Facultad de Estudios Estadísticos
Universidad Complutense de Madrid

ntic master
revolucionamos la comunicación

Blockchain & BigData Canino

ANEXO V: Modelo de red neuronal convolucional para la identificación de razas de perro.

Autores:

Cristina Rodríguez Chamorro
Daniel Lanzas Pellico
Helena García Fernández
José Bennani Pareja
Juan José Lucas de la Fuente
Unai Ares Icaran

Tutor:
Sergio Torres Palomino

Documentación del TFM
Máster Blockchain y Big Data. *Curso 2019-2020*

ÍNDICE

Contenido

Introducción

MOTIVACIÓN

Actualmente, conocemos el gran impacto y desarrollo que está teniendo el concepto del Deep Learning en el campo de la inteligencia artificial. Ya que, el claro avance de la tecnología está permitiendo que las empresas, industrias, e incluso universidades, despierten un notable interés por la obtención de nuevas técnicas que puedan aplicarse en su beneficio, para la generación de nuevas aplicaciones. (ARTEAGA, 2015). El objetivo del estudio del aprendizaje automático se basa en simular la inteligencia humana de forma artificial. De manera que, el Deep learning ha revolucionado el estado del arte, haciendo posible realizar tareas como el reconocimiento de voz, visión artificial etc. (Cárdenas, 2018).

Por todo ello, este trabajo se va a centrar en la aplicación de las técnicas predominantes del Deep Learning, en las que se hace uso de las redes neuronales convolucionales, para la clasificación de razas de perro. Ya que, se ha creído conveniente aprovechar esta herramienta tan interesante, para añadir funcionalidad a la parte Web del presente proyecto, y, de esta manera, hacer posible que cualquier usuario pueda reconocer la raza de un perro.

OBJETIVOS PRINCIPALES

- ### Objetivo general

Generar un modelo de Red Neuronal Convolucional para la clasificación de razas de perro.

- ### Objetivos específicos

  - Obtención de un dataset con más de 20000 imágenes de perros, y 120 razas distintas.
  - Obtención del conjunto de datos de prueba y entrenamiento.
  - Búsqueda de los mejores modelos preentrenados, y entrenamiento con distintos modelos.
  - Comparación de los modelos entrenados y obtención del mejor modelo.
  - Realización de las predicciones con los datos de prueba, y análisis de los resultados.
  - Generar un script que permita la obtención de la predicción de la imagen que el usuario introduzca en la página web.

ESTRUCTURA DEL DOCUMENTO Y TECNOLOGÍAS UTILIZADAS

A continuación, se presenta la estructura de esta parte del proyecto, con el objetivo de comprender los pasos que se han seguido para implementar el modelo en cuestión.

• Estructura del documento

- **Introducción.** El capítulo 1 comprende la introducción y planteamiento al desarrollo del algoritmo para la clasificación de imágenes.
- **Conceptos básicos sobre redes neuronales convolucionales.** El capítulo 2 tiene como objetivo presentar los conceptos básicos de las redes neuronales convolucionales, para comprender su estructura y organización. Ya que, son las redes con las que se va a trabajar para conseguir los objetivos propuestos en el capítulo 1.
- **Análisis y diseño del modelo.** El capítulo 3 tiene como objetivo presentar todas las fases de análisis y diseño que se ha llevado a cabo para la implementación del algoritmo en cuestión.
- **Predicciones del conjunto de prueba.** El capítulo 4, tiene como objetivo mostrar el resultado de las predicciones realizadas sobre el conjunto de prueba.
- **Conclusiones y líneas futuras.** Este capítulo tiene como objetivo evaluar el cumplimiento de los objetivos iniciales, y la presentación de las líneas futuras que podrían complementar el trabajo realizado.

• Tecnologías utilizadas

- **Anaconda Navigator.** Es una GUI de escritorio que viene con Anaconda Individual Edition. Facilita el lanzamiento de aplicaciones y la administración de paquetes y entornos sin usar comandos de línea de comandos. Se ha utilizado para el uso de Python en el desarrollo del notebook para la implementación del modelo de red.

- **PuTTy.** Es un cliente SSH, Telnet, rlogin, y TCP raw con licencia libre. Se ha utilizado para la conexión con el servidor de Amazon Linux.
- **MobaXTerm.** Es una herramienta todo en uno que busca llevar a los usuarios más profesionales de Windows determinadas funciones de Linux como el uso de los comandos más habituales para controlar el sistema operativo desde el teclado. De manera que, se ha utilizado para las conexiones con el servidor de Amazon y para la cargar de datos. Ya que nos ofrece una consola mucho más visual.

- **Servidor Deep Learning Amazon Linux.** Se ha utilizado para el entrenamiento de los modelos, ya que, debido a la gran cantidad de datos utilizada, ha resultado imposible trabajar en local.

Conceptos básicos sobre redes neuronales convolucionales (CNNs)

La red neuronal convolucional es un tipo de red multicapa que presenta una alternancia entre las capas de convolución y de agrupación, finalizando con una red de neuronas artificiales convencional. (García, 2019).

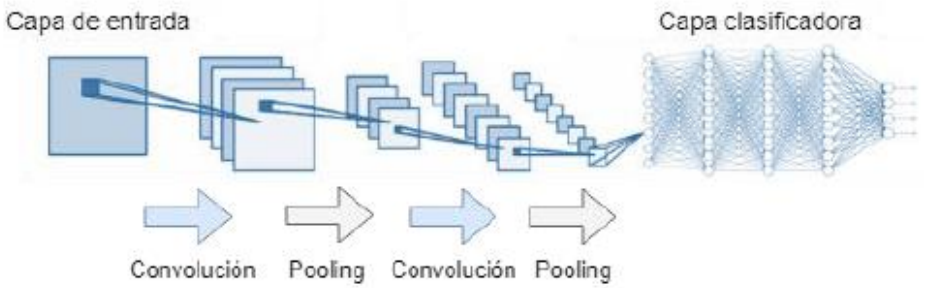


Figura 1. Arquitectura de una red CNN (García, 2019)

Capa de entrada: Capa destinada al procesamiento de los píxeles de la imagen introducida. Es decir, está formada por tantos píxeles como tenga la imagen. Es importante comentar que, aunque la imagen sea una matriz de píxeles, y el valor de los píxeles vaya de 0 a 255, para una red neuronal se normaliza de 0 a 1.

Capas ocultas: Capas destinadas a la obtención de las características más importantes de las imágenes. De manera que, las primeras capas son capaces de detectar líneas, curvas, y se van especializando hasta llegar a las capas más profundas. Dentro de estas capas, encontramos: (García, 2019).

Capas de convolución: Estas capas son las encargadas de aplicar los filtros necesarios para conseguir las características principales o los patrones más característicos de las imágenes recibidas. Por tanto, tras el preprocesamiento de la imagen, comienza el proceso distintivo de las CNN. De manera que, tienen lugar las llamadas convoluciones, en las que se toman grupos de píxeles de la imagen de entrada para realizar el producto escalar contra una pequeña matriz denominada kernel. Es decir, el kernel será el encargado de ir recorriendo todas las neuronas de entrada, dando como resultado una nueva capa de neuronas ocultas. (Pacheco, 2017). Por tanto, a medida que se va desplazando el kernel, se va obteniendo una nueva imagen filtrada por el kernel. De manera que, las imágenes nuevas dibujan las características de la imagen original. A continuación, se muestra una convolución con un cierto Kernel, para la comprensión de lo anteriormente comentado.

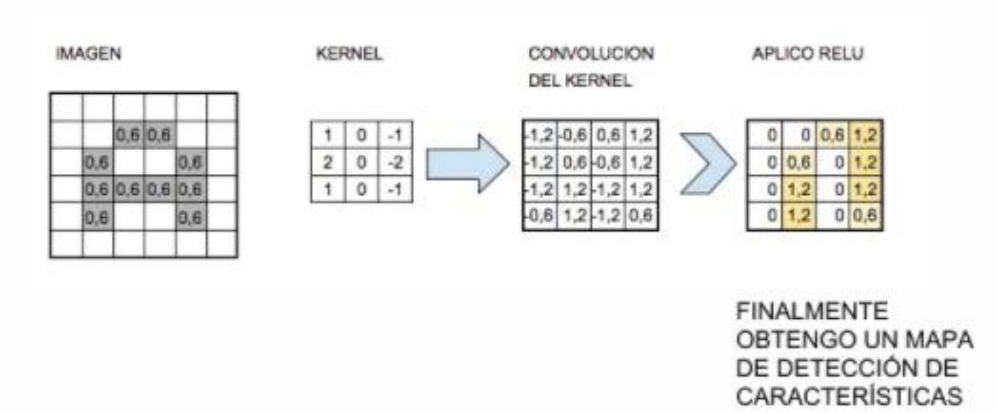


Figura 2. Ejemplo de una convolución con un kernel determinado. Nota: la función de activación consiste en f(x)=max(0,x). (Bagnato, 2018)

Capas de agrupación o pooling: Realizan la reducción del tamaño de la imagen, para poder trabajar con un menor número de píxeles y mejorar el rendimiento y la calidad del algoritmo. (Bagnato, 2018).

