Pistolas vs Smartphones con Deep Learning

Minería de datos: aspectos avanzados

Francisco Manuel García Moreno
Miguel López Campos

Máster Universitario Oficial en Ciencia de Datos e Ingeniería de Computadores

UNIVERSIDAD DE GRANADA

2 de abril de 2018

Índice general

Índice de figuras

Capítulo 1

Clasificación con modelos de Deep Learning

1.1. Arquitectura del modelo

A continuación, en Figura 1.1 presentamos la arquitectura que hemos utilizado para nuestro modelo implementado en *Keras*, que mejores resultados ha reportado.

La arquitectura que hemos utilizado, después de consultar literatura relacionada con la temática del presente trabajo [1], [2], es la arquitectura VGG-16 [3], con pesos de *ImageNet*, ya que se habían conseguido resultados muy competentes en estudios previos1. Además, hemos usado *fine-tune* y *data augmentation*. Por tanto, la arquitectura usada se compone de las siguientes capas:

1. Una entrada para las imágenes con dimensión $128x128x3$, con tres canales para los colores RGB.
2. Dos capas bidimensional convolucionales de $3x3$ (filtro de $3x3$ píxeles) con 64 neuronas, con activación RELU.
3. Una capa de *pooling* (usada habitualmente entre sucesivas capas convolucionales).
4. Dos capas convolucionales 2D de $3x3$ con 128 neuronas y con activación RELU.
5. Una capa de *pooling*.
5. Tres capas convolucionales 2D de $3x3$ con 256 neuronas y con activación RELU.
6. Una capa de *pooling*.
7. Tres capas convolucionales 2D de $3x3$ con 512 neuronas y con activación RELU.
8. Una capa de *pooling*.

A continuación, le conectamos una capa completamente conectada de 1024 neuronas y activación RELU y, posteriormente, una capa de *dropout*, para prevenir el sobreajuste desactivando aleatoriamente ciertas neuronas [4]. En este sentido, realizando diferentes experimentos se ajustó el valor del *dropout* hasta 0,7 (un valor *agresivo*) debido a que observamos cierta diferencia entre la precisión del conjunto

de entrenamiento y de validación (por lo que presuponíamos que aún existía cierto sobreajuste). Y por último, la capa final completamente conectada, con 2 neuronas (número de clases del problema de clasificación actual: pistolas y smartphones) y la función típica de activación *SoftMax* usada en esta última capa.

Además, después de la capa complementamente conectada de 1024 neuronas, se probó a incluir una capa de *Batch Normalization*, ya que es usada ampliamente en la literatura después de estas capas completamente conectadas, forzando las activaciones a través de la red a tomar una distribución *gaussiana* al comienzo del entrenamiento [5]. Desafortunadamente, no mejoraraba el rendimiento del modelo.

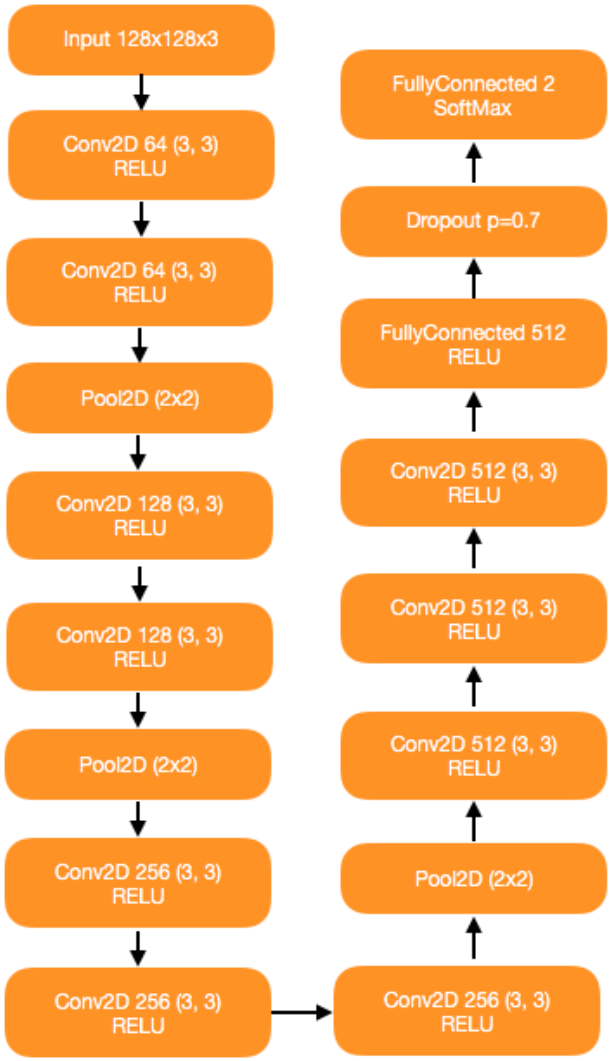

Figura 1.1: Arquitectura del modelo

1.2. Data Augmentation

Las técnicas de Data Augmentation tienen como función principal la de crear nuevas imágenes de entrenamiento a partir de las ya existentes realizando transformaciones sobre estas. Estas transformaciones pueden ser transformaciones de *zoom*, giros a la imagen, agregación de ruido, desenfoques, etc. Con estas técnicas se pretende que el modelo generalice mejor, primero por medio de la ampliación de Training y también por medio del incremento de la variedad de imágenes referentes a una clase (Pistol o Smartphone en nuestro caso). En las Figuras 1.2 y 1.3 se muestran 2 ejemplos de imágenes generadas por una librería de augmentation.

