Universidad Zaragoza
1542

Trabajo Fin de Grado

Técnicas de aprendizaje automático para clasificación de imágenes de microscopía

Deep learning techniques for microscopic image classification

Autor

Roger Pou López

Director

Jesús Clemente Gallardo

Facultad de Ciencias - Grado en Física

2020

Introducción

«El cambio es ley de vida. Cualquiera que sólo mire al pasado o al presente, se perderá el futuro.»

J.F.K.

Si bien J.F.K. no fue un científico, su frase resume muy bien la situación de una de las más importantes revoluciones tecnológicas y científicas de los últimos años: Un nuevo subconjunto de la Inteligencia Artificial, el llamado *Deep Learning*, perteneciente a su vez al *Machine Learning*.

Esta área, que mezcla disciplinas tan diversas como matemáticas (álgebra lineal, probabilidad y estadística) y ciéncias de la computación, sigue imparable en su estelar carrera en la imitación (e incluso mejora) de tareas de uno de los sistemas más complejos y fascinantes de nuestro universo: nuestro própio *cerebro*.

Aunque hace unas décadas nadie se le hubiera ocurrido pensar que podríamos haber llegado tan lejos, en la actualidad hay innumerables artículos de premsa y de literatura científica (de casi todos los ámbitos) haciendo referencia a la explotación de excitantes nuevas ideas con estas herramientas. Cantidades ingentes de dinero están siendo invertidas todos los años por las mayores y más punteras empresas tecnológicas del planeta para desarrollar las mejores redes neuronales artificiales o *ANN*.

Parece que una gran mayoría del mundo lo tiene bastante claro: no es una moda pasajera, es *el futuro*. Estas herramientas han llegado para quedarse y mejorar nuestro presente, para facilitar tareas que hace relativamente poco tiempo ni si quiera nos hubiéramos podido plantear automatizar y hasta para clarificar preguntas fundamentales en la ciencia que hasta ahora no habíamos podido responder.

En este trabajo, exploraremos de forma breve las ideas más básicas que forman este campo y más tarde aplicaremos y analizaremos los resultados de todos estos conocimientos en un caso práctico de imágenes de células de microscopía.

Índice

Capítulo 1

Introducción al *Deep Learning*

1.1. ¿Qué es el *Deep Learning?*

El *Machine Learning* (o aprendizaje automático) es una parte de la inteligencia artificial que consiste en el estudio de los algoritmos computacionales que pueden mejorar **automáticamente** a partir de datos empíricos.

Dentro de él, en la última década se ha empezado a utilizar en masa el llamado *Deep Learning* (o aprendizaje profundo), que utiliza las redes neuronales artificiales con la ayuda de distintas áreas de las matemáticas (Probabilidad, Álgebra Lineal, Optimización y de la Mecánica Estadística) y de las ciencias de la computación. Esta área ha tenido un interés exponencial en la última década para aplicaciones de automatización tanto en la industria como en la investigación.

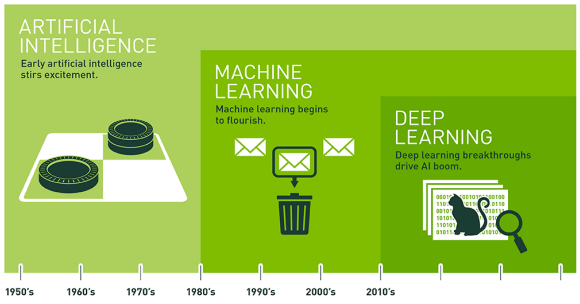


Figura 1.1: Evolución hacia el deep learning. Fuente: Nvidia

En ella, se persigue emular la ejecución de tareas asociadas con procesos que el cerebro humano ejecuta con gran precisión y facilidad, como la identificación de personas en una foto o como la clasificación del 1 al 10 de dígitos escritos en un papel por otra persona [1].

La descripción del problema general consistiría en lo siguiente: el entrenamiento de una red neuronal artificial para que aprenda una tarea en base a unos *inputs* y unos *True targets*

o los valores teóricos a los que la red neuronal aspira aprender. Para ello, cada neurona de la red tiene unos *weights* o pesos, que dan de un valor de la importancia de la activación de una neurona hacia otra a la cual esté conectada y son estos los valores que son modificados a través del proceso de aprendizaje.

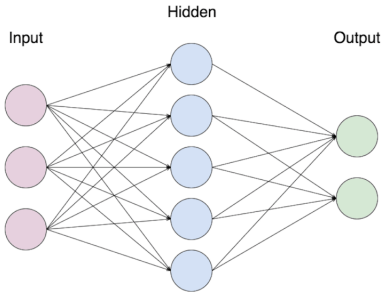


Figura 1.2: Un modelo simple de red neuronal. Fuente: www.dev.to

Estos modelos están basados en el aprendizaje que se produce en el cerebro humano y las interacciones que tienen las neuronas entre sí. Es por eso que se modelizan con modelos de grafos de neuronas "virtuales"que estén conectadas paralelamente a otras capas[1] de neuronas "virtuales"[2]. Para simplificar la modelización, la información fluye por convenio de izquierda a derecha y a veces se le añaden capas intermedias de neuronas que permiten una capacidad de abstracción mayor y así resolver problemas más complejos.

Por ejemplo, el modelo de red neuronal que se usará en este trabajo son las CNN (ver sección 1.4) que están basadas en la estructura de neuronas del Cortex visual del cerebro. En su caso, cada una de las capas es capaz de abstraer más propiedades de las imágenes para las cuales han sido entrenadas. [3]

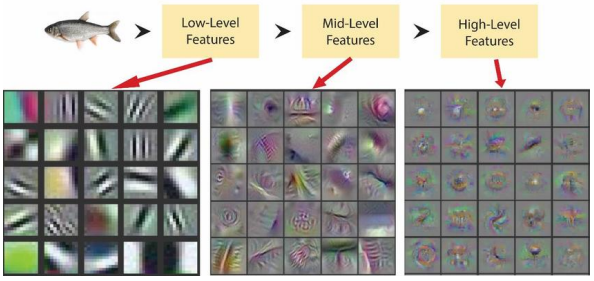


Figura 1.3: Feature representation de una CNN. Fuente: researchgate.net, Shoaib Ahmed Siddiqui

[1] Es importante detallar que la mayoría de modelos actuales de redes neuronales son multicapa
[2] Me refiero a "virtual."[en] el sentido de que no existen físicamente, sino que son una modelización matemática
[3] Usualmente esas abstracciones no son fáciles de entender a simple vista como puede verse en la Figura 1.3

Como hemos comentado, para hace que estas redes neuronales aprendan, se requiere de un entrenamiento previo (*training* en inglés) a través de algoritmos que no veremos en detalle, los llamados *BackPropagation* y el *Stochastisc Gradient Descent* pero que en esencia se reducen a problemas de optimización de una cantidad abrumadora de parámetros.

