en su capa final, uno que tuviera como capa de salida un nodo por estilo arquitectónico posible y otro con un único nodo que, según el rango de valores que tomara, perteneciera a un estilo u otro. Esta idea quedó descartada más adelante debido a que no encontramos la manera de llevarla a cabo, por lo que decidimos diferenciar los distintos modelos mediante sus capas ocultas, aumentando en mayor o menor medida los parámetros que recibieran. De esta forma, sería posible obtener un mayor número de modelos, controlando mejor la diferencia entre ellos.

Una vez empezamos a trabajar, nos dimos cuenta de que, al trabajar con imágenes y no con ficheros csv, de la práctica no podíamos extraer cómo trabajar con imágenes, por lo que empezamos a documentarnos acerca de la librería de Keras y Tensorflow. Por suerte, pudimos encontrar varias guías de cómo preprocesar las imágenes en distintas secciones de las documentaciones. Analizamos el código de los diferentes tutoriales, los contrastamos y pudimos observar los pasos comunes que se realizaban en ellos. Adaptamos estos pasos al preprocesado de nuestras imágenes.

Para poder procesar las imágenes y dar sus clasificaciones en función al directorio de carpetas que se nos proporcionó (en nuestro caso, estilos arquitectónicos) definimos el directorio raíz en el que se encontraban las imágenes en nuestro repositorio de Google Drive y aplicamos 2 veces funciones de preprocessing de la librería keras [1], una para el conjunto de prueba y otra para el conjunto de entrenamiento. Definimos una semilla de manera arbitraria y determinamos que el 25% de los datos formarían el conjunto de validación y, el 75% restante, el de entrenamiento.

Para comprobar que las imágenes se introducían correctamente, mostramos por pantalla las imágenes haciendo uso de la librería matplotlib. En concreto, usando la función pyplot para imprimir por pantalla algunas fotos del conjunto de entrenamiento con su correspondiente clasificación como etiqueta. Esta clasificación la obteníamos mediante el elemento iésimo de la lista de los nombres de las clases del conjunto de entrenamiento.

En poco tiempo, probando con una red simple que se nos mostraba en uno de los tutoriales y adaptándolo a nuestro código, obtuvimos unos primeros resultados.

Fig. 1. Gráfica que representa la efectividad del algoritmo para el conjunto de entrenamiento y para el conjunto de validación.

Como se ve en la Figura 1, nuestra primera versión no fue un éxito. Con sólo 6 épocas, alcanzaba una buena precisión para el conjunto de entrenamiento, pero no para el conjunto de validación. Esto quiere decir que cada vez clasificaba mejor los elementos del conjunto de entrenamiento, pero no pasaba igual con los del conjunto de validación, por lo cual, este algoritmo no funcionaría muy bien con imágenes de edificios que se encuentren fuera del conjunto de entrenamiento.

Se presentó entonces la mayor dificultad que hemos tenido durante el desarrollo del trabajo: encontrar una red que, tras su debido entrenamiento, clasificara lo mejor posible los elementos del conjunto de validación. Para lo que tuvimos que analizar con especial atención las estructuras de las redes neuronales que

trabajan con imágenes y en qué afectaba la modificación de cada una de sus capas.

El motivo por el que clasificaba casi con el 100% las imágenes del conjunto de entrenamiento, pero no era capaz de hacerlo correctamente con nuevas imágenes, se debía al sobreajuste que se estaba produciendo. Por tanto, para alcanzar un mejor objetivo, debíamos corregirlo. Para ello, nos informamos mediante los distintos tutoriales que encontramos, consultando funciones y conceptos que no entendíamos en la documentación de keras.

Un primer intento de aumentar la efectividad de clasificación del conjunto de validación, corrigiendo el sobreajuste, fue añadir un tipo de capa de la librería denominado dropout. La primera vez que introdujimos esta capa, decidimos que su ratio de eliminación fuera del 0,2. Los cambios no eran demasiado notorios, pero había una ligera mejoría en la clasificación del conjunto de validación. Decidimos que no era lo suficientemente bueno aún.

