UNIVERSIDAD DE GRANADA

UNIVERSIDAD DE GRANADA E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN - Course 2019/ 2019

Técnicas de Soft Computing para Aprendizaje y optimización. Redes Neuronales y Metaheurísticas, programación evolutiva y bioinspirada

PRÁCTICA FINAL

Histopathologic Cancer Detection

Jaime cloquell Capp

Email:
jaumecloquell@correo.ugr.es

Contents

1 Introducción

El Aprendizaje Profundo es un subcampo dentro del Aprendizaje de Máquina que utiliza diferentes algoritmos de aprendizaje automático para modelar abstracciones de alto nivel en datos usando arquitecturas jerárquicas, conocidas como redes neuronales profundas (DNNs). Entre los múltiples algoritmos que se pueden encontrar, existen algunos como las redes neuronales convolucionales (CNNs), los autocodificadores y las redes recurrentes (RNNs), que pueden ser de gran ayuda a la hora de analizar imágenes médicas.

En este documento se recoge una introducción a las Redes de Neuronas Artificiales. Desde un enfoque teórico se presentan los fundamentos de las Redes Neuronales clásicas y las arquitecturas de las Redes Neuronales empleadas hoy en día. Asimismo, se muestra cómo implementar los diseños descritos por la teoría mediante el lenguaje de programación Python y la librería TensorFlow para resolver para resolver y experimentar con un problema de la plataforma web Kaggle.

2 Objetivos y justificación

El principal objetivo del trabajo propuesto es en primer lugar un acercamiento a las redes convolucionales, entender su funcionamiento y desarrollar un modelo que sea capaz de reconocer, lo más fielmente posible la detección de cancer histopatológico, a partir de imágenes etiquetadas.

3 El aprendizaje automático y las Redes Neuronales Artificiales

El aprendizaje automático es una rama de la inteligencia artificial y puede definirse como un área de estudio que proporciona a las computadoras la capacidad de aprender, sin éstas estar explícitamente programadas. De esta forma, mediante los variados algoritmos de aprendizaje automático es posible generar un resultado, generalmente un conjunto de predicciones o decisiones, a partir de diversos datos de entrada sin la necesidad de agregar nuevas líneas de código al sistema inicial.

El aprendizaje supervisado tiene un conjunto de herramientas enfocadas a resolver problemas dentro de su dominio, siendo una de ellas, precisamente, las Redes Neuronales Artificiales. Estas redes son un modelo computacional vagamente inspirado en las redes neuronales biológicas que constituyen los cerebros humanos y cuya analogía con el modelo matemático se puede observar en la Figura 6.

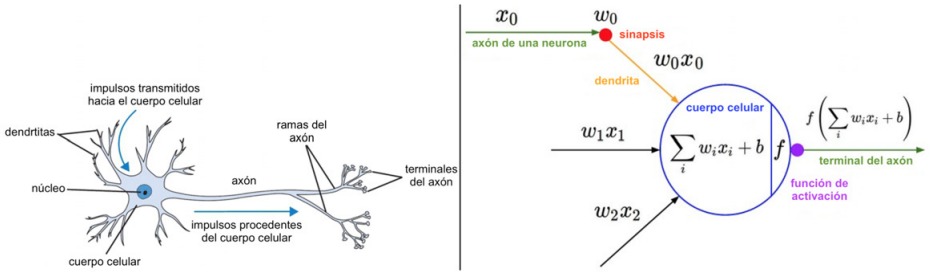


Figure 1: Imagen de una neurona biológica (izquierda) y el modelo matemático que intenta reproducir superficialmente su funcionamiento (derecha)

Estas neuronas son organizadas por capas y a su vez están conectadas con unos pesos que simbolizan las sinapsis de un cerebro real, de la forma que los pesos ajustan la fuerza de cada conexión.

3.1 Training Data

Posiblemente uno de los pasos más importantes a la hora de desarrollar un modelo de inteligencia artificial, sea crear un training data adecuado, puesto que para que este cumpla con los objetivos propuestos se tendrá que sortear diferentes problemas que pueden hacer que el modelo no funcione correctamente. Uno de los principales problemas a lo que hay que enfrentarse es no disponer de datos suficientes para entrenar nuestro modelo, este problema tiene difícilmente solución, sin embargo, en casos muy concretos es posible generar artificialmente datos propios haciendo duplicados con una ligera variación. Por otro lado, otro problema importante es que los datos que disponibles no sean representativos para el modelo que se desea crear, en consecuencia, las predicciones podrían perder precisión.
Finalmente, cuando se dispongan ya de los datos necesarios para entrenar el modelo, pueden surgir dos complicaciones principales durante el entrenamiento:

- Overfitting: Es un problema muy común que surge cuando el modelo es entrenado de forma redundante y es incapaz finalmente de generalizar correctamente. No necesariamente puede ser debido a que los datos sean excesivamente repetitivos o el training data sea demasiado largo, sino incluso al tipo de arquitectura que se haya creado, una red neuronal con excesivas neuronas puede llegar a causar también este efecto. Una forma muy efectiva para evitar que suceda este problema, es aplicar técnicas de regularización como Lasso, Ridege, etc que permitan mantener nuestro modelo lo más simple posible.

- UnderFitting: Tal y como se puede intuir, se trata del problema opuesto, ocurre cuando el modelo es demasiado simple y no dispone del tamaño suficiente para inferir el patrón ya sea porque la arquitectura de la red, modelo o tamaño del training data es insuficiente.

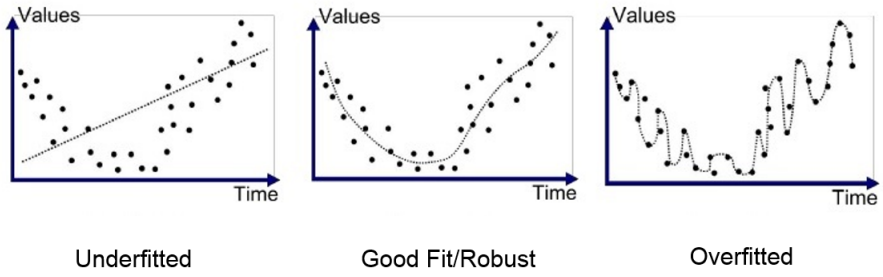


Figure 2: OverFitting vs UnderFitting

4 Redes Neuronales Convolucionales

En las CNN se aplica una filosofía similar a las FNN, este tipo de redes encuentran su inspiración en el cerebro, concretamente en el cortex visual, donde este tipo de neuronas se activan si el patrón de entrada responde al patrón almacenado en ellas, pueden albergar patrones tanto simples como complejos; líneas, bordes, etc. Gracias al experimento realizado por Hubel y Wiesel en 1962 pudo comprobarse este proceso de activación de neuronas a determinados estímulos. Estas capas se van organizando en complejidad, es decir van construyendo e interpretando el estímulo capa por capa.

Las redes neuronales convolucionales son similares a las redes neuronales multicanal, su principal ventaja es que cada parte de la red se le entrena para realizar una tarea, esto reduce significativamente el número de capas ocultas, por lo que el entrenamiento es más rápido. Además, presenta invarianza a la traslación de los patrones a identificar.

Las redes neuronales convolucionales son muy potentes para todo lo que tiene que ver con el análisis de imágenes, debido a que son capaces de detectar características simples como por ejemplo detención de bordes, lineas, etc y componer en características más complejas hasta detectar lo que se busca.

La arquitectura basica de una CNN se divide en cuatro operaciones principales:

- Convolución.
- Capa de activación.
- Clasificación
- Convolución.

Estas 4 operaciones son los componentes basicos de una CNN. La operacion principal de las CNN como su nombre indica es la convolucion, que extrae las características principales dada una imagen de entrada

4.1 Convolucion

Las CNN no utilizan un filtro de convolución codificado a mano como kernel si no que una CNN los parámetros de cada nucleo de convolución son entrenados por el algoritmo backpropagation. Cada capa puede tener diferentes núcleos que estos se utilizan a lo largo de la imagen con los mismos parámetros. La función de esta consiste en extraer diferentes características de la imagen de entrada. Las primeras capas de la convolucion suelen obtener características y detalles de bajo de nivel de una imagen como lineas y bordes en cambio cuantas mas capas tenga nuestra red mas caracteríticas de alto nivel recibirá.

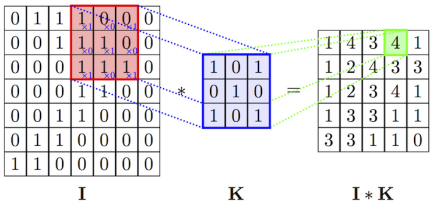


Figure 3: Capa Convolucional

4.2 Capa de activación

Para la funcion de activación en las CNN se utiliza principalmente la signoide ReLU que selecciona una tasa de aprendizaje adecuada y la fraccion de neuronas muertas. También se puede probar con Leaky Relu y Maxout. La función que nunca se utiliza en las CNN son las sigmoide logística.