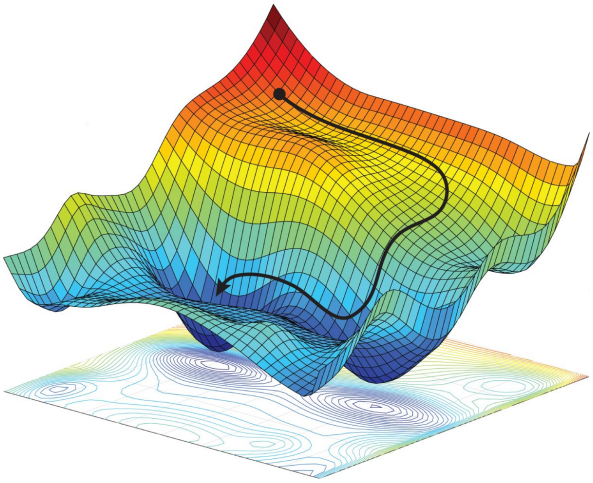


Figura 1.4: Representación gráfica del *Stochastisc Gradient Descent*. Fuente: Navid Azizan, its.caltech.edu

Estos son los procesos que gastan más tiempo computacionalmente y normalmente se requiere que el entrenamiento se realice a través de cómputo en el *cloud* formado por *clusters* de ordenadores con hardware diseñado específicamente para estas tareas para que sean entrenadas en un tiempo razonablemente humano.

Figura 1.5: Las TPU que se utilizan en Google Cloud, diseñadas para tareas como el de entrenamiento de una red neuronal. Fuente: cloud.google.com

1.2. Una posible modelización de las Redes Neuronales

Dentro de las redes neuronales, hay muchas variantes posibles, así que en este apartado daremos una visión general de los componentes más fundamentales. Para ello, utilizamos la misma descripción que se describe en [2]:

– *Input Data* o unos datos de entrada específicos, tal y como se ha mencionado en la sección anterior.

– *Layers* o capas, que están combinadas en nuestra red neuronal. Teóricamente, cada una de esas capas se especializaría en una abstracción más concreta, aunque en la práctica es un poco más complicado de entender que tipos de abstracciones se están realizando en cada una de ellas.

– La *Loss function* o también llamada función de error, que será la que define que tan bueno es el aprendizaje de la red neuronal en base a valores que queramos aprender o también llamados *target*

– Un *Optimizer* o función de optimización, el algoritmo que determina como se procede al aprendizaje que podría ser por ejemplo el *Stochastic Gradient Descent* que hemos comentado anteriormente.

En la siguiente imagen, podemos ver una esquematización general del flujo de información del entrenamiento de una red neuronal:

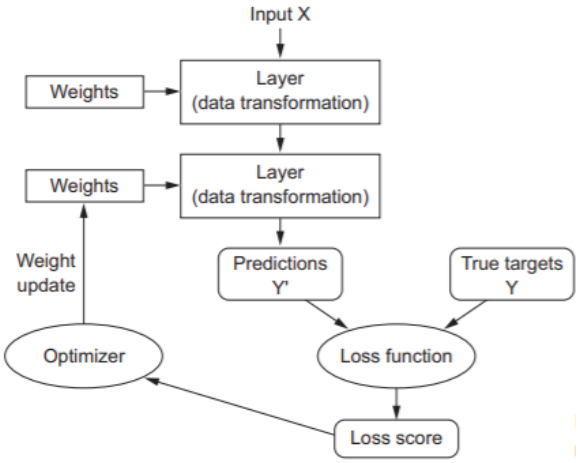


Figura 1.6: Esquema de una red neuronal multicapa. Fuente [2]

1.3. El proceso de entrenamiento de una red neuronal

Una vez escogida el tipo de red neuronal para nuestro problema, esta tiene que ser entrenada con una cantidad de datos suficientemente correcta sobre nuestro problema. Para ello, hay que separar los datos previamente en tres subtipos de contenido: unos datos para únicamente el proceso de entrenamiento y unos datos para realizar un test y/o la validación del aprendizaje. [4] De esta manera, uno se asegura que se esté entrenando adecuadamente la

[4]Normalmente, se sigue una separación de muestras aleatorias con unas proporciones de [80:10:10] o [60:20:20] del total de cantidad de archivos ([train:test:val]).

red neuronal.

1.4. Un breve inciso en las *Convolutional Neural Networks* (CNN)

Como hemos dicho antes, nosotros vamos a tratar con imágenes, que en el ordenador se representan como matrices multidimensionales en formato RGB. Para ello se ha elegido trabajar con ellas de la forma más usual en la práctica, que son las llamadas *Convolutional Neural Networks* o simplemente CNN.

La razón fundamental por las que se utilizan especialmente en este tipo de problemas es que este tipo de redes neuronales, por construcción, se puedan aprovechar de los patrones espaciales de las imágenes que de carácter general tienen invarianza traslacional. Todo esto hace que el entrenamiento para problemas de imágenes con este tipo de redes se reduzca en varios órdenes de magnitud en comparación a lo que haría una red neuronal normal sin ningún tipo de modificación.