Lo siguiente que hicimos para disminuir el sobreajuste fue añadir datos adicionales, ya que podría estar siendo causado por tener un conjunto de imágenes demasiado pequeño. Para la generación de nuevas imágenes, recorremos cada una de las imágenes del conjunto de entrenamiento original y creamos imágenes que eran copias de esta, pero aplicándose una ligera rotación aleatoria. Estas imágenes conformarían una nueva capa en la red, llamada data_augmentation.

Este aumento de datos fue un cambio muy significante. A partir de ahí, la clasificación del conjunto de entrenamiento disminuyó su ritmo de crecimiento, necesitando ahora muchas más etapas de entrenamiento para clasificar correctamente un alto porcentaje de este. Sin embargo, no habíamos conseguido mejorar demasiado el conjunto de validación, cuyas clasificaciones acertaban tan solo el 30-40%

Buscando mejorar la clasificación del conjunto de validación, ahora con una considerable reducción del sobreajuste, fuimos haciendo diferentes modificaciones a las distintas capas de la red, desde el orden en el que aparecían hasta la variación de parámetros y eliminación de capas. De esta forma, además, tendríamos cada vez más claras las funciones de cada una de las capas y los efectos que podrían producir sus alteraciones, para poder emplear estos cambios de la manera óptima.

Para poder organizar mejor las pruebas de los diferentes cambios que fuimos produciendo, entendimos la estructura de nuestras redes neuronales de la siguiente manera:

1) Capa de entrada de datos: imágenes a clasificar

2) Capas de preprocesado de datos: donde añadimos las nuevas imágenes extra generadas por rotación. Compuesta por dos capas.

*a) Capa de aumento de datos:*Donde se añaden los nuevos datos generados por rotación de imágenes.

b) Capa de reescalado: Donde normalizamos los valores de las imágenes de 0-255 a 0-1, para poder trabajar adecuadamente con ellas.

3) Capas de convoluciones: que sirven para similarizar el tono y color de los píxeles de las distintas imágenes. Está compuesta por pares de capas formados por una capa Conv2d y una capa MaxPooling2D. A mayor número se pase por parámetro del primer tipo de capa, más convoluciones se realizarán, y por tanto, menores serán los detalles.

4) Capa Dropout: para eliminar algunas salidas de las capas de convoluciones según su ratio indicado.

5) Capas finales: habiendo ya procesado los detalles de las imágenes y descartado algunos datos para evitar sobreajuste, se adaptan al formato de la imagen mediante una capa Flatten opera con los diferentes resultados mediante una capa dense, cuyos nodos están conectados con cada uno de los nodos de la capa anterior.

6) Capa de salida: capa final que cuenta por un nodo por cada estilo arquitectónico, de forma que el nodo con mayor valor determina el estilo arquitectónico de la imagen de entrada.

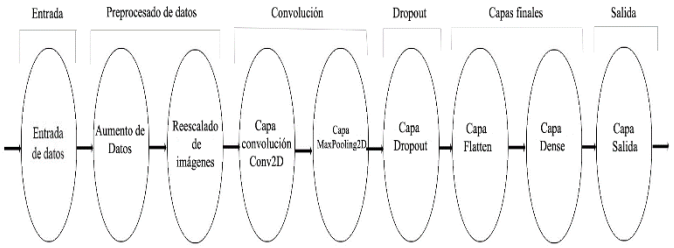

Fig. 2. Esquema simple que representa la estructura de capas de nuestras redes neuronales.

En la bibliografía proporcionada para este trabajo se nos explicaba una forma específica de entrenamiento de la red para mejorar el modelo, llamada “Fine Tuning” [5]. Hicimos varios intentos a la hora de aplicar este método de entrenamiento, ya que supusimos que las clasificaciones del conjunto de validación aumentarían significativamente en porcentaje de aciertos. Sin embargo, no fuimos capaces de adaptar los procedimientos que se indicaban en esta guía para poder entrenar las redes de manera adecuada. Decidimos, tras múltiples intentos, buscar el mejor modelo sin ser entrenado por “Fine Tuning”, a pesar de que no alcanzaría un porcentaje de acierto tan alto como el que se podría obtener.