Posteriormente a la convolución, viene el paso del pooling o subsampling. Es decir, después de la convolución se produce una reducción de la cantidad de neuronas antes de realizar una nueva convolución. Esto es porque el número de neuronas después de la primera convolución genera una capa oculta del número de neuronas iniciales por el número de mapas que se incluyan en la convolución. Es decir, si tenemos una foto en blanco y negro 28x28px y 32 mapas de características, tendremos inicialmente 784 neuronas, y después de la convolución, 25088 neuronas. Lo cual quiere decir que, si se continúan realizando convoluciones, el número de neuronas resultantes en la segunda convolución, será demasiado grande, por lo que el procesamiento necesario para esta red sería muy elevado. (Bagnato, 2018). Por esta razón aparece el concepto de pooling o subsampling. Ya que, gracias a este procedimiento se va a reducir considerablemente el tamaño de las imágenes filtradas. De manera que, se obtendrán las características más importantes de cada una de ellas, para reducir el tamaño de las neuronas antes de una nueva convolución. Es importante comentar que, encontramos 2 tipos de pooling:

- Max pooling: Procedimiento a partir del cual se selecciona el mayor valor de los píxeles que intervienen. A continuación, se puede observar un ejemplo de max pooling:

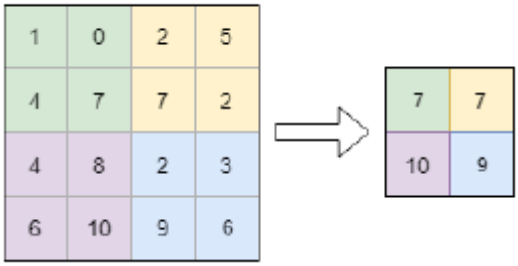


Figura 3. Ejemplo max pooling (García, 2019)

- Average pooling: Este procedimiento consiste en calcular la media de los píxeles de la matriz filtrada. A continuación, se puede observar en la figura 6 un ejemplo de average pooling.

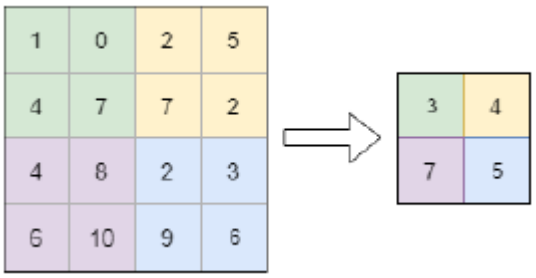


Figura 4. Ejemplo de Average Pooling. (García, 2019)

Red de neuronas artificial convencional: Red de neuronas que conectará con la última capa de subsampling y finalizará con la cantidad de neuronas que queremos clasificar. Utilizará backpropagation para ajustar los pesos de todas las interconexiones de las capas. Sin embargo, en la red CNN también se utilizará este procedimiento para ajustar los pesos de los kernels.

Análisis y diseño del modelo

ANÁLISIS Y DISEÑO DEL MODELO

Para la obtención de un mejor modelo de red neuronal convolucional que sea capaz de identificar razas de perros, ha sido necesario hacer uso de las 120 razas de perro utilizadas en el dataset Standford Dog DataSet. Este conjunto de datos se ha creado utilizando imágenes y anotaciones de ImageNet para la tarea de categorización detallada de imágenes. (Li, s.f.) . Dicho Dataset es accesible desde el siguiente sitio web: https://www.kaggle.com/jessicali9530/stanford-dogs-dataset.

De manera que, se han utilizado 120 categorías de perro y 22.580 imágenes, distribuidas en 9960 imágenes para el conjunto de prueba y 10620 para el conjunto de entrenamiento. Es importante comentar que, para la obtención del conjunto de entrenamiento y del conjunto de prueba, se ha llevado a cabo la realización de un script que añade 83 imágenes de cada categoría a la carpeta Images_test. El path de dicha carpeta se utilizará posteriormente para realizar las predicciones sobre el conjunto de prueba, es decir, sobre imágenes de distintas razas de perro que no se van a utilizar para entrenar los modelos.

Además de todo ello, para lograr un óptimo análisis exploratorio de los datos, se ha desarrollado en el notebook ModeloBigData-TFM.ipynb realizado, un código que te permite conocer el nombre de las razas de perro utilizadas, y el número de imágenes correspondientes a cada una de ellas, para el conjunto de entrenamiento. El resultado referente a la implementación de dicho código se puede apreciar a continuación:

INFO:root: Directorio del dataset de imágenes: /home/ec2-user/images/Images
INFO:root: Buscando la 7ctivación7 del dataset
INFO:root: Numero de clases: 120 con nombres: [‘n02085620-Chihuahua’, ‘n02085782-Japanese_spaniel’, ‘n02085936-Maltese_dog’, ‘n02086079-Pekinese’, ‘n02086240-Shih-Tzu’, ‘n02086646-Blenheim_spaniel’, ‘n02086910-papillon’, ‘n02087046-toy_terrier’, ‘n02087394-Rhodesian_ridgeback’, ‘n02088094-Afghan_hound’, ‘n02088238-basset’, ‘n02088364-beagle’, ‘n02088466-bloodhound’, ‘n02088632-bluetick’, ‘n02089078-black-and-tan_coonhound’, ‘n02089867-Walker_hound’, ‘n02089973-English_foxhound’, ‘n02090379-redbone’, ‘n02090622-borzoi’, ‘n02090721-Irish_wolfhound’, ‘n02091032-Italian_greyhound’, ‘n02091134-whippet’, ‘n02091244-Ibizan_hound’, ‘n02091467-Norwegian_elkhound’, ‘n02091635-otterhound’, ‘n02091831-Saluki’, ‘n02092002-Scottish_deerhound’, ‘n02092339-Weimaraner’, ‘n02093256-Staffordshire_bullterrier’, ‘n02093428-American_Staffordshire_terrier’, ‘n02093647-Bedlington_terrier’, ‘n02093754-Border_terrier’, ‘n02093859-Kerry_blue_terrier’, ‘n02093991-Irish_terrier’, ‘n02094114-Norfolk_terrier’, ‘n02094258-Norwich_terrier’, ‘n02094433-Yorkshire_terrier’, ‘n02095314-wire-haired_fox_terrier’, ‘n02095570-Lakeland_terrier’, ‘n02095889-Sealyham_terrier’, ‘n02096051-Airedale’, ‘n02096177-cairn’, ‘n02096294-Australian_terrier’, ‘n02096437-Dandie_Dinmont’, ‘n02096585-Boston_bull’, ‘n02097047-miniature_schnauzer’, ‘n02097130-giant_schnauzer’, ‘n02097209-standard_schnauzer’, ‘n02097298-Scotch_terrier’, ‘n02097474-Tibetan_terrier’, ‘n02097658-silky_terrier’, ‘n02098105-soft-coated_wheaten_terrier’, ‘n02098286-West_Highland_white_terrier’, ‘n02098413-Lhasa’, ‘n02099267-flat-coated_retriever’, ‘n02099429-curly-coated_retriever’, ‘n02099601-golden_retriever’, ‘n02099712-Labrador_retriever’, ‘n02099849-Chesapeake_Bay_retriever’, ‘n02100236-German_short-haired_pointer’, ‘n02100583-vizsla’, ‘n02100735-English_setter’, ‘n02100877-Irish_setter’, ‘n02101006-Gordon_setter’, ‘n02101388-Brittany_spaniel’, ‘n02101556-clumber’, ‘

