Práctica 2
Deep Learning para multi-clasificación

Sistemas Inteligentes para la Gestión en la Empresa

25 de mayo de 2019

Felipe Peiró Garrido
felipepg@correo.ugr.es

Juan Carlos Serrano Pérez
jcsp0003@correo.ugr.es

Pedro Manuel Gómez-Portillo López
gomezportillo@correo.ugr.es

Índice

1. Fundamentos teóricos

A continuación se presenta una introducción a la práctica y se justifican las tecnologías utilizadas a lo largo de la misma.

1.1. Introducción

El *Deep Learning* es una de las áreas de investigación más populares dentro del campo de la Inteligencia Artificial. La mayoría de las nuevas investigaciones que se realizan trabajan con modelos basados en las de *Deep Learning*, ya que gracias a ellas se han logrado resultados sorprendentes en campos como Procesamiento del Lenguaje Natural y Visión por Computadora.

Este área hace uso de estructuras jerárquicas de redes neuronales artificiales que se construyen de una forma similar a la estructura neuronal del cerebro humano y que permite lograr el aprendizaje a través de diversas capas ejecutando un análisis de datos de manera no lineal.

Los logros del *Deep Learning* son muy diversos, en los últimos años el *Deep Learning* ha producido toda una revolución en el campo del *Machine Learning*, con resultados notables en todos los problemas que requieren de ver y escuchar, problemas que implican habilidades que son naturales para los seres humanos pero que son muy difíciles para las máquinas. Ejemplos de esto son los asistentes virtuales como Siri o Alexa, la clasificación de imágenes o el ganar a personas en juegos y videojuegos.

1.2. Propósito de la práctica

En este trabajo se pondrá en práctica la capacidad de clasificación mediante *Deep Learning*. Se utilizará el conjunto de datos de predicción de adopción de PetFinder.my de Kaggle[1] con el que se crearán redes neuronales para intentar clasificar el mayor número de mascotas posible. Este conjunto de datos consta de lo siguiente:

- Un conjunto de entrenamiento de 46652 imágenes
- Un conjunto de prueba de 11659 imágenes

Para la realización de la red neuronales se hará uso de un framework donde se decidirá el número de capas a utilizar así como la interconexión entre ellas.

[1] https://www.kaggle.com/c/petfinder-adoption-prediction/overview/evaluation

1.3. Elección del framework

Para la realización de la práctica se debió discutir entre escoger un framework de desarrollo de redes neuronales o realizar un desarrollo manual de estas. Tras sucesivos intentos de un desarrollo manual se ha optado por utilizar un framework de desarrollo.

Actualmente existen muchos frameworks y librerías que trabajan con estas técnicas, los más importantes son TensorFlow y Keras, aunque hay otros que merecen ser destacados como Pytorch, Stickit-learn, Lasagne, DSSTINE, MXNet, DL4J, o Microsoft Cognitive Toolkit.

Keras[2] es una biblioteca de generación de redes neuronales de código abierto escrita en Python. Está diseñado para permitir una rápida experimentación con redes neuronales profundas, se enfoca en ser fácil de usar, modular y extensible. Keras fue concebido para ser una interfaz en lugar de un marco de aprendizaje automático autónomo. Ofrece un conjunto de abstracciones más intuitivo y de mayor nivel que facilita el desarrollo de modelos de aprendizaje profundo, independientemente del backend computacional utilizado.

TensorFlow[3], a diferencia de Keras, es un framework de código abierto desarrollado por Google que se utiliza para realizar cálculos numéricos mediante diagramas de flujo de datos en los que los nodos de los diagramas representan operaciones matemáticas y las aristas reflejan las matrices de datos multidimensionales (tensores) comunicadas entre ellas.

A continuación se muestran las similitudes y diferencias entre estos frameworks que han ayudado a decidir sobre cuál se va a utilizar.

Propiedad	Descripción
Prototipado rápido	Tensorflow permite una mayor personalización en el desarrollo de redes neuronales y ayuda a entender con mejor precisión cómo funcionan. Sin embargo, Keras ofrece una rápida construcción y entrenamiento de redes neuronales utilizando muy pocas líneas de código siendo, además, más trivial que Tensorflow.
Flexibilidad	A veces se necesita definir algo propio como una función de coste, una métrica o una capa. Tensorflow al ser una biblioteca de más bajo nivel ofrece mayor flexibilidad y ajuste en comparación con Keras. Aún así, Keras ha sido diseñado para poder implementar casi todo lo necesario.