IV. Resultados

Para realizar el análisis de los resultados, hemos realizado un total de 5 modelos, donde cada uno muestra una serie de resultados distintos.

Los modelos se construyeron mientras intentábamos alcanzar el mejor de nuestros modelos; lo que hicimos fue construir una serie de capas de ayudarían a mejorar la precisión del algoritmo, y usando los distintos modelos justificamos nuestras decisiones. En esencia, hay 3 operaciones que van variando entre los modelos: número de capas de convolución, existencia de una capa de dropout y aumento de datos de entrada.

Los 5 modelos son los siguientes:

1. Modelo básico construido a partir de pocas capas que vimos necesarias.
2. Ampliación del modelo básico que incluye un número de capas adecuado para la convolución y aumento de datos.
3. Ampliación del modelo básico que incluye una capa de dropout y aumento de datos.
4. Ampliación del modelo básico que incluye capa de dropout y un número adecuado de capas de convolución.
5. Modelo final, que incluye las operaciones que hemos visto óptimas para que el entrenamiento de nuestro sistema sea adecuado.

Para los 5 modelos, hemos realizado un entrenamiento de 100 épocas, y luego hemos imprimido un par de gráficas que representen la evolución de la red neuronal.

Nuestra hipótesis es que necesitamos optimizar la capa de dropout, el número capas de convoluciones y aumentar el número de datos de nuestra red para que el rendimiento de nuestra capa sea máximo.

Uno a uno, iremos mostrando los resultados de los modelos enumerados anteriormente en orden.

Primero, mostraremos y analizaremos los resultados obtenidos de entrenar la red neuronal sin optimizar ninguno de los parámetros mencionados.

Modelo básico

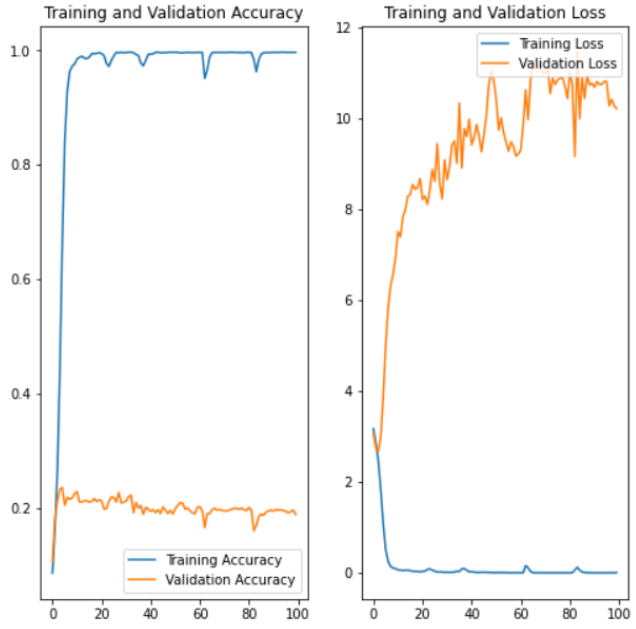

Fig. 3. Gráfica de la precisión de la red para los conjuntos de validación y entrenamiento y gráfica de la pérdida en ambos conjuntos para el primer modelo.

Como observamos en la figura 3, el modelo inicial de nuestra red se ajusta bien al conjunto de entrenamiento, y muy rápidamente, pero no se ajusta bien al conjunto de validación.

Esto quiere decir que existe sobreajuste en nuestro modelo. Para intentar solventar esta incidencia, decidimos aplicar una capa de dropout, que reduce el número de datos a tener en cuenta del conjunto de entrenamiento de forma arbitraria en cada época, lo que obliga a nuestra red a adaptarse a distintos conjuntos de entrenamiento. Además, decidimos aumentar el número de convoluciones para reducir detalles y que nuestra red trabaje con menos parámetros, ya que una cantidad excesiva de parámetros es otro causante típico de sobreajuste. Por último, aumentar el número de datos aplicando rotaciones sobre las imágenes del conjunto de entrenamiento permitirá analizar distintas imágenes parecidas, que también ayuda a reducir el sobreajuste.