Figura 1.2: Imagen tras data augmentation (Flip)

Figura 1.3: Imagen tras data augmentation (Zoom In)

En nuestro caso particular, hemos empleado una utilidad de la librería *Keras* que ya implementa Data Augmentation. Esta utilidad, dentro de keras.preprocessing.image se llama ImageDataGenerator. Empleamos como parámetros para el augmentation rescale=1./255, rotation_range=20, shear_range=0.2, zoom_range=0.2, horizontal_flip=True, fill_mode='nearest' y para el conjunto de Test redimensionamos todas a 1./255 (igual que las de Train).

1.3. Evaluación

El modelo se entrenó en 30 etapas, usando CUDA en una GPU NVidia GTX 1060, empleando un *train_generator* con *data aumentation* (puede consultarse en el *script main.py*)*test_generator*. Con objeto de validar el modelo, se empleó *Hold-out validation* dejando un 80 % para *training* y un 20 % para validación. A continuación, en Figura 1.4 y Figura 1.5 se presentan los resultados de la evaluación del modelo, según la métrica de *accuracy* y *loss*, respectivamente, obteniendo un 100 % de *accuracy* en *training*, un 99,22 % en *validation* y un 99,749 % en el *test* público de *Kaggle*.

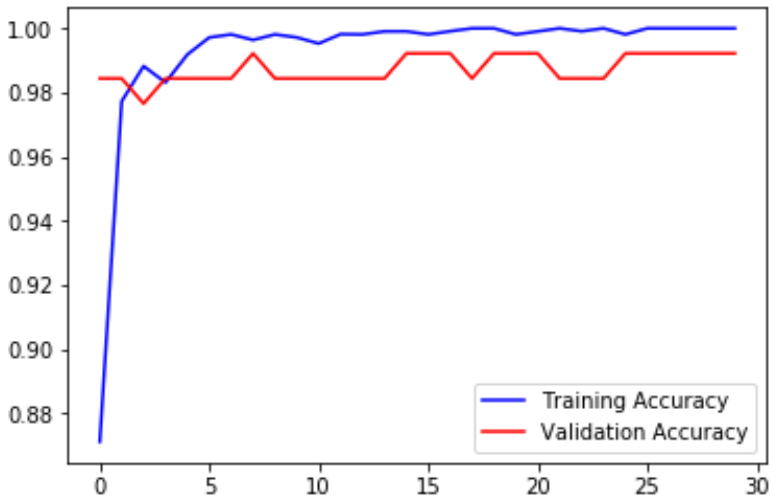

Figura 1.4: Evaluación del modelo (Accuracy)

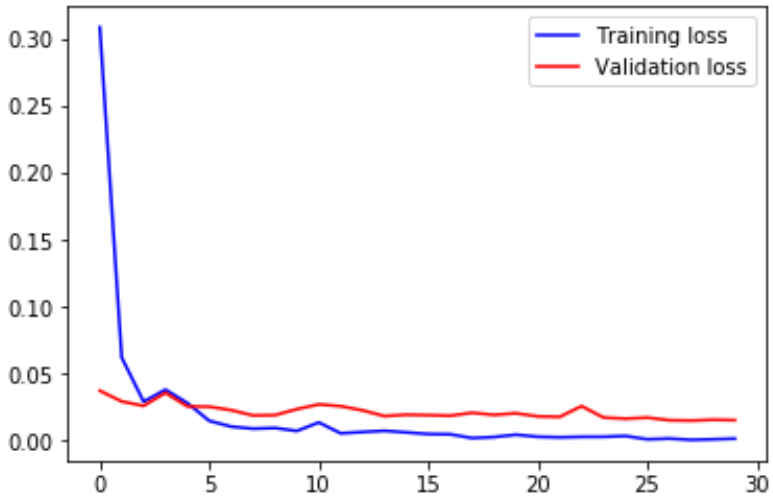

Figura 1.5: Evaluación del modelo (loss)

Por último, en Figura 1.6 se muestra el resumen del final del proceso de entrenamiento.

```
Epoch 25/30
18/18 [==============================] - 4s 207ms/step - loss: 0.0036 - acc: 0.9983 - val_loss: 0.0167 - val_acc: 0.9922
Epoch 26/30
18/18 [==============================] - 4s 204ms/step - loss: 0.0012 - acc: 1.0000 - val_loss: 0.0175 - val_acc: 0.9922
Epoch 27/30
18/18 [==============================] - 4s 209ms/step - loss: 0.0017 - acc: 1.0000 - val_loss: 0.0155 - val_acc: 0.9922
Epoch 28/30
18/18 [==============================] - 4s 209ms/step - loss: 8.2545e-04 - acc: 1.0000 - val_loss: 0.0153 - val_acc: 0.9922
Epoch 29/30
18/18 [==============================] - 4s 209ms/step - loss: 0.0012 - acc: 1.0000 - val_loss: 0.0159 - val_acc: 0.9922
Epoch 30/30
18/18 [==============================] - 4s 212ms/step - loss: 0.0017 - acc: 1.0000 - val_loss: 0.0156 - val_acc: 0.9922
```