n02102040-English_springer', 'n02102177-Welsh_springer_spaniel', 'n02102318-cocker_spaniel', 'n0 2102480-Sussex_spaniel', 'n02102973-Irish_water_spaniel', 'n02104029-kuvasz', 'n02104365-schipp erke', 'n02105056-groenendael', 'n02105162-malinois', 'n02105251-briard', 'n02105412-kelpie', 'n02 105505-komondor', 'n02105641-Old_English_sheepdog', 'n02105855-Shetland_sheepdog', 'n021060 30-collie', 'n02106166-Border_collie', 'n02106382-Bouvier_des_Flandres', 'n02106550-Rottweiler', 'n 02106662-German_shepherd', 'n02107142-Doberman', 'n02107312-miniature_pinscher', 'n0210757 4-Greater_Swiss_Mountain_dog', 'n02107683-Bernese_mountain_dog', 'n02107908-Appenzeller', 'n 02108000-EntleBucher', 'n02108089-boxer', 'n02108422-bull_mastiff', 'n02108551-Tibetan_mastiff', 'n02108915-French_bulldog', 'n02109047-Great_Dane', 'n02109525-Saint_Bernard', 'n02109961-Esk imo_dog', 'n02110063-malamute', 'n02110185-Siberian_husky', 'n02110627-affenpinscher', 'n02110 806-basenji', 'n02110958-pug', 'n02111129-Leonberg', 'n02111277-Newfoundland', 'n02111500-Gre at_Pyrenees', 'n02111889-Samoyed', 'n02112018-Pomeranian', 'n02112137-chow', 'n02112350-kees hond', 'n02112706-Brabancon_griffon', 'n02113023-Pembroke', 'n02113186-Cardigan', 'n02113624- toy_poodle', 'n02113712-miniature_poodle', 'n02113799-standard_poodle', 'n02113978-Mexican_h airless', 'n02115641-dingo', 'n02115913-dhole', 'n02116738-African_hunting_dog']
INFO:root: Numero de imágenes de la clase n02085620-Chihuahua: 69
INFO:root: Numero de imágenes de la clase n02085782-Japanese_spaniel: 102
INFO:root: Numero de imágenes de la clase n02085936-Maltese_dog: 169
INFO:root: Numero de imágenes de la clase n02086079-Pekinese: 66
INFO:root: Numero de imágenes de la clase n02086240-Shih-Tzu: 131
INFO:root: Numero de imágenes de la clase n02086646-Blenheim_spaniel: 105
INFO:root: Numero de imágenes de la clase n02086910-papillon: 113
INFO:root: Numero de imágenes de la clase n02087046-toy_terrier: 89
INFO:root: Numero de imágenes de la clase n02087394-Rhodesian_ridgeback: 89
INFO:root: Numero de imágenes de la clase n02088094-Afghan_hound: 156
INFO:root: Numero de imágenes de la clase n02088238-basset: 92
INFO:root: Numero de imágenes de la clase n02088364-beagle: 112
INFO:root: Numero de imágenes de la clase n02088466-bloodhound: 104
INFO:root: Numero de imágenes de la clase n02088632-bluetick: 88
INFO:root: Numero de imágenes de la clase n02089078-black-and-tan_coonhound: 76
INFO:root: Numero de imágenes de la clase n02089867-Walker_hound: 70
INFO:root: Numero de imágenes de la clase n02089973-English_foxhound: 74
INFO:root: Numero de imágenes de la clase n02090379-redbone: 65
INFO:root: Numero de imágenes de la clase n02090622-borzoi: 68
INFO:root: Numero de imágenes de la clase n02090721-Irish_wolfhound: 135
INFO:root: Numero de imágenes de la clase n02091032-Italian_greyhound: 99
INFO:root: Numero de imágenes de la clase n02091134-whippet: 104
INFO:root: Numero de imágenes de la clase n02091244-Ibizan_hound: 105
INFO:root: Numero de imágenes de la clase n02091467-Norwegian_elkhound: 113
INFO:root: Numero de imágenes de la clase n02091635-otterhound: 68
INFO:root: Numero de imágenes de la clase n02091831-Saluki: 117
INFO:root: Numero de imágenes de la clase n02092002-Scottish_deerhound: 149
INFO:root: Numero de imágenes de la clase n02092339-Weimaraner: 77
INFO:root: Numero de imágenes de la clase n02093256-Staffordshire_bullterrier: 72
INFO:root: Numero de imágenes de la clase n02093428-American_Staffordshire_terrier: 81
INFO:root: Numero de imágenes de la clase n02093647-Bedlington_terrier: 99
INFO:root: Numero de imágenes de la clase n02093754-Border_terrier: 89

INFO:root: Numero de imágenes de la clase n02093859-Kerry_blue_terrier: 96
INFO:root: Numero de imágenes de la clase n02093991-Irish_terrier: 86
INFO:root: Numero de imágenes de la clase n02094114-Norfolk_terrier: 89
INFO:root: Numero de imágenes de la clase n02094258-Norwich_terrier: 102
INFO:root: Numero de imágenes de la clase n02094433-Yorkshire_terrier: 81
INFO:root: Numero de imágenes de la clase n02095314-wire-haired_fox_terrier: 74
INFO:root: Numero de imágenes de la clase n02095570-Lakeland_terrier: 114
INFO:root: Numero de imágenes de la clase n02095889-Sealyham_terrier: 119
INFO:root: Numero de imágenes de la clase n02096051-Airedale: 119
INFO:root: Numero de imágenes de la clase n02096177-cairn: 114
INFO:root: Numero de imágenes de la clase n02096294-Australian_terrier: 113
INFO:root: Numero de imágenes de la clase n02096437-Dandie_Dinmont: 97
INFO:root: Numero de imágenes de la clase n02096585-Boston_bull: 99
INFO:root: Numero de imágenes de la clase n02097047-miniature_schnauzer: 71
INFO:root: Numero de imágenes de la clase n02097130-giant_schnauzer: 74
INFO:root: Numero de imágenes de la clase n02097209-standard_schnauzer: 72
INFO:root: Numero de imágenes de la clase n02097298-Scotch_terrier: 75
INFO:root: Numero de imágenes de la clase n02097474-Tibetan_terrier: 123
INFO:root: Numero de imágenes de la clase n02097658-silky_terrier: 100
INFO:root: Numero de imágenes de la clase n02098105-soft-coated_wheaten_terrier: 73
INFO:root: Numero de imágenes de la clase n02098286-West_Highland_white_terrier: 86
INFO:root: Numero de imágenes de la clase n02098413-Lhasa: 103
INFO:root: Numero de imágenes de la clase n02099267-flat-coated_retriever: 69
INFO:root: Numero de imágenes de la clase n02099429-curly-coated_retriever: 68
INFO:root: Numero de imágenes de la clase n02099601-golden_retriever: 67
INFO:root: Numero de imágenes de la clase n02099712-Labrador_retriever: 88
INFO:root: Numero de imágenes de la clase n02099849-Chesapeake_Bay_retriever: 84
INFO:root: Numero de imágenes de la clase n02100236-German_short-haired_pointer: 69
INFO:root: Numero de imágenes de la clase n02100583-vizsla: 71
INFO:root: Numero de imágenes de la clase n02100735-English_setter: 78
INFO:root: Numero de imágenes de la clase n02100877-Irish_setter: 72
INFO:root: Numero de imágenes de la clase n02101006-Gordon_setter: 70
INFO:root: Numero de imágenes de la clase n02101388-Brittany_spaniel: 69
INFO:root: Numero de imágenes de la clase n02101556-clumber: 67
INFO:root: Numero de imágenes de la clase n02102040-English_springer: 76
INFO:root: Numero de imágenes de la clase n02102177-Welsh_springer_spaniel: 67
INFO:root: Numero de imágenes de la clase n02102318-cocker_spaniel: 76
INFO:root: Numero de imágenes de la clase n02102480-Sussex_spaniel: 68
INFO:root: Numero de imágenes de la clase n02102973-Irish_water_spaniel: 67
INFO:root: Numero de imágenes de la clase n02104029-kuvasz: 67
INFO:root: Numero de imágenes de la clase n02104365-schipperke: 71
INFO:root: Numero de imágenes de la clase n02105056-groenendael: 67
INFO:root: Numero de imágenes de la clase n02105162-malinois: 67
INFO:root: Numero de imágenes de la clase n02105251-briard: 69
INFO:root: Numero de imágenes de la clase n02105412-kelpie: 70
INFO:root: Numero de imágenes de la clase n02105505-komondor: 71
INFO:root: Numero de imágenes de la clase n02105641-Old_English_sheepdog: 86