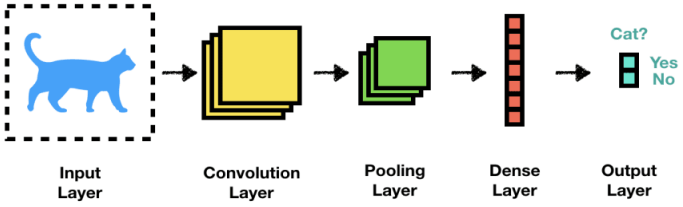


Figura 1.7: Abstracción de la clasificación de objetos en imágenes utilizando CNN. Fuente: www.dev.to

Tal y como indica su nombre, en ellas se utiliza la operación de convolución pero en vez del sentido usual de procesamiento de señales, se utiliza una versión en 2 dimensiones aplicando distintas operaciones con matrices, llamadas *matrices de Kernel*.

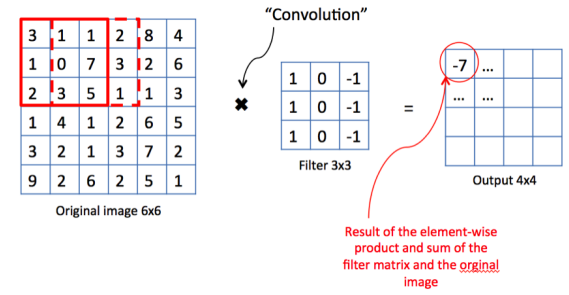


Figura 1.8: Representación de la operación convolución en matrices que representan pixeles de imágenes. Fuente:www.mc.ai, Deep Learning Series

Este tema, que pertenece a la área de estudio llamada *Computer Visión*, es una de las ramas con más atención y investigación en el presente y por poner un mero ejemplo de actualidad, se está utilizando en los nuevos pilotos automáticos de automóviles de la famosa empresa americana *Tesla*.

Capítulo 2

Estimaciones estadísticas en imágenes preclasificadas de microscopio

La implementación propuesta para este trabajo fue la de programación y optimización de una red neuronal que permita una clasificación automática de células tumorales a través de imágenes extraídas a través de microscopio.

Como veremos, esta es una tarea que puede realizar completamente un humano a mano. Sin embargo, la cantidad de imágenes extraídas por un microscopio alcanza un tamaño tal que se podría estimar que el procesamiento y la clasificación de todas esas imágenes podría tardar semanas si se realizara completamente a mano. Es por eso que se motiva la investigación de un proceso que permita automatizar esta tarea con la menor intervención humana.

2.1. Experimento *Cell Spotting*

Todos los datos en el que se basa el trabajo salen de un previo proyecto financiado por la UE realizado entre 2013 y 2016, donde se llevaron a cabo colaboraciones entre la ciudadanía y procesos científicos con el objetivo de educar y motivar a los ciudadanos el uso y entendimiento de la ciencia [3].

En concreto en el proyecto conocido como *Cellspotting*[4] se eligieron a alumnos de secundaria que en sus clases de biología, hicieron uso de una plataforma online (facilitada por la fundación Ibercivis [5]) para una clasificación de células tumorales en imágenes de microscopio.

Esas fotografías correspondían a distintos grupos de células a los que se les había administrado ciertos fármacos antitumorales de manera que sufrían apoptosis. Esto hacía

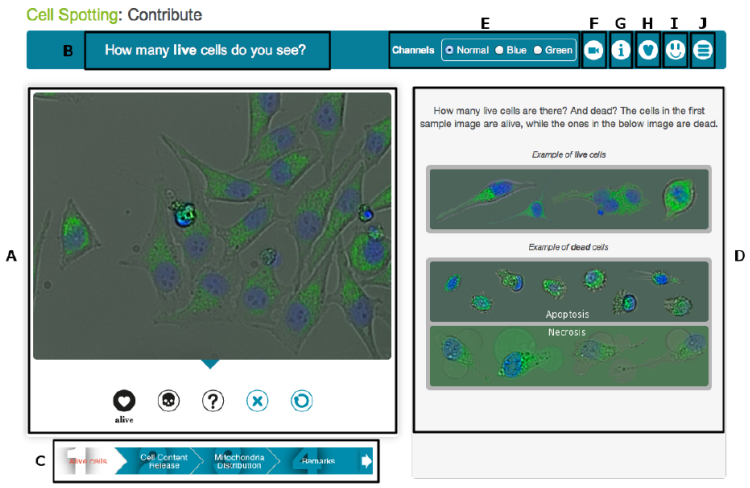


Figura 2.1: Imágenes del proyecto Cell Spotting extraídas de [3]

una fácil identificación a simple vista que podían realizar los alumnos de secundaria sin ninguna complicación.

Los datos generados por el proyecto quedaron almacenados en una base de datos en formato JSON de gran tamaño, la cual, después de un proceso de limpieza de su información más útil, se ha utilizado en este trabajo para la automatización del proceso de clasificación.

Aunque el proyecto inicial no se diseño con este objetivo, posteriormente se planteo ser utilizado para una implementación de *machine y deep learning*. Es por eso que se aprovecha los resultados de estos experimentos para otro proyecto europeo BRITEC[6] el cual, replicará el mismo experimento original y utilizará la plataforma desarrollada por este trabajo de fin de grado en la automatización de las imágenes de microscopio del centro de investigación BIFI[7].

2.2. Extracción de información: limpieza de la base de datos

El fichero de información de gran tamaño, tenia que ser preprocesado antes de poder ser utilizado. En otras palabras, se tuvo que eliminar, organizar y extraer mucha información de manera "manual"[1], puesto que hasta ahora parece que los seres humanos son eficientemente imbatibles respecto a las máquinas en este tipo de tareas.

Esto es lo que se llama un **proceso de limpieza** (*Data Cleaning* en inglés), que en este trabajo fue llevado a cabo a través de la librería de *Python*, *Pandas*[8], librería que sirve para

[1]Manualmente programada en *Python*. Puede verse el código en el anexo.