[2] https://keras.io/
[3] https://www.tensorflow.org

Propiedad	Descripción
Funcionalidad	Aunque TensorFlow ofrece operaciones más avanzadas en comparación a Keras, este proporciona todas las funcionalidades de propósito general para construir modelos de aprendizaje profundo, siendo más fácil de utilizar. Esto es muy útil si se está realizando una investigación o desarrollando algún tipo especial de modelos de aprendizaje profundo.
Control	Un mayor control sobre las redes neuronales implica una mayor comprensión sobre lo que está ocurriendo con ellas. TensorFlow permite obtener un control sobre las redes neuronales ya que permite controlar cada aspecto que actúa sobre ellas como por ejemplo las operaciones sobre pesos o gradientes.

Teniendo en cuenta esta serie de decisiones se ha optado por la utilización de Keras para la realización de la práctica.

1.4. Computación en la nube

Debido al alto coste computacional que requiere la ejecución de redes neuronales, y en especial de aquellos que trabajan con imágenes, se ha utilizado la infraestructura cloud de Google para entrenar el modelo. La máquina, de tipo *Deep Learning VM,* cuenta con las siguiente configuración y especificaciones técnicas.

- **Procesador.** Intel Broadwell x2
- **Caché.** 13GB Memoria RAM
- **Tarjeta gráfica.** NVIDIA Tesla K80
- **Tipo de disco duro**. SSD
- **Localización.** Oeste de Estados Unidos

2. Descripción de las redes empleadas

2.1. Configuración de capas

Las configuración de la red utilizada finalmente, formada por siete capas, ha sido la siguiente.

1. **`layer_conv_2d`.** Es una capa convolutiva con la función de activación de unidad lineal rectificada, o `relu`, una ventana convolutiva de 3x3 y un tamaño de entrada de 150x150.

2. **`layer_max_pooling_2d`.** Es una capa de operación de agrupación máxima con un tamaño de pool de 2x2.

3. **`layer_conv_2d`.** Es una capa convolutiva con la función de activación `relu` y una ventana convolutiva de 3x3.

4. **`layer_max_pooling_2d`.** Es una capa de operación de agrupación máxima con un tamaño de pool de 2x2.

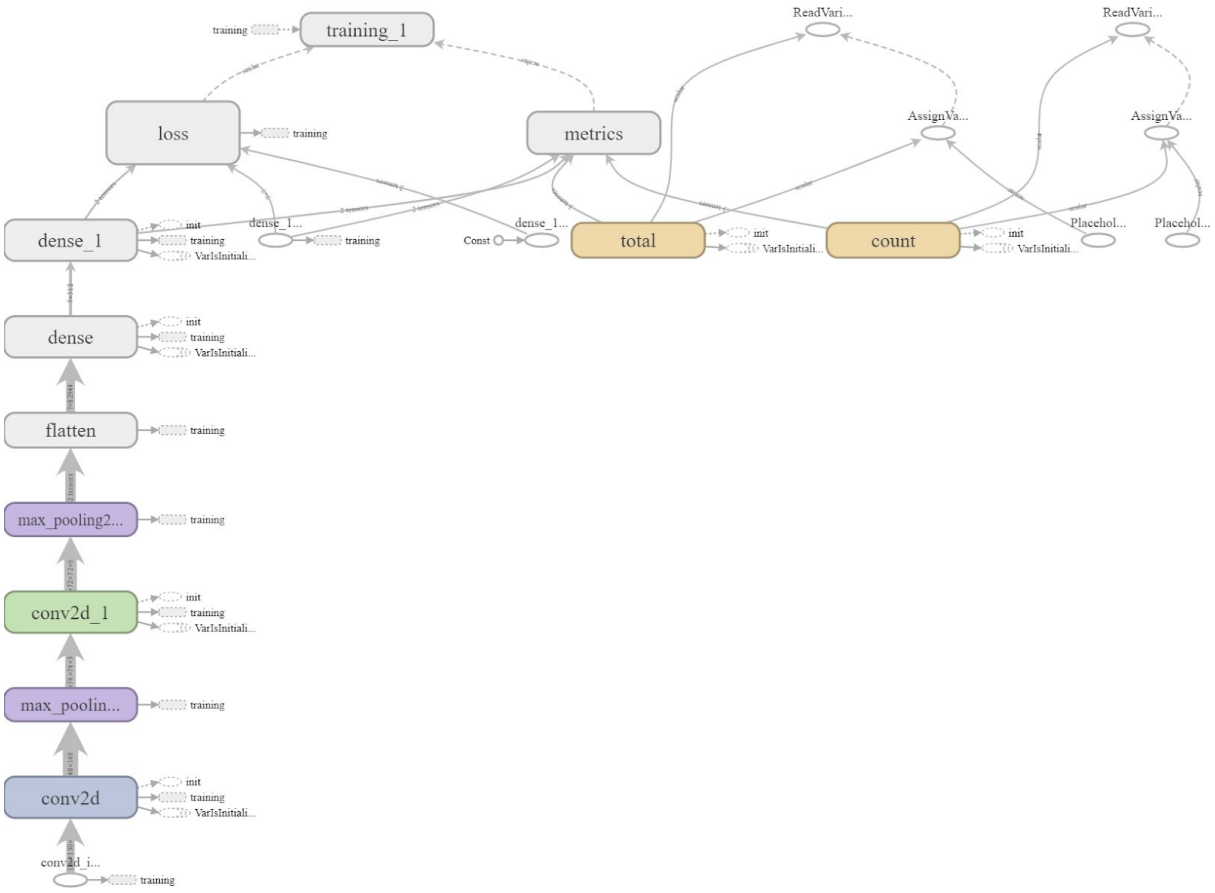
5. **`layer_flatten`.** Esta capa convierte los datos de entrada de una matriz bidimensional a un vector unidimensional para que puedan ser procesados por capas totalmente conectadas.