INFO:root: Numero de imágenes de la clase n02105855-Shetland_sheepdog: 74
INFO:root: Numero de imágenes de la clase n02106030-collie: 70
INFO:root: Numero de imágenes de la clase n02106166-Border_collie: 67
INFO:root: Numero de imágenes de la clase n02106382-Bouvier_des_Flandres: 67
INFO:root: Numero de imágenes de la clase n02106550-Rottweiler: 69
INFO:root: Numero de imágenes de la clase n02106662-German_shepherd: 69
INFO:root: Numero de imágenes de la clase n02107142-Doberman: 67
INFO:root: Numero de imágenes de la clase n02107312-miniature_pinscher: 101
INFO:root: Numero de imágenes de la clase n02107574-Greater_Swiss_Mountain_dog: 85
INFO:root: Numero de imágenes de la clase n02107683-Bernese_mountain_dog: 135
INFO:root: Numero de imágenes de la clase n02107908-Appenzeller: 68
INFO:root: Numero de imágenes de la clase n02108000-EntleBucher: 119
INFO:root: Numero de imágenes de la clase n02108089-boxer: 68
INFO:root: Numero de imágenes de la clase n02108422-bull_mastiff: 73
INFO:root: Numero de imágenes de la clase n02108551-Tibetan_mastiff: 69
INFO:root: Numero de imágenes de la clase n02108915-French_bulldog: 76
INFO:root: Numero de imágenes de la clase n02109047-Great_Dane: 73
INFO:root: Numero de imágenes de la clase n02109525-Saint_Bernard: 87
INFO:root: Numero de imágenes de la clase n02109961-Eskimo_dog: 67
INFO:root: Numero de imágenes de la clase n02110063-malamute: 95
INFO:root: Numero de imágenes de la clase n02110185-Siberian_husky: 109
INFO:root: Numero de imágenes de la clase n02110627-affenpinscher: 67
INFO:root: Numero de imágenes de la clase n02110806-basenji: 126
INFO:root: Numero de imágenes de la clase n02110958-pug: 117
INFO:root: Numero de imágenes de la clase n02111129-Leonberg: 127
INFO:root: Numero de imágenes de la clase n02111277-Newfoundland: 112
INFO:root: Numero de imágenes de la clase n02111500-Great_Pyrenees: 130
INFO:root: Numero de imágenes de la clase n02111889-Samoyed: 135
INFO:root: Numero de imágenes de la clase n02112018-Pomeranian: 136
INFO:root: Numero de imágenes de la clase n02112137-chow: 113
INFO:root: Numero de imágenes de la clase n02112350-keeshond: 75
INFO:root: Numero de imágenes de la clase n02112706-Brabancon_griffon: 70
INFO:root: Numero de imágenes de la clase n02113023-Pembroke: 98
INFO:root: Numero de imágenes de la clase n02113186-Cardigan: 72
INFO:root: Numero de imágenes de la clase n02113624-toy_poodle: 68
INFO:root: Numero de imágenes de la clase n02113712-miniature_poodle: 72
INFO:root: Numero de imágenes de la clase n02113799-standard_poodle: 76
INFO:root: Numero de imágenes de la clase n02113978-Mexican_hairless: 72
INFO:root: Numero de imágenes de la clase n02115641-dingo: 73
INFO:root: Numero de imágenes de la clase n02115913-dhole: 67
INFO:root: Numero de imágenes de la clase n02116738-African_hunting_dog: 86
INFO:root: Numero de imágenes de todas las clases en total: 10620

CONFIGURACIÓN DEL SERVIDOR AMAZON LINUX (DEEP LEARNING | VERSIÓN 33.0)

Tras conocer en detalle la cantidad de datos que se iban a utilizar, y tras realizar varias pruebas en local, fue realmente necesario el uso de un servidor de Amazon Web Service (AWS), de pago, para el entrenamiento del modelo. Ya que, es importante tener en cuenta que, por la gran cantidad de datos seleccionados, no era viable entrenar los modelos en la máquina local. Por esta razón, se ha seleccionado una máquina de 8 cores, 32 GB de RAM y 100 GB de memoria, correspondiente con el servidor de Amazon Linux Versión 33.0.

De manera que, una vez levantado el servidor, se ha procedido a su configuración para la elaboración de los modelos pertinentes. Para ello, se ha hecho uso del cliente SSH PuTTY para generar la key *.ppk* que nos servirá junto a la *ip* para conectarnos con el terminal de MobaXterm, a la máquina en cuestión. Además de todo ello, es necesaria la descarga e instalación de Anaconda, que incluye la aplicación jupyter notebooks que utilizaremos para el diseño de los modelos.

El procedimiento realizado para la descarga y configuración del servidor es la siguiente:

Paso 1: Descargar Anaconda en la instancia EC2 de la máquina.

Wget https://repo.continuum.io/archive/Anaconda3-4.2.0-Linux-x86_64.sh

Paso 2: Instalar Anaconda.

Bash Anaconda3-4.2.0-Linux-x86_64.sh

Paso 3: Creamos una password para el jupyter notebook.

Ipython

from Ipython.lib import passwd

passwd()

Enter password: [Create password and press enter] Verify password: [Press enter]

Paso 4: Creamos config profile

mkdir certs

cd certs

openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem

Paso 5: Configuramos Jupyter

cd ~/.jupyter/

vi jupyter_notebook_config.py

Añadimos lo siguiente en el .py:

```
c = get_config()
# Kernel config
c.IPKernelApp.pylab = 'inline'  # if you want plotting support always in your notebook
# Notebook config
c.NotebookApp.certfile = u'/home/12ctiva/certs/mycert.pem' #location of your certificate file
c.NotebookApp.ip = '0.0.0.0'
c.NotebookApp.open_browser = False  #so that the ipython notebook does not opens up a browser by default
c.NotebookApp.password = u'sha1:98ff0e580111:12798c72623a6eecd54b51c006b1050f0ac1a62d' #the encrypted password we generated above
# Set the port to 8888, the port we set up in the AWS EC2 set-up
c.NotebookApp.port = 8888
```

Posteriormente, pulsamos:

esc
shift-z

Paso 6: Creamos carpetas para los notebooks

cd ~

mkdir Notebooks

cd Notebooks

Paso 7: Creamos un nuevo screen

screen

Paso 8: Lanzamos Jupyter Notebooks

sudo chown $USER:$USER /home/12ctiva/certs/mycert.pem

jupyter notebook

Paso 9: Accedemos a jupyter por internet con el dns de la máquina de Amazon.

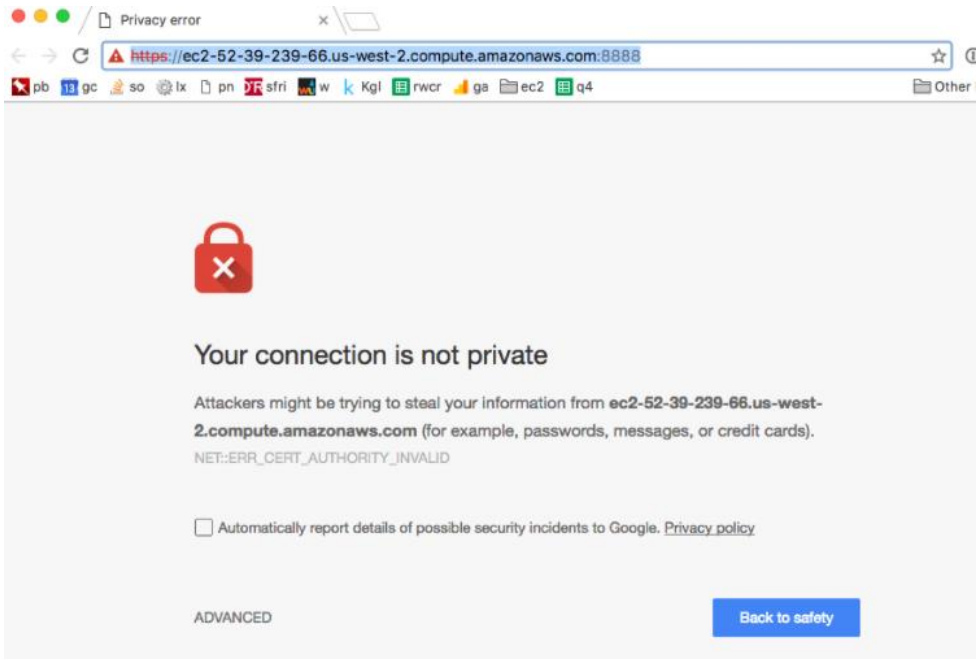


Figura 5. Acceso a jupyter por internet

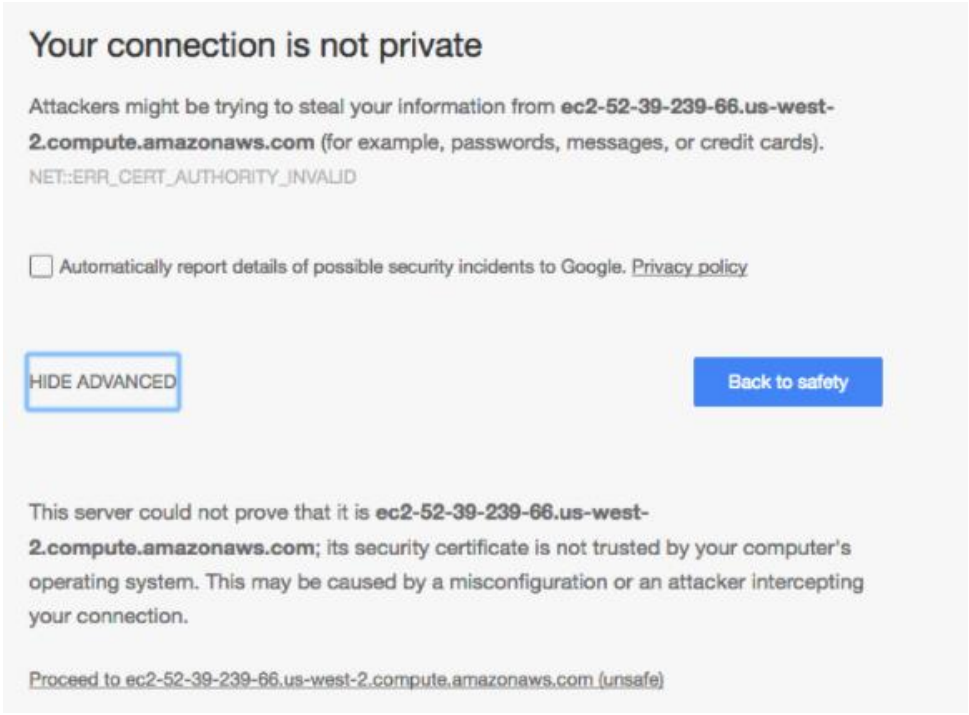
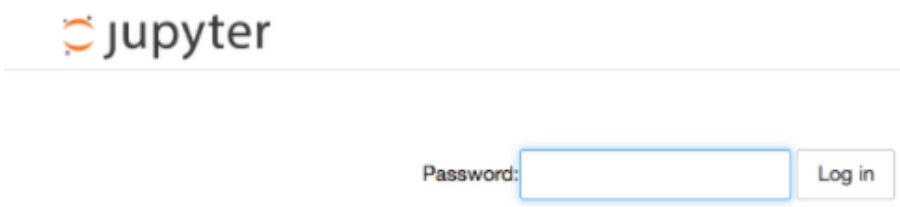


Figura 6. Acceso a jupyter por internet

A continuación, aparece una ventana en la que es necesario introducir la contraseña generada en el paso 3 del presente Anexo.