Desde aquí, tenemos que ir añadiendo distintas funcionalidades para mejorar la red. Los experimentos que observaremos son acerca de los modelos mencionados antes. Lo que pretendemos explicar mostrando los 3 próximos modelos, es que todas las operaciones incluyen una mejora, pero por separado no llegan a mejorar lo suficiente la red como hace la combinación de los 3.

Ahora mostraremos el segundo modelo, donde hemos optimizado la convolución y hemos aumentado el número de datos, pero aún no hemos aplicado dropout.

Modelo con convolución optimizada y aumento de datos

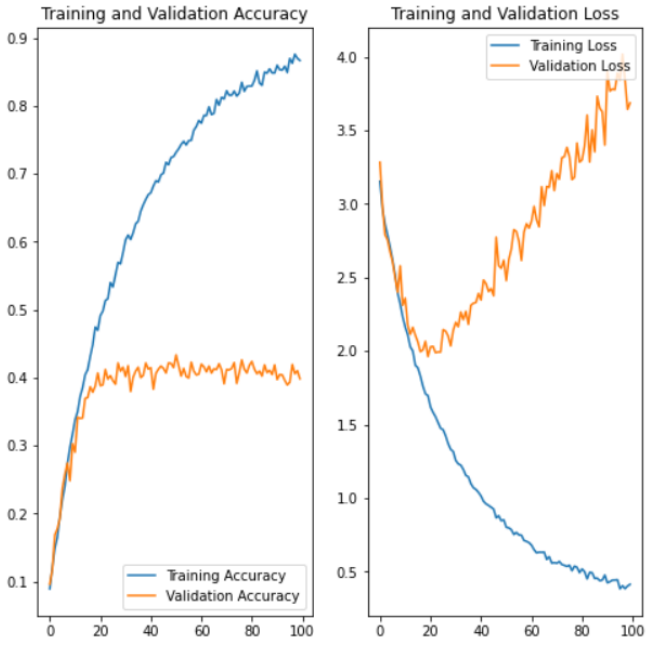

Fig. 4. Gráfica de la precisión de la red para los conjuntos de validación y entrenamiento y gráfica de la pérdida en ambos conjuntos para el segundo modelo.

La gráfica de la figura 4, al contrario que la previa, aumenta su rendimiento para el conjunto de entrenamiento más lentamente, pero su rendimiento para el conjunto de validación

es el doble de efectivo. Esto demuestra que, aplicar una capa de aumento de datos y/o aumentar el número de capas de convolución ayuda a combatir el sobreajuste que teníamos en el primer modelo.

Observemos cómo funciona si, en lugar de aumentar las convoluciones, añadimos una capa de dropout.

Modelo con capa de dropout y aumento de datos

Fig. 5. Gráfica de la precisión de la red para los conjuntos de validación y entrenamiento y gráfica de la pérdida en ambos conjuntos para el tercer modelo.

Podemos observar en la figura 5 que entrena mucho peor para el conjunto de entrenamiento (no llega a 60% de probabilidad de acierto cuando los otros dos modelos superaban el 90%). Esto es esperado, ya que le estamos retirando elementos de dicho conjunto en cada época, por lo que se adapta a casos más genéricos, y menos específicos, por eso ha entrenado mejor para el conjunto de validación que el modelo original que mostramos (se acerca más al 40% que el modelo original).

De aquí deducimos que, en efecto, el sobreajuste se reduce al aplicar dropout, pero se tarda más en entrenar el modelo.

Ahora mostraremos el resultado que obtenemos al aplicarle dropout y aumento de capas de convolución, pero no aumento de datos.