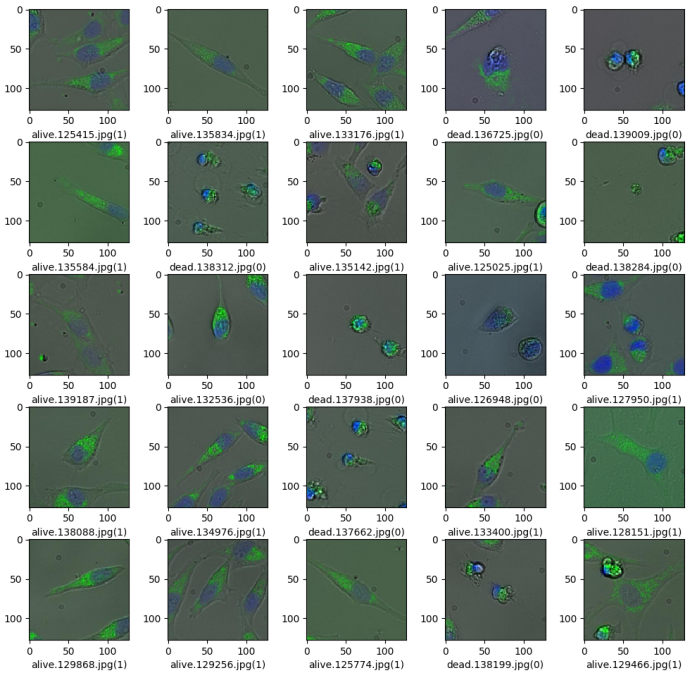


Figura 2.2: Muestra de las fotos, clasificadas posteriormente por los alumnos de secundaria

el manejo de grandes cantidades de datos en tablas de distintos formatos, como el JSON.

Dentro de este proceso de limpieza, se tuvieron que hacer ciertas estimaciones estadísticas que vamos a a ver a continuación.

2.3. Estimaciones estadísticas y creación del *dataset*

En nuestro caso teníamos dos cosas: un *dataset* de imágenes y un JSON que era un conjunto de datos de distintos usuarios sobre si las células del *dataset* de imágenes estaban vivas o muertas y su posición identificada aproximadamente.

Uno puede entender fácilmente que no todas las personas seleccionaron ni el mismo estado vital ni la misma posición para una misma célula, debido a que la naturaleza humana es propia de errores. Así pues, es evidente que los datos del JSON permitían y necesitaban de una solución estadística, haciendo un promediado de las distintas mediciones para cada célula en una misma foto. Es por eso que se realizó una estimación de su estado vital a través de un pesado estadístico correspondiente las distintas mediciones de usuarios.

Teniendo en cuenta todo esto, se ha extraído para cada foto una serie de puntos correspondientes a los distintos puntos compatibles entre si. Para ello, se ha ido calculando la

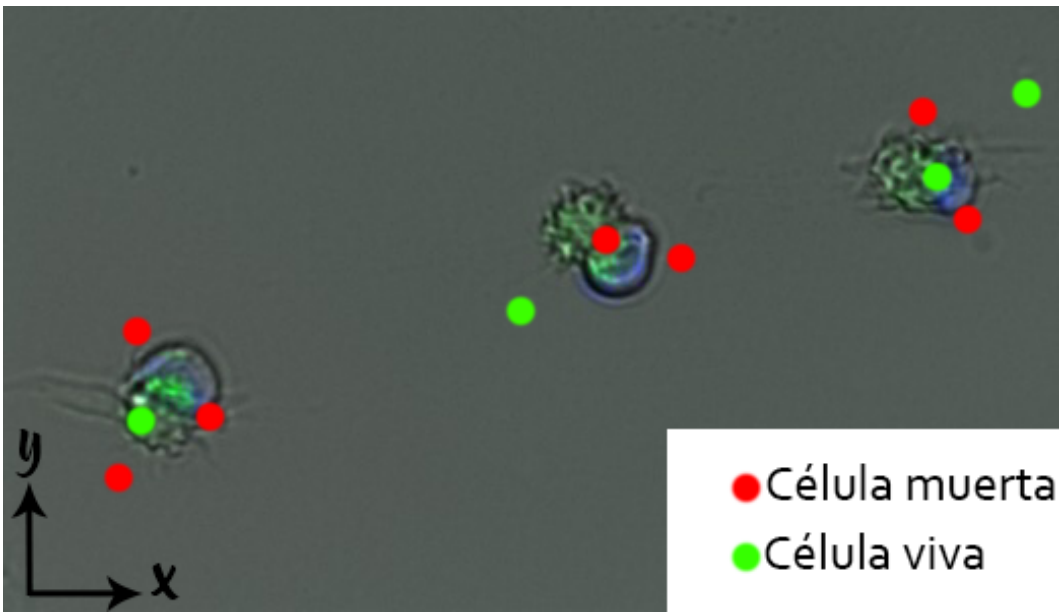


Figura 2.3: Representación gráfica del proceso de análisis llevado a cabo. Cada punto representa una medición de un usuario distinto para una misma foto. El color indica que tipo de estado vital ha detectado cada usuario. Los puntos que no estén cercanos en función del parámetro R, no se consideran que se estén refiriendo a la misma célula

distancia entre puntos en el plano de cada foto y a partir de un parámetro R [2] se ha aceptado o rechazado puntos distintos correspondientes a medidas de otros usuarios. Es decir, si los puntos cumplían que

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} < R = 35 \tag{2.1}$$

entonces se consideraban los puntos como compatibles y se calculaba un punto nuevo como promedio de estos, junto con el promedio de las mediciones vitales y se añadían a la lista de puntos final. Si no se rechazaban [3].

Una vez identificadas las posiciones compatibles para cada foto se guardaron en un nuevo JSON, creando una nueva base de datos filtrada. De esta nueva base de datos es de donde se han recortado del *dataset* original imágenes cuadradas de distintos tamaños (90, 95, 100 y 128 píxeles) generando un nuevo *dataset*. Como hemos comentado en el capítulo 1, se ha hecho un *splitting* del nuevo *dataset* en 3 de distintos: train, test, val siguiendo una proporción [80:10:10] para el entrenamiento de la red neuronal [4].