Paso 10: Cargamos el dataset en la ruta: /home/ec2-user/images.

Durante el proceso de diseño, se han realizado varias pruebas en local, con menos cantidad de datos. De manera que, se ha generado un primer notebook como borrador, en local, que posteriormente se ha cargado en el servidor para realizar las pruebas con la información completa del Dataset utilizado, y, de esta forma, obtener un notebook definitivo.

Paso 11: Configuración e instalación de las librerías necesarias en el kernel del notebook en cuestión.

ESTUDIOS DE LOS MEJORES MODELOS CNNs

Para conseguir el objetivo propuesto, se van a elegir los mejores modelos preentrenados. Todo ello se va a realizar comparando el comportamiento de los modelos que nos ofrece keras. De manera que, va a ser necesario observar *Top-1 Accuracy* y *Top-5 Accuracy*, de los modelos proporcionados en la página oficial de keras (Modelos Keras, s.f.), para la elección de tres de ellos. Dichos modelos, también se pueden observar en la tabla 1, que se observa a continuación:

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22.910.480	126
VGG16	582 MB	0.713	0.901	138.357.544	23
VGG19	549 MB	0.713	0.900	143.667.240	26
ResNet50	98 MB	0.749	0.921	25.636.712	-
ResNet101	171 MB	0.764	0.928	44.707.176	-
ResNet152	232 MB	0.766	0.931	60.419.944	-
ResNet50V2	98 MB	0.760	0.930	25.613.800	-
ResNet101V2	171 MB	0.772	0.938	44.675.560	-
ResNet152V2	232 MB	0.780	0.942	60.380.648	-
InceptionV3	92 MB	0.779	0.937	23.851.784	159
InceptionResNetV2	215 MB	0.803	0.953	55.873.763	572
MobileNet	16 MB	0.704	0.895	4.253.864	88
MobileNetV2	14 MB	0.713	0.901	3.538.984	88
DenseNet121	33 MB	0.750	0.923	8.062.504	121
DenseNet169	57 MB	0.762	0.932	14.307.880	169
DenseNet201	80 MB	0.773	0.936	20.242.984	201
NASNetMobile	23 MB	0.744	0.919	5.326.716	-
NASNetLarge	343 MB	0.825	0.960	88.949.818	-

Tabla 1. Modelos obtenidos de la página oficial de Keras.

Por lo que, teniendo en cuenta la información anterior, se van a usar los siguientes modelos preentrenados: VGG16, InceptionResNetV2, MobileNetV2. De manera que, se entrenarán únicamente las capas nuevas de cada modelo, y, posteriormente, se escogerá el modelo que presente mayor accuracy para realizar las predicciones con el conjunto de prueba.

DEFINICIÓN Y ENTRENAMIENTO DE LOS MODELOS

En el presente apartado se procederá a la definición y entrenamiento de los modelos seleccionados: *VGG16, InceptionResNetV2, y MobileNetV2*.

- Definición de los modelos.

De manera que, es necesario tener en cuenta que, al ser modelos preentrenados, será necesario eliminar la última capa de cada uno de ellos para adaptarla a los requerimientos de nuestro objetivo principal. Además, no será necesario entrenar las capas ya entrenadas, por lo que sólo se entrenaran las capas añadidas. De manera que, por cada modelo, se tienen las siguientes capas añadidas:

- **Capa añadida con el output de la última capa**, para la obtención promedio de las dimensiones espaciales. Para ello, se ha utilizado la función *GlobalAveragePooling2D*.

- **Capa con 512 nodos en la que se utilizará el rectificador lineal *activation relu***, que busca eliminar los valores negativos y dejar los positivos tal y como entran, es la función de activación más usada en deep learning y, especialmente, en los trabajos con imágenes.

- **Última capa densa, con 120 nodos.** Es la capa que identificará el tipo de raza de perro que tenemos en la imagen según los resultados que obtiene de las capas anteriores de la red. Esta capa utilizará la función *softmax*, que nos permite obtener la probabilidad de a qué clase de raza pertenece la imagen a identificar.

Los pasos que se han seguido para implementación de las capas comentadas se definen a continuación:

- **Paso 1:** Eliminamos la última capa del modelo preentrenado estableciendo el atributo *include_top* a False.

- **Paso 2:** Añadimos la primera capa utilizando el output del modelo pre-entrenado junto con la función *GlobalAveragePooling2D* para aplicar la agrupación promedio de las dimensiones espaciales.

- **Paso 3:** Añadimos la segunda capa con 512 nodos en ella y se aplicará la activation *relu* en la capa. Esta función es un rectificador lineal que busca eliminar los valores negativos y dejar los positivos tal y como entran, es la función de activación más usada en el 16cti learning y, especialmente, en los trabajos con imágenes.

- **Paso 4:** Añadimos la última capa densa, utilizando la función Dense, donde se usan los parámetros units y activation. Estos dos parámetros obtendrán el valor *120 y 'softmax'*. Estos valores son debidos al número de clases que tenemos que identificar en nuestro dataset y, por el otro lado, la función *softmax*, nos permite obtener una probabilidad de a qué clase de raza pertenece la imagen a identificar, es decir, es la capa que identificará el tipo de raza de perro que tenemos en la imagen según los resultados que obtiene de las capas anteriores de la red.

Ejemplos de la implementación descrita en los modelos utilizados:

- **Modelo VGG16**

```
#Modelo preentrenado VGG16
logging.info("Obtenemos el modelo base VGG16 sin la ultima capa")
modelVGG16_base=VGG16(weights='imagenet',include_top=False)
modelVGG16_base.trainable=False
x=modelVGG16_base.output
#Primera capa con el output de la ultima capa
logging.info("Primera capa con el output de la ultima capa")
x=GlobalAveragePooling2D()(x)
#dense layer 3 crea una nueva capa oculta con 512 nodos con activacion relu
logging.info("Segunda capa oculta con 512 nodos con activacion relu")
x=Dense(512,activation='relu')(x)
#Creamos la ultima capa con 120 neuronas
#las 120 razas de perros que queremos identificar)
#y aplicamos la activacion softmax
logging.info("Ultima capa")
preds=Dense(120,activation='softmax')(x) #final layer with softmax activation
#Creamos el modelo con las nuevas capas
logging.info("Generamos el modelo con las capas nuevas")
modelVGG16=Model(inputs=modelVGG16_base.input,outputs=preds)
```

Figura 7. Modelo preentrenado VGG16

- **Modelo InceptionResNetV2**

```
#Modelo preentrenado InceptionResNetV2
logging.info("Obtenemos el modelo base InceptionResNetV2 sin la ultima capa")
modelInceptionResNetV2_base = keras.applications.resnet_v2.ResNet50V2(weights='imagenet', include_top=False)
modelInceptionResNetV2_base.trainable=False
x=modelInceptionResNetV2_base.output
logging.info("Primera capa con el output de la ultima capa")
x=GlobalAveragePooling2D()(x)
#dense layer 3 crea una nueva capa oculta con 512 nodos con activacion relu
logging.info("Segunda capa oculta con 512 nodos con activacion relu")
x=Dense(512,activation='relu')(x)
#Crea la ultima capa con 3 nodos y activacion softmax
logging.info("Ultima capa")
preds=Dense(120,activation='softmax')(x) #final layer with softmax activation
logging.info("Generamos el modelo con las capas nuevas")
modelInceptionResNetV2=Model(inputs=modelInceptionResNetV2_base.input,outputs=preds)
```

Figura 8. Modelo InceptionResNetV2

- **Modelo MobileNetV2**

```
#Modelo preentrenado MobileNetV2
logging.info("Obtenemos el modelo base MobileNetV2 sin la ultima capa")
modelMobileNetV2_base = keras.applications.mobilenet_v2.MobileNetV2(weights='imagenet', include_top=False)
modelMobileNetV2_base.trainable=False
x=modelMobileNetV2_base.output
logging.info("Primera capa con el output de la ultima capa")
x=GlobalAveragePooling2D()(x)
logging.info("Segunda capa oculta con 512 nodos con activacion relu")
#dense layer 3 crea una nueva capa oculta con 512 nodos con activacion relu
x=Dense(512,activation='relu')(x)
#Crea la ultima capa con 3 nodos y activacion softmax
logging.info("Ultima capa")
preds=Dense(120,activation='softmax')(x) #final layer with softmax activation
logging.info("Generamos el modelo con las capas nuevas")
modelMobileNetV2=Model(inputs=modelMobileNetV2_base.input,outputs=preds)
```

Figura 9. Modelo preentrenado MobileNetV2

Por otro lado, tal y como se ha comentado previamente, se van a entrenar únicamente las capas añadidas, para ello, se ha utilizado el *transfer Learning,* que consiste en tomar las

características aprendidas en un problema y aprovecharlas para un problema nuevo similar. De manera que, se reutilizan las características antiguas en predicciones para un nuevo conjunto de datos. Esto es posible gracias al atributo booleano que tienen todas capas & modelos: layer.trainable. De manera que, únicamente se tienen que poner a false las capas que no se quieran entrenar, y a true las nuevas entrenables.