4.3 Pooling

En esta capa se busca producir una reducción del muestreo para reducir así la resolución y por ende el exceso de información reduciendo la carga computacional de la red y la memoria utilizada, mediante una función de agregado como por ejemplo una media, suma o el máximo. Como podemos observar en la figura 4 de una imagen 2x2 que contiene información importe destacada con un número elevado e información menos importante con un numero mas cerca del 0 cuando se le aplica max pooling guarda de cada cuadrante de la imagen el máximo de información de la misma ayudando al entrenamiento de la red y reducir el ruido de las imagenes para una mejor clasificación.

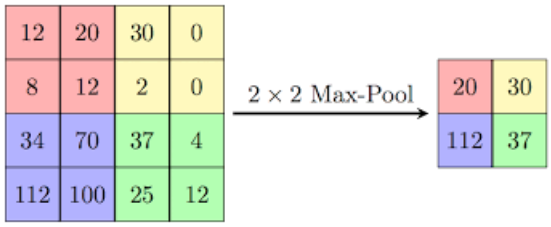


Figure 4: Resultado de aplicar Max Pool

La gran ventaja que tiene utilizar “max-pooling”, es que la red puede preguntarse si una característica está en alguna región de la imagen, por ende, no es necesario saber la localización exacta sino más

bien como está relacionada con las otras características, por lo tanto permite reducir así el número de parámetros necesarios en las próximas capas ya que hay menos características agrupadas.

4.4 Neuronas de Clasificación "Fully Connected Layer"

Después de pasar por las diferentes capas de extracción de características, los datos finales han de clasificarse hacia una categoría u otra según los objetivos. Las neuronas de estas capas son idénticas a otras redes como las FNN, pueden utilizar diferentes algoritmos de aprendizaje backpropagation, Adam, RMSProp, etc.
Por último, la última capa es un conjunto de neuronas simples que sirven para hacer la clasificación final de las características inferidas en el proceso mediante una función sigmoid en el caso de que sea una clasificación binaria o softmax en el caso de la existencia de múltiples categorías.

4.5 Algoritmo de aprendizaje

El algoritmo normalmente empleado para el aprendizaje es el mismo que en otras FNN, se trata del "backpropagation" empleando el gradiente descendiente, siendo utilizado tanto en las capas convolucionales como en las fully connected. La estructura final, podría ser algo similar a la Figura 5 donde se puede observar la estructura general de una CNN, con sus múltiples capas convolucionales, pooling y una última capa de clasificación. En el trabajo, haremos uso de esta estructura básica para que a partir de ella adaptarla a los datos.

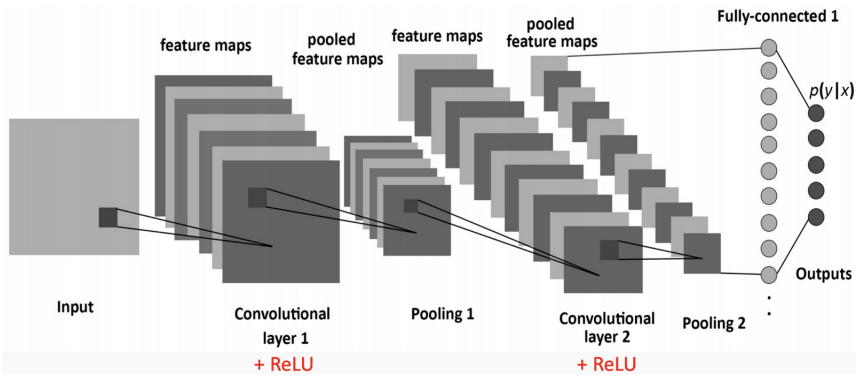


Figure 5: Convolutional Neural Network Layers [

5 Marcos de Implementación

Actualmente hay una gran cantidad de entornos de trabajo desarrollados exclusivamente para el aprendizaje profundo. Algunos de los más populares son TensorFlow, Theano, Caffe, Keras o PyTorch. En el caso particular de este proyecto y tal como se expondrá a lo largo de este capítulo son usados básicamente dos: Keras y Tensorflow, dado que son los más versátiles al ofrecer la mayor cantidad de herramientas que simplifican notablemente las implementaciones y los que permiten una integración más directa con otras plataformas como Kaggle.