[2] De acuerdo con la observación directa de los datos

[3] Al haber una cantidad de datos muy grande, se pudo permitir la opción de rechazar cierta información como redundante

[4] Este tipo de proporciones son lo que nos asegura que posteriormente el entrenamiento de la red neuronal sea correcto y prediga con eficacia nuevas fotos a las que no haya visto previamente

Capítulo 3

Una red neuronal como un clasificador binario de imágenes

3.1. Descripción de la red neuronal utilizada

Se ha elegido una arquitectura de la red neuronal tipo VGG-16 [9] simplificada, que supone alrededor de 10 o 15 millones de parámetros dependiendo del tamaño del *input* [1].

Esta red se ha entrenado[2] bajo los nuevos datos filtrados en el JSON de la sección anterior.

Para mejorar este entrenamiento, se han utilizado técnicas de aumento artificial de las imágenes disponibles dentro de la API de *Keras* que junto a técnicas de regularización usuales como el *dropout*, se ha asegurado que la red no este sobre-entrenada para esta serie de datos y que tenga una capacidad de funcionar con imágenes para las que no ha sido entrenada.

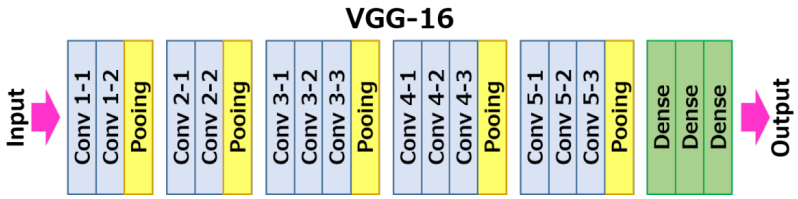


Figura 3.1: La arquitectura de capas de la VGG-16. Fuente: https://neurohive.io

La red se ha diseñado para hacer una clasificación binaria. De esta manera, el resultado final de la red entrenada es un sistema que es capaz de clasificar una foto de una célula por viva o muerta [3], siempre que sea parecida a la de las fotos para las que ha sido entrenada.

[1] La red está implementada en el lenguaje *Python* a través de la API *Keras* dentro de la librería de *Tensorflow 2.2.0*. Se deja una explicación de la elección en el anexo

[2] El entrenamiento de la red se ha hecho en Google Colaboratory, que pone a disposición pública clouds que permiten correr eficientemente modelos de Deep Learning con GPUs.

[3] En el caso de que la célula esté muerta, eso significaría que la célula ha muerto por apoptosis gracias al tratamiento antitumoral inyectado.

Como la red original VGG-16 mencionada en [9] fue diseñada para hacer una clasificación automática de múltiples clases, se ha eliminado último conjunto de 3 capas de convolución de la red ya que estas no mejoraban su precisión y hacía aumentar el tiempo de entrenamiento. También se han ajustado el número de neuronas en cada capa para el problema en cuestión (Ver código en anexo).

3.2. Recortes en base a las estimaciones estadísticas

El recorte basado en las estimaciones estadísticas son más fiables que en la base de datos brutos del JSON sin limpiar. Esto es porque se ha hecho un filtrado de las mediciones que podían ser incorrectas. Aun así, no funciona como hubiésemos esperado en un principio:

- Por una parte se han tenido que rechazar parte de los puntos que, aunque correctos, quedaban en zonas donde no se podía hacer un recorte cuadrado del tamaño deseado.
- Alternativamente, si bien en número total no parecen ser estadísticamente significativas, algunas imágenes recortadas se han observado sin ninguna célula. Esto puede acabar confundiendo a la red neuronal en su aprendizaje de clasificación porque es como si le estuviésemos entrenando de forma incorrecta.

3.3. Estudio del tamaño del recorte en la eficiencia del aprendizaje

Uno de los parámetros que hemos querido optimizar y estudiar es la eficiencia de aprendizaje de la red en función del tamaño de recorte. Este se ha hecho con una descomposición del *dataset* de [80:10:10]. A la hora de diseñar un ajuste óptimo se tiene

Tamaño(píxeles)	90x90	95x95	100x100	128x128
Val accuracy (%)	84.272	87.056	85.160	86.486

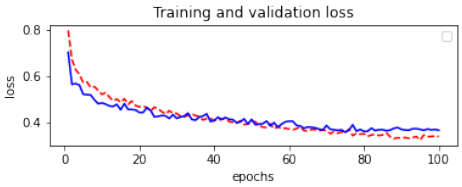
Tabla 3.1: Tabla de valores de la *validation accuracy* para los distintos tamaños de recorte analizados

que tener en cuenta que hay dos fenómenos compitiendo en paralelo:

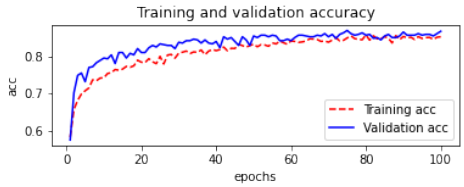
- El número de fotos recortadas para el entrenamiento del aprendizaje es variable dependiendo del radio del recorte. En general, cuantas más fotos correctas se le den a la red neuronal, mejor será el aprendizaje.
- La uniformidad del contenido, es decir, que en una gran mayoría se encuentre una foto con una célula visible en un estado determinado. Se puede observar que reduciendo

el tamaño del recorte, el número de fotos vacías dónde no aparecen células vivas o muertas también lo hace. Esto claramente perjudica el entrenamiento confundiendo a la red neuronal y consiguiendo una menor precisión en la clasificación.