- **Modelo ModelVGG16**

```
logging.info("Establecemos las capas del modelo preentrenado a NO entrenables")
for layer in modelVGG16.layers[:19]:
    layer.trainable=False

logging.info("Establecemos las capas nuevas a entrenables")
for layer in modelVGG16.layers[19:]:
    layer.trainable=True
```

```
INFO:root:Establecemos las capas del modelo preentrenado a NO entrenables
INFO:root:Establecemos las capas nuevas a entrenables
```

Figura 10. Establecemos las capas nuevas a entrenables y a no entrenables para el Modelo ModelVGG16.

- **Modelo ModelMobileNetV2**

```
logging.info("Establecemos las capas del modelo preentrenado a NO entrenables")
for layer in modelMobileNetV2.layers[:155]:
    layer.trainable=False
logging.info("Establecemos las capas nuevas a entrenables")
for layer in modelMobileNetV2.layers[155:]:
    layer.trainable=True
```

```
INFO:root:Establecemos las capas del modelo preentrenado a NO entrenables
INFO:root:Establecemos las capas nuevas a entrenables
```

Figura 11. Establecemos las capas nuevas a entrenables y a no entrenables para el Modelo ModelMobileNetV2.

- **Modelo ModelInceptionResNetV2**

```
logging.info("Establecemos las capas del modelo preentrenado a NO entrenables")
for layer in modelInceptionResNetV2.layers[:190]:
    layer.trainable=False
logging.info("Establecemos las capas nuevas a entrenables")
for layer in modelInceptionResNetV2.layers[190:]:
    layer.trainable=True
```

```
INFO:root:Establecemos las capas del modelo preentrenado a NO entrenables
INFO:root:Establecemos las capas nuevas a entrenables
```

Figura 12. Establecemos las capas nuevas a entrenables y a no entrenables para el Modelo ModelInceptionResNetV2.

- Preentrenamiento y entrenamiento de las capas añadidas

Los datos de las imágenes del conjunto de entrenamiento, no se pueden leer ni convertir directamente en tensores. Sin embargo, Keras proporciona métodos incorporados que pueden realizar esta tarea fácilmente. Por tanto, para el preentrenamiento de los modelos se ha utilizado la clase *ImageDataGenerator* y el *train_generator,* que nos permiten trabajar con la información de las

imágenes, y, además, van a ser muy útiles para cambiar el tamaño de las imágenes, voltearlas, etc. De esta manera, agregamos más entrenamiento a los datos, para evitar un ajuste excesivo.

A continuación, se muestra la tabla 2 y la tabla 3, con las características propias de las clases utilizadas en el preentrenamiento de los modelos.

ImageDataGenerator	
Genere lotes de datos de imágenes de tensores con aumento de datos en tiempo real. Los datos se repetirán (en lotes).	
preprocessing_function	Función que se aplicará en cada entrada. La función se ejecutará después de que la imagen cambie de tamaño y aumente. La función debe tomar un argumento: una imagen (tensor Numpy con rango 3) y debe generar un tensor Numpy con la misma forma
zoom_range	Flotante o [inferior, superior]. Rango para zoom aleatorio. Si es un flotante, [inferior, superior] = [1-zoom_range, 1 + zoom_range]
horizontal_flip	Booleano. Voltee las entradas de forma aleatoria horizontalmente.
rango_rotación	Int. Rango de grados para rotaciones aleatorias

Tabla 2. Clase ImageDataGenerator

Train Generator	
Path	El directorio debe establecerse en la ruta donde están presentes sus "n" clases de carpetas.
Target_size	Es el tamaño de sus imágenes de entrada, cada imagen cambiará de tamaño a este tamaño
color_mode	Si la imagen es en blanco y negro o en escala de grises, configure "escala de grises" o si la imagen tiene tres canales de color, configure "rgb
batch_size	No. de imágenes que se generarán desde el generador por lote
class_mode	Establezca "binario" si solo tiene dos clases para predecir, si no está establecido en "categórico", en caso de que esté desarrollando un sistema Autoencoder, tanto la entrada como la salida probablemente sean la misma imagen, para este caso, establezca a "entrada".
shuffle	Establezca True si desea mezclar el orden de la imagen que se está generando, de lo contrario, establezca False

Tabla 3. Train generator

Ejemplo código para el modelo VGG16:

```
logging.info("Utilizando ImageDataGenerator")
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_vgg16,
    zoom_range=0.2,
    rotation_range = 5,
    horizontal_flip=True)

logging.info("Generando el train_generator")
train_generator=train_datagen.flow_from_directory(path,
                                                  target_size=(256,256),
                                                  # default parameters
                                                  color_mode='rgb',
                                                  batch_size=4,
                                                  class_mode='categorical',
                                                  shuffle=True)
```

Figura 13. Preentrenamiento del modelo VGG16.

Posteriormente, se compilan los modelos y se hace uso de la función *fit_generator* para proceder al entrenamiento de los mismos. A continuación, se pueden observar las características de las funciones utilizadas para la compilación y entrenamiento del modelo, en la tabla 4, y tabla 5. Además, también se observa un ejemplo de código utilizado para el entrenamiento del modelo VGG16 en la figura 13.

Compile Method	
Optimizador	Cadena (nombre del optimizador) o instancia del optimizador. Consulte tf.keras.optimizers.
Loss	String (nombre de la función objetivo), función objetivo o instancia tf.keras.losses.Loss. Ver tf.keras.losses.
Metrics	Lista de métricas que el modelo evaluará durante el entrenamiento y las pruebas. Cada uno de estos puede ser una cadena (nombre de una función incorporada), función o una instancia de tf.keras.metrics.Metric. Consulte tf.keras.metrics.

Tabla 4. Compile Method

fit_generator	
Generator	Un generador cuya salida debe ser una lista de la forma: - (insumos, objetivos) - (entrada, objetivos, pesos_muestra) una sola salida del generador hace un solo lote y, por lo tanto, todas las matrices en la lista deben tener una longitud igual al tamaño del lote
steps_per_epoch	Especifica el número total de pasos tomados del generador tan pronto como termina una época y comienza la siguiente. Podemos calcular el valor de steps_per_epoch como el número total de muestras en su conjunto de datos dividido por el tamaño del lote
Épocas	Un número entero y número de épocas para las que queremos entrenar nuestro modelo

Tabla 5. Fit_generator

A continuación, se muestra el código implementado para el entrenamiento del modelo VGG16.

```
logging.info("Compilando el modelo")
modelVGG16.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
logging.info("Comienzo de entrenamiento")
step_size_train=train_generator.n//train_generator.batch_size
model = modelVGG16.fit_generator(generator=train_generator,
                    steps_per_epoch=step_size_train,
                    epochs=20)
```

Figura 14. Compilamos y entrenamos el modelo VGG16

COMPARACIÓN DE LOS MODELOS Y SELECCIÓN DEL MEJOR

A continuación, se muestra una tabla con la precisión obtenida por los diferentes modelos entrenados, únicamente se muestra la precisión obtenida en la etapa 20, es decir, la última precisión obtenida:

Modelo	Precisión
VGG16	0.87
InceptionResNetV2	0.87
MobileNetV2	0.92

Figura 15. Comparación de los modelos seleccionados.