6. **`layer_dense`.** Es una capa totalmente conectada con 512 neuronas y una función de activación `relu`.

7. **`layer_dense`.** Es una capa totalmente conectada con 5 neuronas, el número de clases a predecir, y una función de activación `softmax`, que asigna probabilidades decimales a cada clase en un caso de clases múltiples.

Como en trabajo de teoría se ha investigado sobre herramientas de visualización de redes neuronales para poder ilustrar la parte práctica, se ha utilizado **TensorBoard** para obtener una visión gráfica del modelo. A continuación se muestra el diagrama generado por esta herramienta en el que puede verse la estructura de las redes empleadas de una manera más gráfica.

Se adjunta el archivo PNG del diagrama con más resolución.



2.2. Creación de los conjuntos de entrenamiento y test

El objetivo inicial a la hora de la realización de la práctica era utilizar los conjuntos de entrenamiento y de test proporcionados y a través de la función

```
predictions <- predict_generator(
  model,
  test_generator,
  steps = test_generator$n/test_generator$batch_size
)
```

generar las predicciones y exportarlas a un fichero CSV con el formato:

```
PetID, AdoptionSpeed
```

Pero debido a incompatibilidades de los resultados se optó por dividir el conjunto de entrenamiento (del cual se conocen las clases) en dos conjuntos entrenamiento (80% de los datos) y test (el 20% restante de los datos).

Para la realización de la división del conjunto de imágenes en estos dos subconjuntos a la hora de crear el generador de datos de imagen hacemos uso del parámetro `validation_split` para la división de los datos.

2.3. Función de coste

El objetivo del entrenamiento de una red neuronal es obtener el conjunto de ponderaciones que produce el menor error sobre el conjunto de entrenamiento. Para calcular el *loss*, que debería ser lo más cerca a 0 posible, se ha utilizado la función de coste `categorical_crossentropy`, que se utiliza para calcular la entropía cruzada de un conjunto de datos categóricos como es el nuestro. Esta entropía permite calcular los mejores valores de las ponderaciones e inclinaciones.

El error de entropía cruzada para el vector v de salida de una red neuronal y un vector t de salida de prueba se calcula al determinar la suma negativa del producto de cada componente del vector v y el componente del vector t[4].

2.4. Algoritmo de optimización

Para la optimización se utiliza la función `optimizer_rmsprop` y los parámetros con los que hemos obtenido un mejor resultado son:

- Tasa de aprendizaje 0.0001
- Tasa con la que decae el aprendizaje con cada actualización: 1e-6

```
optimizer = optimizer_rmsprop(lr = 0.0001, decay = 1e-6),
```

[4] https://msdn.microsoft.com/es-es/magazine/jj190808.aspx

3. Discusión de resultados

En esta sección se presentan los resultados finales obtenidos tras la ejecución del modelo sobre los datos de test. Cabe destacar que, como la ejecución final de la práctica ha sido realizada en las infraestructuras cloud de Google, los tiempos de ejecución han sido mucho menores que de haberlo ejecutado localmente en alguno de nuestros portátiles.

Esta máquina cuenta con una GPU NVIDIA Tesla K80 y un disco duro disco duro SSD. Al haber realizado todas las ejecuciones sólamente en una máquina se puede saber cuántos créditos se han consumido; en total, han sido 24.89$ teniendo en cuenta tanto el tiempo que ha estado apagada como ejecutándose.