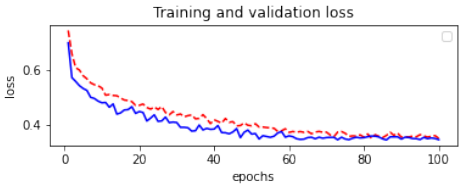
(a) *Accuracy* para 90x90



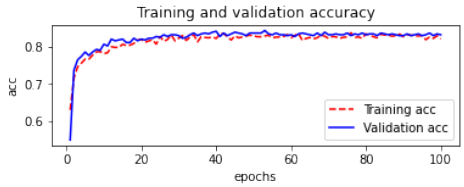
(b) *Loss* para 90x90



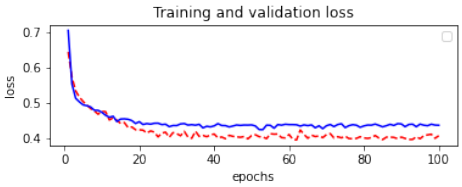
(c) *Accuracy* para 95x95



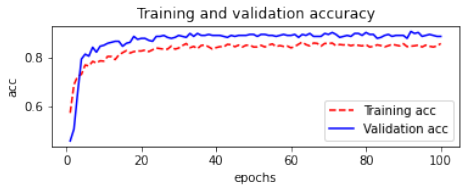
(d) *Loss* para 95x95



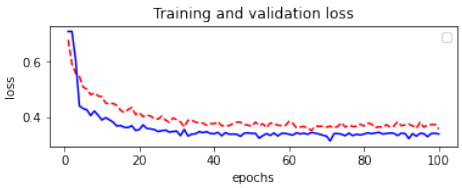
(e) *Accuracy* para 100x100



(f) *Loss* para 100x100



(g) *Accuracy* para 128x128



(h) *Loss* para 128x128

Figura 3.2: Gráficas del entrenamiento de la red neuronal para distintos tamaños de recorte

Presentamos los resultados en la gráfica 3.2 y en la tabla 3.1. Previamente se ha realizado un estudio del *fit* con los parámetros del *Batch Size* y *Epoch* para ver cuales son los que generarían un comportamiento más correcto y suave del que pueda extraerse conclusiones correctas.

En estos datos se puede observar que en el recorte de tamaño 95x95, las gráficas de aprendizaje del *accuracy* y *loss* son las que tienen un mejor *fit*. Aun así, el comportamiento es muy similar con las del tamaño de recorte de 90x90 pero la precisión en el subconjunto de datos de validación del caso de 95x95 es mejor. Esto puede explicarse porque probablemente en este último caso haya un *dataset* con menor número de fotos vacías. Otra observación es la de que si se sube mucho el tamaño del recorte, el entrenamiento de la red neuronal muestra

underfitting o overfitting que se puede deber a la falta de fotos para el entrenamiento. [4]

3.4. Factores que quedarían por estudiar

Estos son algunos de los factores que quedarían para explorar en un nuevo trabajo:

- Entender que efecto tiene el parámetro R entre la distancia de punto aceptables entre células. Puede ser que en vez de ser fijo, sea un parámetro variable que dependa de otras variables como el número de células que clasificadas a su alrededor.
- Entender en precisión el proceso de recorte para conseguir una mayor número de imágenes en los tamaños de recorte mayores. De esta manera, uno podría aumentar la precisión de aprendizaje.

[4] El código de la red neuronal puede encontrarse en el anexo.

Capítulo 4

Conclusiones del trabajo

En este capítulo resumimos los conocimientos adquiridos, los logros y las posibles mejoras que podrían implementarse.

4.1. ¿Qué he aprendido en la producción de este trabajo?

Durante el proceso de producción este Trabajo de Fin de Grado he podido aprender y ganar experiencia en diversos ámbitos:

- No solo he consolidado mis conocimientos de programación en el lenguaje *Python*, sino que he aprendido a utilizar librerías de uso profesional de actualidad como *Pandas, Numpy, Matplotlib* para el pre-procesamiento de grandes estructuras de bases de datos en formato JSON y cálculos científicos y *Tensorflow*, *Jupyter Notebook y Lab*, *Keras* para la producción y entrenamiento de una red neuronal funcional desde zero. (Ver código en el anexo).

- Se ha consolidado y madurado mis conocimientos sobre*Machine y Deep Learning* para la implementación y producción en proyectos reales y en más en concreto, en el área de *Computer Visión* como es el caso de este trabajo.

- Claramente se ha conseguido madurez a la hora de representar resultados en un informe de formato científico.

4.2. Logros conseguidos en este trabajo

Los logros más destacables desde el punto de vista técnico son:

- Se ha conseguido montar y trabajar en una plataforma computacional tanto en un *Cluster* HPC del BIFI como en *Google Colaboratory*. En este último, es donde se han computado los distintos resultados y gráficas del capítulo 3.

– El proceso de limpieza ha sido eficiente (ver código en el anexo) y a partir de él se puede generar recortes de fotos individuales de células etiquetadas con mayor confianza que sin las estimaciones estadísticas.

– Se ha sido capaz de adaptar un proceso de recorte de las imágenes para entrenar un modelo de red neuronal convolucional que ha conseguido casi el 90 % de precisión para la detección de células en fotos de microscopio.

4.3. Posibles mejoras del trabajo que quedarían por hacer

Como todo trabajo, hay posibles mejoras que se quedan limitadas por el tiempo:

– Se podría implementar un algoritmo de *clustering, k-means* [10] para clasificar de forma semiautomática cuales de las mediciones en el plano de la fotografía de las células son compatibles, en vez de definir un parámetro R arbitrario en función de la distancia que puede no ajustarse a la realidad en todos los casos.

– Un estudio general de optimización de los parámetros para ver si se puede encontrar algunos de ellos que permitan a la red neuronal aprender de forma más eficiente y que den a una precisión mayor a la hora de identificar nuevas células. Esto se conoce en la literatura como *Hyperparameter Tuning* y una posible implementación podría hacerse con la librería Keras Tuner [11].

– Un estudio con distintos tipos de arquitecturas como YOLOv5 [12] para conseguir una red neuronal que sea capaz de tener una precisión de clasificación que supere más del 90 % de aciertos. Alternativamente, este tipo de arquitecturas permiten hacer una identificación automática de los objetos de muy alta precisión. Esto seria incluso preferible a una preclasificación humana junto estimaciones estadísticas que pueden estar sujetas a una acumulación de errores y por lo tanto, unos resultados de localización y determinación del estado vital de las células de forma deficiente.

Capítulo 5

Bibliografía

[1] `http://yann.lecun.com/exdb/mnist/`. Mnist dataset.

[2] François Chollet. *Deep Learning with Python*. Manning Publications Co, 2018.