Modelo con capa de dropout y con covolución optimizada

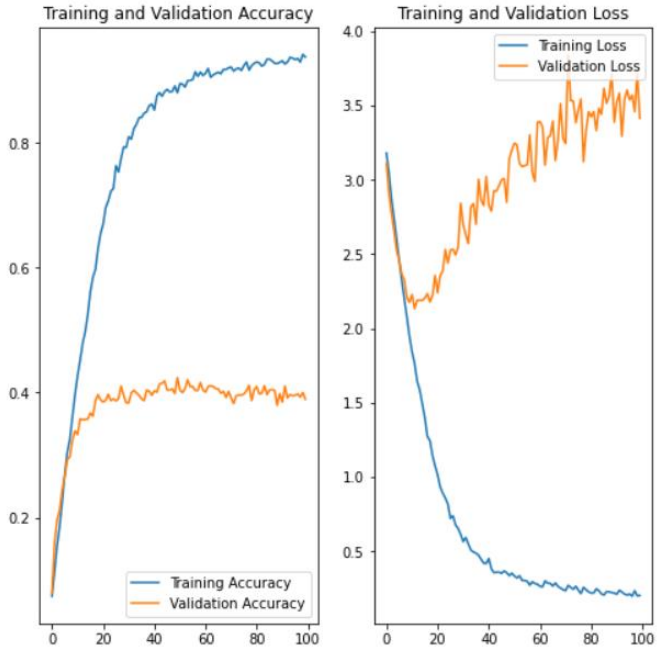

Fig. 6. Gráfica de la precisión de la red para los conjuntos de validación y entrenamiento y gráfica de la pérdida en ambos conjuntos para el cuarto modelo.

Los resultados del modelo de la figura 5 son muy parecidos a los de la figura 6, a pesar de tener operaciones distintas.

Dicho esto, podemos llegar a una conclusión similar a la que llegamos en el segundo modelo, pero esta vez referido a la combinación de la capa de dropout y el aumento de convoluciones.

Bien, ahora que hemos visto que las 3 operaciones ayudan a reducir el sobreajuste, probamos a aplicar las 3 operaciones de forma simultánea y ver cómo mejora la red.

Modelo final con los 3 tipos de capas

Fig. 7. Gráfica de la precisión de la red para los conjuntos de validación y entrenamiento y gráfica de la pérdida en ambos conjuntos para el último modelo.

Como podemos observar en la figura 7, muy similar a la figura 4, la velocidad a la que se adapta al conjunto de entrenamiento es menor, pero es el modelo que mejor se adapta al conjunto de validación. De hecho, la gráfica está aún en aumento, si aumentásemos el número de épocas, la clasificación de ambos conjuntos seguiría mejorando un poco más.

V. Conclusiones

Elaboramos una red neuronal que tiene aproximadamente un 50% de probabilidad de clasificar bien un edificio por su estilo arquitectónico. Para optimizar la red, fuimos empleando técnicas para reducir el sobreajuste (dropout, aumento de datos y disminución de detalles vía convoluciones).

Hemos observado y demostrado la eficacia de las técnicas que hemos empleado para reducir el sobreajuste, mejorando mucho la capacidad de predicción de nuestra red, comparando los resultados obtenidos al emplear o no estas técnicas.

Para poder mejorar esta red, deberíamos implementar “Fine Tuning”, que mejoraría la tasa de aciertos sobre el conjunto de validación.

Referencias

[1] Herramientas de preprocesamiento de imágenes de keras https://keras.io/api/preprocessing/image/

[2] Tutorial de clasificación de imágenes de tensorflow https://www.tensorflow.org/tutorials/images/classification#compile_the_model

[3] Tutorial de clasificación de imágenes de keras https://keras.io/examples/vision/image_classification_from_scratch/

[4] Página de información sobre técnicas para reducir sobreajuste https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/

[5] Página relacionada con información de “Fine Tuning” https://torres.ai/data-augmentation-y-transfer-learning-en-keras-tensorflow/#Fine_Tuning

[6] Página con explicaicón de la convolución para el procesamiento de imágenes https://www.monografias.com/trabajos108/filtros-y-convoluciones/filtros-y-convoluciones.shtml

Bibliografía adicional

Herramienta de guardado de modelos entrenados por tensorflow https://www.tensorflow.org/guide/keras/save_and_serialize?hl=es-419

Página de la asignatura Inteligencia Artificial https://www.cs.us.es/cursos/iais