Se observa como claramente el modelo que utiliza la red preentrenada **MobileNetV2** obtiene una precisión superior a todos los demás modelos estudiados, **por lo que elegimos este modelo como ganador**. A continuación, se puede observar el resultado de la precisión obtenida en cada etapa por cada modelo:

- **Modelo VGG16**

  Resultados del entrenamiento:

Epoch 1/20
2655/2655 [==============================] - 1059s 399ms/step - loss: 0.6733 - accuracy: 0.82 83
Epoch 2/20
2655/2655 [==============================] - 1059s 399ms/step - loss: 0.7151 - accuracy: 0.82 83
Epoch 3/20
2655/2655 [==============================] - 1059s 399ms/step - loss: 0.6755 - accuracy: 0.82 87
Epoch 4/20
2655/2655 [==============================] - 1062s 400ms/step - loss: 0.6658 - accuracy: 0.83 35
Epoch 5/20
2655/2655 [==============================] - 1062s 400ms/step - loss: 0.6374 - accuracy: 0.84 04
Epoch 6/20
2655/2655 [==============================] - 1064s 401ms/step - loss: 0.6546 - accuracy: 0.84 23
Epoch 7/20
2655/2655 [==============================] - 1065s 401ms/step - loss: 0.6296 - accuracy: 0.84 89
Epoch 8/20
2655/2655 [==============================] - 1065s 401ms/step - loss: 0.6355 - accuracy: 0.84 75
Epoch 9/20
2655/2655 [==============================] - 1063s 400ms/step - loss: 0.6146 - accuracy: 0.85 31
Epoch 10/20

2655/2655 [==============================] - 1064s 401ms/step - loss: 0.6247 - accuracy: 0.85 68
Epoch 11/20
2655/2655 [==============================] - 1066s 401ms/step - loss: 0.6350 - accuracy: 0.85 39
Epoch 12/20
2655/2655 [==============================] - 1065s 401ms/step - loss: 0.6159 - accuracy: 0.85 74
Epoch 13/20
2655/2655 [==============================] - 1063s 401ms/step - loss: 0.6508 - accuracy: 0.85 72
Epoch 14/20
2655/2655 [==============================] - 1065s 401ms/step - loss: 0.5847 - accuracy: 0.86 34
Epoch 15/20
2655/2655 [==============================] - 1063s 400ms/step - loss: 0.5974 - accuracy: 0.86 41
Epoch 16/20
2655/2655 [==============================] - 1063s 400ms/step - loss: 0.6036 - accuracy: 0.86 69
Epoch 17/20
2655/2655 [==============================] - 1062s 400ms/step - loss: 0.6079 - accuracy: 0.87 05
Epoch 18/20
2655/2655 [==============================] - 1063s 400ms/step - loss: 0.6315 - accuracy: 0.86 34
Epoch 19/20
2655/2655 [==============================] - 1062s 400ms/step - loss: 0.6147 - accuracy: 0.86 98
Epoch 20/20
2655/2655 [==============================] - 1065s 401ms/step - loss: 0.5761 - accuracy: 0.87 66

- **Modelo InceptionResNetV2**

  Resultados del entrenamiento:

Epoch 1/20
5310/5310 [==============================] - 369s 69ms/step - loss: 0.9341 - accuracy: 0.7515
Epoch 2/20
5310/5310 [==============================] - 369s 69ms/step - loss: 0.8763 - accuracy: 0.7664
Epoch 3/20
5310/5310 [==============================] - 368s 69ms/step - loss: 0.8536 - accuracy: 0.7745
Epoch 4/20
5310/5310 [==============================] - 367s 69ms/step - loss: 0.8376 - accuracy: 0.7866
Epoch 5/20
5310/5310 [==============================] - 368s 69ms/step - loss: 0.7926 - accuracy: 0.7893
Epoch 6/20
5310/5310 [==============================] - 367s 69ms/step - loss: 0.7865 - accuracy: 0.8005
Epoch 7/20
5310/5310 [==============================] - 368s 69ms/step - loss: 0.7600 - accuracy: 0.8060
Epoch 8/20
5310/5310 [==============================] - 370s 70ms/step - loss: 0.7289 - accuracy: 0.8182

Epoch 9/20
5310/5310 [==========] - 370s 70ms/step - loss: 0.7355 - accuracy: 0.8198
Epoch 10/20
5310/5310 [==========] - 368s 69ms/step - loss: 0.6922 - accuracy: 0.8336
Epoch 11/20
5310/5310 [==========] - 367s 69ms/step - loss: 0.7020 - accuracy: 0.8298
Epoch 12/20
5310/5310 [==========] - 367s 69ms/step - loss: 0.6821 - accuracy: 0.8394
Epoch 13/20
5310/5310 [==========] - 367s 69ms/step - loss: 0.6946 - accuracy: 0.8432
Epoch 14/20
5310/5310 [==========] - 369s 70ms/step - loss: 0.6715 - accuracy: 0.8464
Epoch 15/20
5310/5310 [==========] - 369s 69ms/step - loss: 0.6331 - accuracy: 0.8549
Epoch 16/20
5310/5310 [==========] - 369s 69ms/step - loss: 0.6546 - accuracy: 0.8579
Epoch 17/20
5310/5310 [==========] - 369s 69ms/step - loss: 0.6190 - accuracy: 0.8623
Epoch 18/20
5310/5310 [==========] - 367s 69ms/step - loss: 0.6068 - accuracy: 0.8651
Epoch 19/20
5310/5310 [==========] - 367s 69ms/step - loss: 0.6195 - accuracy: 0.8654
Epoch 20/20
5310/5310 [==========] - 367s 69ms/step - loss: 0.6009 - accuracy: 0.8761

- **Modelo MobileNetV2**

Resultados del entrenamiento:

Epoch 1/20
5310/5310 [==========] - 213s 40ms/step - loss: 1.2467 - accuracy: 0.6503
Epoch 2/20
5310/5310 [==========] - 212s 40ms/step - loss: 0.9041 - accuracy: 0.7337
Epoch 3/20
5310/5310 [==========] - 212s 40ms/step - loss: 0.7751 - accuracy: 0.7717
Epoch 4/20
5310/5310 [==========] - 212s 40ms/step - loss: 0.6829 - accuracy: 0.7987
Epoch 5/20
5310/5310 [==========] - 213s 40ms/step - loss: 0.6420 - accuracy: 0.8146
Epoch 6/20
5310/5310 [==========] - 214s 40ms/step - loss: 0.5777 - accuracy: 0.8347
Epoch 7/20
5310/5310 [==========] - 212s 40ms/step - loss: 0.5474 - accuracy: 0.8483
Epoch 8/20
5310/5310 [==========] - 214s 40ms/step - loss: 0.5126 - accuracy: 0.8565
Epoch 9/20
5310/5310 [==========] - 212s 40ms/step - loss: 0.4898 - accuracy: 0.8695
Epoch 10/20
5310/5310 [==========] - 212s 40ms/step - loss: 0.4614 - accuracy: 0.8771

```
Epoch 11/20
5310/5310 [==============================] - 212s 40ms/step - loss: 0.4538 - accuracy: 0.8837
Epoch 12/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.4312 - accuracy: 0.8914
Epoch 13/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.4179 - accuracy: 0.8957
Epoch 14/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.4047 - accuracy: 0.9032
Epoch 15/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.4105 - accuracy: 0.9056
Epoch 16/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.3910 - accuracy: 0.9088
Epoch 17/20
5310/5310 [==============================] - 214s 40ms/step - loss: 0.3958 - accuracy: 0.9169
Epoch 18/20
5310/5310 [==============================] - 214s 40ms/step - loss: 0.3963 - accuracy: 0.9153
Epoch 19/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.3939 - accuracy: 0.9194
Epoch 20/20
5310/5310 [==============================] - 213s 40ms/step - loss: 0.3874 - accuracy: 0.9234
```

Generación de un Apache Tomcat en el servidor de Amazon Linux de Deep Learning.

Ha sido necesario el despliegue de un tomcat en el servidor de amazon Linux para conectar el Servicio REST con la parte de Big Data. De manera que, cuando un usuario introduzca una imagen en la página web, el servicio REST, será el encargado de devolver una ruta concreta de la imagen. De manera que, el servicio Rest tendrá que preguntar al tomcat de amazon linux de Big Data, por la URL en cuestión. Dicha URL será recibida por el modelo generado, y, en consecuencia, se generará una respuesta con el tipo de raza de perro correspondiente a dicha imagen. De manera que, la respuesta será enviada desde el tomcat de big data, hasta el servicio rest, y éste hará posible que el usuario visualice el tipo de raza de perro por la consola del sitio web implementado. Todo ello, se puede visualizar en la imagen:

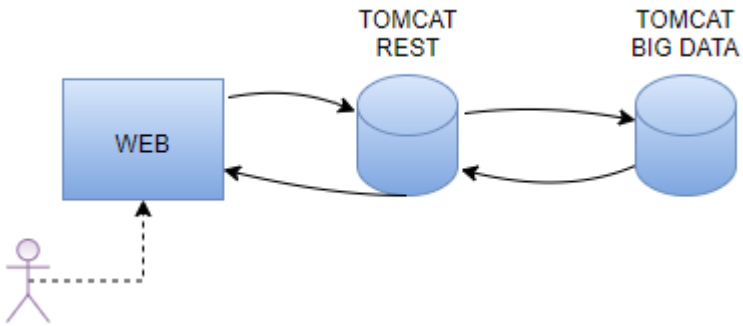


Figura 16. . Esquema sobre la estructura y conexión del servicio Rest y la aplicación Big Data

De manera que, cuando el usuario acceda a la parte web para identificar la raza de su perro:

Figura 17. Sección de sitio web

Figura 18. Pasos necesarios en el sitio web desarrollado para la identificación de la raza de perro.