[3] Caroline Manahl Eduardo Lostal Teresa Holocher-Ertl Nazareno Andrade Francisco Brasileiro Paulo Gama Mota Fermín Serrano Sanz José A. Carrodeguas Cândida G. Silva, António Monteiro and Rui M. M. Brito. *Cell Spotting: educational and motivational outcomes of cell biology citizen science project in the classroom*. 2016.

[4] Proyecto Cellspotting. *http://cellspotting.socientize.eu/pybossa/*.

[5] Fundación Ibercivis. *http://pybossa.socientize.eu/pybossa/app/cellspotting/*.

[6] Proyecto europeo BRITEC. *https://britec.igf.edu.pl/*.

[7] Centro de investigación BIFI. *https://www.bifi.es/*.

[8] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[10] El algoritmo de clustering K-means. Página web del proyecto.

[11] `https://www.tensorflow.org/tutorials/keras/keras_tuner`.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

Anexos

Anexos A

Programación en Python

A.1. Python y Keras

Si bien en la actualidad, la cantidad de lenguajes y librerías disponibles para implementar redes neuronales y aprendizaje automático son bastante numerosas debido al creciente interés y uso de estas tecnologías, en este trabajo se ha elegido utilizar el lenguaje *Python* debido a su alta sencillez de escritura, su gran y creciente comunidad de usuarios y la familiaridad que se tiene con el lenguaje de programación *C*.

A día de hoy, parece ser que este lenguaje ya se está convirtiendo en el nuevo estándar para la producción de modelos de *Data Science, Machine y Deep Learning, Computer Vision y Natural Language Processing*.

En nuestro caso, del gran número de librerías disponibles se ha elegido la API de *Keras* dentro de la la librería *Tensorflow* para su producción. Esto es debido a gracias a que tiene una alta abstracción, permite la construcción y entrenamiento de redes neuronales complejas de forma elocuentemente rápida.

A.2. Código del recorte

```
1  import pandas as pd
2  import numpy as np
3  import os
4  import json
5
6
7  os.chdir('/home/roger/Desktop/Cells Deep learning')
8  f = open('export.json')
9  d = json.load(f)
10 l = pd.read_csv('limpieza.csv')
11 l.columns = ['pos','foto']
12 g=[]
13 i=0
```

```
for i in range(l.shape[0]):
    g.append(d[l['pos'][i]]['info']['alive_array'])
l['alive']=g
l.to_csv('limpieza2.csv')
df =pd.read_csv('limpieza2.csv')
names = list(set(df.foto))
names.sort()
nombres = pd.DataFrame(names,columns=['foto_name'])
nombres['foto_name'].to_csv('foto_name.csv')
datos = []
for i in names:
    datos.append(l.groupby('foto').get_group(i)['alive'].reset_index(drop=True))

#%%
np.set_printoptions(suppress=True)

R = 35 #This parameter was chosen by inspection

#function that gives a Boolean in function of the R parameter
def dist(x1,x2,y1,y2):
    return np.sqrt((x1-x2)**2+(y1-y2)**2)<R


#This works in the way that save the different number of data
for every photo in a list to be saved in JSON format
def foto(datos_user):
    aux=0
    if len(datos_user)==1:
        temp=datos_user[0]
        for j in range(len(temp)):
            if temp[j][0]=='alive':
                temp[j][0]=1.0
            elif temp[j][0]=='dead':
                temp[j][0]=0.0
            elif temp[j][0]=='not_sure_alive':
                temp[j][0]=0.5
        return [{'x':i[1],'y':i[2],'alive':i[0]} for i in temp]

    b=[len(datos_user[i]) for i in range(len(datos_user))]
    if max(b)!=min(b):
        return 1
    temp=datos_user[0]

    for j in range(len(temp)):
        if temp[j][0]=='alive':
            temp[j][0]=1.0
            #print(temp[j][0])
        elif temp[j][0]=='dead':
            temp[j][0]=0.0
        elif temp[j][0]=='not_sure_alive':
            temp[j][0]=0.5

    aux=np.array(temp,dtype=np.float)*(1/len(datos_user))
```

```
    #print('aux inicial:', aux)
    for user in datos_user:
        if user == temp:
            continue
        else:
            for i in range(len(temp)):
                for j in range(len(user)):
                    if user[j][0]=='alive':
                        user[j][0]=1.0
                    elif user[j][0]=='dead':
                        user[j][0]=0.0
                    elif user[j][0]=='not_sure_alive':
                        user[j][0]=0.5
                    if dist(temp[i][1],user[j][1],temp[i][2],user[j][2])==True:
                   # if dist(temp[i][1],user[j][1],temp[i][2],user[j][2])==True:
                        #print('SI se acepta i:',i,'j:',j)

                        user_float=np.array(user[j])*(1/len(datos_user))
                        #print('user_float:',user_float,'aux[i] antes',aux[i])
                        aux[i]+=user_float
                        #print('aux[i] after:',aux[i],'\n')
                    #else:
                        #print('no se acepta i:',i,'j:',j,'\n')


    #print('aux final:',aux)
    return [{'x':i[1],'y':i[2],'alive':i[0]} for i in aux]









def info_datos(names,datos):
    lista=[]
    for i in range(len(datos)):
        if type(foto(datos[i]))!=int:
            lista.append({names[i]: foto(datos[i])})
        else:
            lista.append({names[i]: 'empty'})
    print(lista)
    return json.dumps(lista)


with open('exportlimpio.json','w') as f:
    f.write(info_datos(names,datos))
```