Figura 1.6: Resumen final del proceso de entrenamiento

1.4. Otros experimentos

Además del modelo con el que obtuvimos mejores resultados explicado en la sección anterior, también hemos empleado otras arquitecturas con TensorFlow, con la que alcanzamos un ránking de 0.95 aproximadamente. Para implementarlo nos basamos en el tutorial [6]. En este tutorial se explica paso a paso cómo implementar con TensorFlow una CNN desde 0, explicando cada una de sus componentes y parámetros. El primer modelo implementado siguió la misma arquitectura que la del tutorial. La arquitectura de la que hablamos es bastante simple y se puede ver representada gráficamente en Figura 1.7.

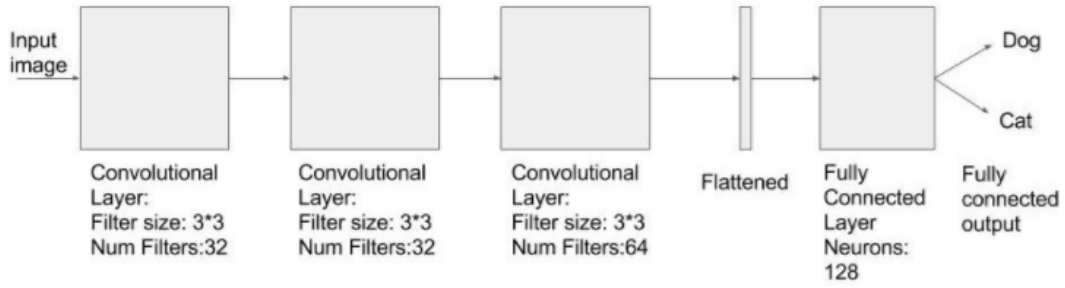

Figura 1.7: Arquitectura de otros experimentos

Para los experimentos se hizo un *Hold-out validation* dejando un 80 % del conjunto de Train para el entrenamiento y un 20 % para validar. El primer experimento obtuvo un Train accuracy de 96.5 % y un Validation accuracy de 95 % apróximadamente ejecutando solo 13 épocas, ejecutando tan pocas porque aun no teníamos instalado CUDA y las ejecuciones iban muy lentas. En el public score se obtuvo un 89.9 % de accuracy.

El segundo experimento teniendo como base este tutorial, se realizó cambiando el tamaño al que redimensionábamos las imágenes. En el tutorial se redimensionaban a 128x128 y lo cambiamos a 256x256. Además ampliamos la ejecución de 13 épocas a 22. Con esta variación otuvimos en el public leaderboard un score de 91.12 %.

Posteriormente se probaron otras variaciones más sobre el modelo. Cambiamos los tamaños de los filtros de las capas convolucionales a 5x5 (siendo previamente de 3x3) y, una vez teníamos instalado CUDA, ya las ejecuciones de las épocas eran

bastante más rápidas. Así, en la siguiente ejecución se obtuvo un 100 % de accuracy en train y en el dataset de validación y en el ranking obtuvimos un 93 % de accuracy.

Por último, basándonos en el tutorial, se probó a cambiar la arquitectura a $Conv1 + Conv2 + Conv3 + Pooling + Conv4 + Conv5 + Pooling + FC + FC$ y se obtuvo un accuracy de en torno al 95 %.

Bibliografía

[1] R. Olmos, S. Tabik y F. Herrera, «Automatic Handgun Detection Alarm in Videos Using Deep Learning», 2017. dirección: https://arxiv.org/pdf/1702.05147.pdf.

[2] F. Carrilo, «Deep Learning para diagnosticar Alzheimer usando Resonancia Magnética», Granada, 2018. dirección: https://www.meetup.com/es-ES/Granada-Geek/events/248019162/.

[3] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition», 2015. arXiv: arXiv:1409.1556v6. dirección: https://arxiv.org/pdf/1409.1556.pdf.

[4] S. Raschka, *In Keras, what is a 'dense' and a 'dropout' layer? - Quora*, 2016. dirección: https://www.quora.com/In-Keras-what-is-a-dense-and-a-dropout-layer (visitado 13-03-2018).

[5] C. Szegedy y S. Ioffe, «Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift», en *International Conference on Machine Learning*, 2015, págs. 448-456. arXiv: arXiv:1502.03167v3. dirección: https://arxiv.org/pdf/1502.03167.pdf.

[6] Ankit Sachan, *Tensorflow Tutorial 2: image classifier using convolutional neural network*. dirección: http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/.