Figura 19. Obtención de la raza de perro

Predicciones con los datos de prueba

Después de haber seleccionado el mejor modelo, se procede a continuación a su evaluación, y a la elaboración de las predicciones del conjunto de prueba.

Para visualizar la precisión del modelo, se ha generado una gráfica que representa la precisión obtenida por cada época. De manera que, se comprueba que el modelo ganador **MobileNetV2** está alcanzado el 0.92 cuando se alcanza la etapa 20.

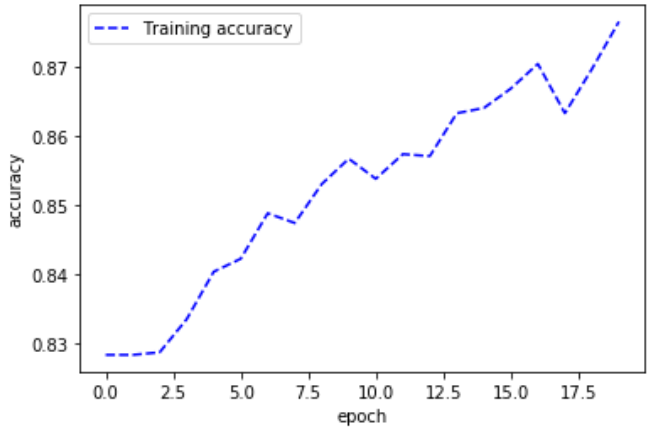


Figura 20. Training accuracy del modelo ganador MobileNetV2

Vamos a utilizar nuestro modelo para predecir las imágenes que hemos guardado anteriormente, las cuales no han sido utilizadas en el entrenamiento del modelo. Para ello, se ha generado una función *predict_image* que recibe como parámetro el path referente a la imagen de prueba. De manera que, esta función será la que reciba el path de la imagen que el usuario introduzca por el sitio web. A continuación, se observa el código realizado para la realización de esta función en la figura 17.

- **Código implementado para la función *predict_image*:**

```
class_dict = {v:k for k, v in train_generator.class_indices.items()}

def predict_image(path):
    img = image.load_img(path)
    img = img.resize((224, 224))
    data = expand_dims(image.img_to_array(img), 0)
    data = preprocess_mobilenet(data)
    preds = modelMobileNetV2.predict(data)
    pred = np.argmax(preds)
    pred = class_dict[pred]
    print(pred)
    return img
```

Figura 21. Función predict_image

No obstante, la función comentada forma parte de un script que es capaz de procesar la imagen que recibe desde el rest. Dicho Script, se ejecuta dentro del tomcat del servicio de Big Data, ya que debe de cargar el modelo implementado, y dicho TomCat está configurado para la realización de las funciones o tareas específicas de Deep Learning. Es importante comentar que, debido principalmente a las configuraciones intrínsecas del servidor de big data para la ejecución del modelo, no ha sido viable el uso de un solo tomcat que se encargue del envío y recepción de

las peticiones por parte de blockchain y big data. Ya que, el tomcat del rest no presentaba la misma configuración.

A continuación, se puede observar el código referente al script implementado:

```
import sys
import os
from keras.models import load_model
import urllib.request
from PIL import Image
from keras.models import load_model
import urllib.request
from PIL import Image
import pandas as pd
import numpy as np
from numpy import expand_dims
import os
import sys
import re
import shutil
import keras
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet, VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.mobilenet import preprocess_input as preprocess_mobilenet
from tensorflow.keras.applications.densenet import preprocess_input as preprocess_densenet
from tensorflow.keras.applications.nasnet import preprocess_input as preprocess_nasnet
from tensorflow.keras.applications.vgg16 import preprocess_input as preprocess_vgg16
from tensorflow.keras.applications.inception_resnet_v2 import preprocess_input as
preprocess_inceptionResNetV2
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import logging
from random import randrange
import pydot
from keras.utils.vis_utils import plot_model
from IPython.display import Image
from IPython.core.display import HTML
import collections
import math
import os
import time
import tensorflow as fl
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.models import Model, Sequential
from PIL import Image
```

```
model = keras.models.load_model('/home/ec2-user/Notebooks/modelMobileNetV2.h5')
path = sys.argv[1]
print("El path es: ", path)
images = Image.open(urllib.request.urlopen(path))
images

def predict_image(path):
    img = path.resize((224, 224))
    data = expand_dims(path, 0)
    data = preprocess_mobilenet(data)
    preds = model.predict(data)
    pred = np.argmax(preds)
    return pred

indice = predict_image(images)

#Obtenemos las clases
classes = []
count = 0
data = []

datasetPath = "/home/ec2-user/images/Images"
logging.info(" Buscando la informacion del dataset")
for root, dirnames, filenames in os.walk(datasetPath):
    if(not classes):
        classes = dirnames
        continue
    data.append((classes[count],filenames))
    count+=1

print("raza:"+ classes[indice])
```

En resumen, estamos obteniendo buenos resultados por parte del algoritmo realizado para la generación de las predicciones. De manera que, se asegura con una probabilidad superior al 90%, de que el usuario no va a recibir como respuesta una raza de perro que no se corresponda con la raza del perro de la imagen introducida por el sitio web.

Conclusiones y líneas futuras

CONCLUSIONES

En el presente Anexo V, sobre Big Data Canino, se ha desarrollado un modelo de red neuronal convolucional con una precisión de 0.92. Para ello, se han llevado a cabo los siguientes pasos, coincidentes con el cumplimiento de los objetivos iniciales:

- Se ha utilizado un dataset de 22.580 imágenes, llamado Standford Dog Dataset. Obtenido del siguiente sitio web https://www.kaggle.com/jessicali9530/stanford-dogs-dataset.
- Obtención del conjunto de datos de prueba y entrenamiento. Para ello, se ha desarrollado un script que introduce 83 imágenes de cada tipo de raza en una carpeta llamada Images_Test. Por lo tanto, finalmente se tienen 9960 imágenes destinadas al conjunto de prueba, y 10620, destinadas al conjunto de entrenamiento.
- Búsqueda de los mejores modelos preentrenados, y entrenamiento con distintos modelos. Se han evaluado los modelos presentes en la página oficial de Keras, y se han entrenado en un servidor de Amazon de pago para poder compararlos y seleccionar el mejor de ellos. Los modelos seleccionados han sido: VGG16, InceptionResNetV2 y MobileNetV2.
- Comparación de los modelos entrenados y obtención del mejor modelo. Se han comparado los tres modelos seleccionados y se ha seleccionado el modelo mobilenet, como mejor modelo, por tener mayor precisión.
- Realización de las predicciones con los datos de prueba, y análisis de los resultados.
- Generar un script que permita la obtención de la predicción de la imagen que el usuario introduzca en la página web.

LÍNEAS FUTURAS

- Generación e implementación de un proceso automático que corra en el servidor del Servicio Rest, cuyo principal objetivo sea entrenar el modelo a una hora concreta, de un día particular. De manera que, el modelo sería capaz de ir mejorándose o aprendiendo por sí sólo, con las nuevas imágenes introducidas por el usuario, a través de la página web.

Bibliografía

ARTEAGA, G. J. (2015). *APLICACIÓN DEL APRENDIZAJE PROFUNDO ("DEEP LEARNING") AL.* SANTIAGO DE CALI .

Bagnato, J. I. (Noviembre de 2018). *Aprende Maching Learning*. Obtenido de https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

Cárdenas, I. R. (22 de Enero de 2018). *EEP LEARNING PARA LA DETECCIÓN DE PEATONES Y VEHÍCULOS.* Obtenido de http://148.215.1.182/bitstream/handle/20.500.11799/70995/tesisfinalRVC-ilovepdf-compressed%20%281%29.pdf?sequence=1&isAllowed=y

Egea, J. (1994). Redes neuronales: concepto, fundamento y aplicaciones en el laboratorio clínico. *Química clínica*, 13 (5): 221-228.

García, E. M. (Septiembre de 2019). *TFG.* Obtenido de https://e-archivo.uc3m.es/bitstream/handle/10016/30357/TFG_Elena_Martinez_Garcia_2019.pdf?sequence=1&isAllowed=y

J. A. Pérez-Carrasco, C. S.-G. (s.f.). *RED NEURONAL CONVOLUCIONAL.* Obtenido de https://idus.us.es/bitstream/handle/11441/79308/RED%20NEURONAL.pdf?sequence=1&isAllowed=y

Li, J. (s.f.). *kaggle*. Obtenido de https://www.kaggle.com/jessicali9530/stanford-dogs-dataset

Modelos Keras. (s.f.). Obtenido de https://keras.io/api/applications/),

Pacheco, M. A. (Agosto de 2017). *Identificaci´on de sistemas no lineales con redes neuronales convolucionales.* Obtenido de https://www.ctrl.cinvestav.mx/~yuw/pdf/MaTesMLP.pdf