A.3. Código para el *splitting* del dataset

```
import os
from os import makedirs
from os import listdir
from shutil import copyfile
import split_folders

#%%
def split_dataset():# create directories
    dataset_home = 'cropdataset/'
    labeldirs = ['dead/', 'alive/']
    for labldir in labeldirs:
        newdir = dataset_home + labldir
        makedirs(newdir, exist_ok=True)

    # copy training dataset images into subdirectories
    src_directory = 'dataset_rad=95/'
    i=0
    j=0
    NUM_OF_FILES=1000

    for file in listdir(src_directory):
        if i<NUM_OF_FILES and j<NUM_OF_FILES:
            src = src_directory + '/' + file

            if file.startswith('alive'):
                dst = dataset_home + 'alive/'  + file
                copyfile(src, dst)
                i+=1
            elif file.startswith('dead'):
                dst = dataset_home + 'dead/'  + file
                copyfile(src, dst)
                j+=1

#Size of the photo crop
rad=95
input_dir="dataset_radio="+str(rad)+"/"

split_folders.ratio(input_dir, output="output", seed=1337, ratio=(.8, .1, .1)) # default values
train_dead_dir='output/train/dead'
train_alive_dir= 'output/train/alive'
validation_dead_dir='output/val/dead'
validation_alive_dir='output/val/alive'
test_dead_dir='output/test/dead'
test_alive_dir='output/test/alive'

print('total training dead images :', len(os.listdir( train_dead_dir ) ) )
print('total training alive images :', len(os.listdir( train_alive_dir ) ) )

print('total validation dead images :', len(os.listdir( validation_dead_dir ) ) )
print('total validation alive images :', len(os.listdir( validation_alive_dir ) ) )
```

```
print('total test dead images :', len(os.listdir( test_dead_dir ) ) )
print('total test alive images :', len(os.listdir( test_alive_dir ) ) )
```

A.4. Código de la Red Neuronal

```
%load_ext tensorboard
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import random
import os
import datetime
import sys
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import  Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation,
BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoard
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam



input_shape=(95,95,3)
batch_size = 10
epochs = 100



def build_model():
    model = Sequential([

        Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same',
                  input_shape=input_shape),
        BatchNormalization(),
        Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2,2)),

        Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2), strides=(2,2)),

        Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'),
```

```
46          BatchNormalization(),
47          Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'),
48          BatchNormalization(),
49          MaxPooling2D(pool_size=(2, 2), strides=(2,2)),
50
51          Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'),
52          BatchNormalization(),
53          Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'),
54          BatchNormalization(),
55          Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'),
56          BatchNormalization(),
57          MaxPooling2D(pool_size=(2, 2), strides=(2,2)),
58
59
60          GlobalAveragePooling2D(),
61
62          Flatten(),
63
64          Dense(1024, activation='relu'),
65          Dropout(0.1),#0.3 for 90%
66
67          Dense(512, activation='relu'),
68          Dropout(0.1),
69
70          Dense(1, activation='sigmoid')
71
72      ])
73      opt = Adam(lr=1e-6)
74      model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
75      model.summary()
76      return model
77
78  def training():
79      # define model
80      model = build_model()
81      # create data generator
82      train_datagen = ImageDataGenerator(
83          rotation_range=15,
84          rescale=1.0/255.,
85          shear_range=0.1,
86          zoom_range=0.2,
87          horizontal_flip=True,
88          width_shift_range=0.1,
89          height_shift_range=0.1)
90
91      validation_datagen = ImageDataGenerator( rescale = 1.0/255.)
92
93      test_datagen  = ImageDataGenerator( rescale = 1.0/255. )
94
95      # prepare iterators
96      train_generator = train_datagen.flow_from_directory('output/train/',
97                                                          class_mode='binary',
98                                                          batch_size=batch_size,
```

```
                                                target_size=input_shape[:2])
test_generator = test_datagen.flow_from_directory('output/test/',
                                                  class_mode='binary',
                                                  batch_size=batch_size,
                                                  target_size=input_shape[:2])
validation_generator = validation_datagen.flow_from_directory('output/val/',
                                                              class_mode='binary',
                                                              batch_size=batch_size,
                                                              target_size=input_shape[:2])
steps_per_epoch = train_generator.n // batch_size
validation_steps = validation_generator.n // batch_size



#preparing callbacks

earlystop = EarlyStopping(
                          monitor='val_accuracy',
                          min_delta=0.05,
                          patience=70,
                          verbose=1,
                          mode='auto'
                          )

learning_rate_reduction = ReduceLROnPlateau(
                                            monitor='val_loss',
                                            patience=4,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=1e-9,
                                            mode='auto'
                                           )
checkpoint_filepath = 'weights/'+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True,
    verbose=1
    )

logdir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

tensorboard = TensorBoard(log_dir=logdir,
                          histogram_freq=1,
                          write_graph=True,
                          write_images=True
)
callbacks_list = [earlystop, learning_rate_reduction,model_checkpoint_callback,tensorboard]

%tensorboard --logdir=/content/logs/
```

```
    # fit model
    history = model.fit(train_generator,
                        steps_per_epoch=steps_per_epoch,
                        validation_data=validation_generator,
                        validation_steps=validation_steps,
                        epochs=epochs,
                        verbose=1,
                        callbacks=callbacks_list
                       )
    #load best model
    model.load_weights(checkpoint_filepath)
    # evaluate model
    _, acc = model.evaluate(test_generator, steps=len(test_generator), verbose=1)
    print('The accuracy for the test_generator is: > %.3f' % (acc * 100.0))

    # learning curves

    summarize_diagnostics(history)


def summarize_diagnostics(history):

    acc      = history.history['accuracy']
    val_acc  = history.history['val_accuracy']
    loss     = history.history['loss']
    val_loss = history.history['val_loss']

    epochs   = range(1,len(acc)+1,1)
    plt.subplot(211)
    plt.title('Classification Accuracy')
    plt.plot ( epochs,     acc, 'r--', label='Training acc'  )
    plt.plot ( epochs, val_acc,  'b', label='Validation acc')
    plt.title ('Training and validation accuracy')
    plt.ylabel('acc')
    plt.xlabel('epochs')

    plt.legend()
    plt.figure()

    plt.subplot(212)
    plt.title('Cross Entropy Loss')
    plt.plot ( epochs,     loss, 'r--' )
    plt.plot ( epochs, val_loss ,  'b' )
    plt.title ('Training and validation loss'   )
    plt.ylabel('loss')
    plt.xlabel('epochs')

    plt.legend()
    plt.figure()
    #plt.savefig('training.png')
    plt.show()

if __name__=='__main__':
```

```
205     training()
```