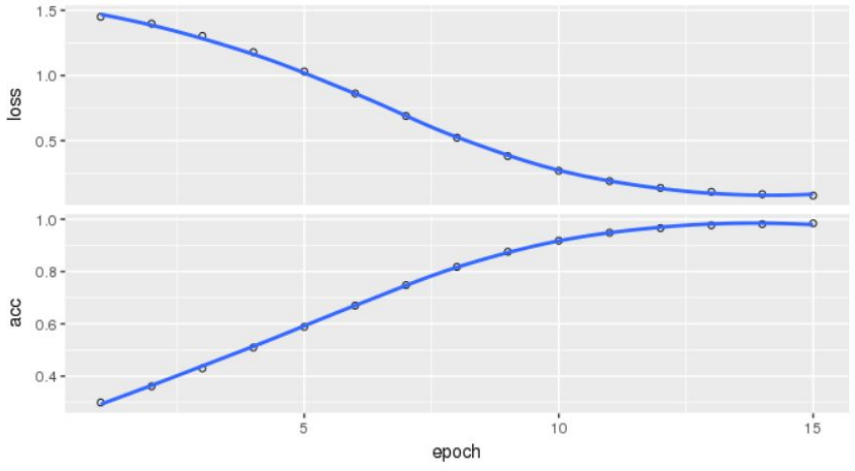
La evolución de los valores *accuracy* y *loss* a lo largo del entrenamiento, que ha durado 1 hora y 56 minutos, pueden verse en la tabla inferior, que también se adjunta para poder ser vista con mayor resolución.

Iteración	*loss*	*accuracy*
1	1.4612	0.2906
2	1.4165	0.3413
3	1.3417	0.4056
4	1.2430	0.4723
5	1.1238	0.5416

Iteración	*loss*	*accuracy*
6	0.9825	0.6165
7	0.8291	0.6863
8	0.6690	0.7589
9	0.5191	0.8202
10	0.3854	0.8741

Iteración	*loss*	*accuracy*
11	0.2759	0.9179
12	0.1954	0.9471
13	0.1413	0.9659
14	0.1076	0.9761
15	0.0885	0.9814

En el siguiente gráfico se representan los valores de la tabla:



Tras de ejecución final se ha obtenido un **porcentaje de aciertos del 98,14% sobre el conjunto de entrenamiento y del 28.11% sobre el conjunto de prueba**.

4. Conclusiones

Esta práctica nos ha servido como primera toma de contacto tanto para el uso de redes neuronales como su despliegue en servidores cloud computing con potentes recursos. Ésto nos ha permitido reducir los tiempos de espera en la fase de entrenamiento, algo que en proyectos de esta naturaleza se vuelve muy importante, dándonos tiempo suficiente para realizar más pruebas.

Por otro lado, y compara con la primera práctica, la temática de esta ha sido más interesante, ya que nos ha resultado más motivador trabajar con fotografías de perretes que con los crípticos datos de Santander, aunque a nivel práctico pueda ser más importante los segundos.

En conclusión, estamos muy contentos de haber realizado esta práctica y de haber podido aplicar los conocimientos adquiridos gracias al trabajo de teoría en la misma, permitiéndonos comprender mejor las redes neuronales con las que hemos trabajado.

5. Webgrafía

A continuación se presentan los principales recursos web con los que se han trabajado y que se han consultado para realizar la práctica.

- GitHub. (2019). *jgromero/sige2019*. [online] Available at: https://github.com/jgromero/sige2019 [Accessed 8 Jun. 2019].
- Aakash Nain, A Medium Corporation (2019). *TensorFlow or Keras? Which one should I learn?* [online] Medium. Available at: https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0 [Accessed 8 Jun. 2019].
- Link.medium.com. (2019). [online] Available at: https://link.medium.com/y0vF9nxNSW [Accessed 8 Jun. 2019].
- correctly, K. (2019). *Keras Linear regression model from images not predicting correctly*. [online] Stack Overflow. Available at: https://stackoverflow.com/q/56452587/3594238 [Accessed 8 Jun. 2019].
- Keras.rstudio.com. (2019). *fine_tuning*. [online] Available at: https://keras.rstudio.com/articles/examples/fine_tuning.html [Accessed 8 Jun. 2019].
- Teschner, F. (2019). *Transfer Learning with augmented Data for Logo Detection*. [online] Flovv.github.io. Available at: http://flovv.github.io/Logo_detection_transfer_learning_part2/ [Accessed 8 Jun. 2019].
- ImageDataGenerator, K. (2019). *Keras split train test set when using ImageDataGenerator*. [online] Stack Overflow. Available at: https://stackoverflow.com/q/42443936/3594238 [Accessed 8 Jun. 2019].
- Elsinghorst, D. (2019). *It's that easy! Image classification with keras in roughly 100 lines of code.*. [online] Shirin's playgRound. Available at: https://shirinsplayground.netlify.com/2018/06/keras_fruits/ [Accessed 8 Jun. 2019].
- RStudio Support. (2019). *Getting Started with RStudio Server Pro for GCP*. [online] Available at:https://support.rstudio.com/hc/en-us/articles/115010260627-Getting-Started-with-RStudio-Server-Pro-for-GCP [Accessed 8 Jun. 2019].
- Kuhn, M. (2019). *The caret Package*. [online] Topepo.github.io. Available at: http://topepo.github.io/caret/index.html [Accessed 8 Jun. 2019].