5.1 Kaggle

Kaggle es una plataforma online para realizar competiciones de Data Mining, proporciona un repositorio para que las compañías publiquen sus datos y desde ahí comienza un concurso abierto para que los expertos en Data Mining de todo el mundo descarguen esos datos y propongan soluciones a los problemas de la compañía en cuestión. La mejor solución se hace con un premio que puede llegar a varios millones de dólares. EL uso de esta plataforma es debido a que ofrece gratuitamente on servidor donde llevar a cabo las pruebas realizadas.

5.2 Keras y TensorFlow

Python dispone de multiples librerías como podrían ser Keras o TensorFlow, para el desarrollo de redes neuronales o algoritmos de machine learning. En concreto, en este apartado se utilizarán ambas para la construcción de la CNN propuestay observaremos el desempeño de ambas APIs.
El módulo de capas de TensorFlow nos ofrece una API desde donde la podemos construir de forma rápida y sencilla cualquier tipo de red. Sin embargo, han diseñado una nueva High-API que funciona por encima de Tensorflow, aún más sencilla y será la que utilizaremos, llamada keras desde donde poder crear la red mediante unos métodos más simplificados.
Los métodos empleados fueron los siguientes:

- Conv2D() : Construye una capa convolucional de 2 dimanesiones, se ha de especificar el número de filtros, su tamaño (ej 5x5) , el padding y su función de activación.
- MaxPoolingLayer(): indicamos el tamaño del pool o de ventana, que permitirá la reducción de información, tal y como hemos explicado anteriormente.
- sequential(): se indica a Keras que se va a crear un secuenciado de capas
- dropout(): se indica la probabilidad de que una neurona sea desechada, se usa para evitar los minimos locales y mejorar la optimización.
- Dense(): capa ForwardPass.

5.3 Base de datos

El conjunto de datos es un subconjunto del conjunto de datos del PCam y la única diferencia entre ambos es que todas las imágenes duplicadas han sido eliminadas. El conjunto de datos de PCam se deriva del conjunto de datos de Camelyon16 Challenge que contiene 400 imágenes de diapositivas enteras teñidas de HE de secciones de ganglios linfáticos centinela que fueron adquiridas y digitalizadas en 2 centros diferentes utilizando un objetivo 40x.
El dataset contiene 220k imágenes de entrenamiento y 57k imágenes de evaluación.

6 EDA

6.1 Caracterísiticas de los datos

De acuerdo con la descripción de los datos, hay un equilibrio de 50/50 entre los ejemplos positivos y negativos en las divisiones de entrenamiento y de pruebas. Sin embargo, como podemos ver en la

siguiente tabla 1 la distribución de la capacitación parece ser 60/40 (negativos/positivos). Una etiqueta positiva significa que hay al menos un píxel de tejido tumoral en la región central (32 x 32px) de la imagen.

Descripción del dataset	
Fromato	TIF
Tamaño	96 X 96
Negativo / Positivo	130908 / 89117

Table 1: Descripción del dataset

Como el dataset es considerablemente grande y además existen mecanismo de generación de imágenes aleatorias a partir de las imágenes reales se decidió equilibar el porcentaje de ejemplos de cada clase cojiendo un total de 80000 ejemplos de cada clase.

6.2 Visualización de los datos

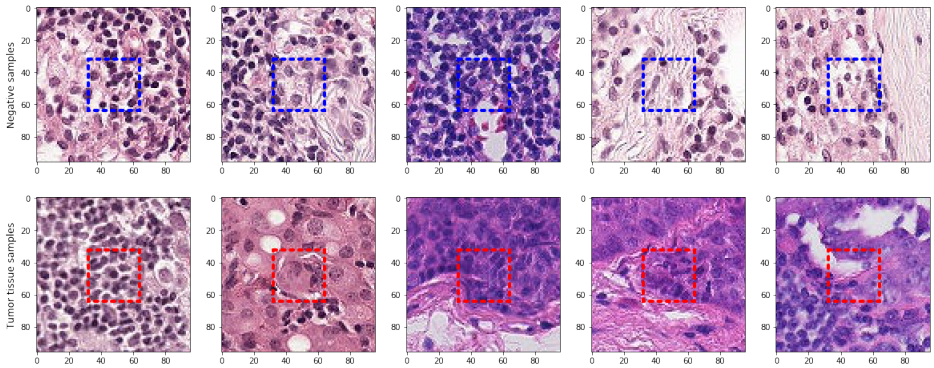


Figure 6: Imágenes aleatorias obtenidas del dataset

7 Datos de entrenamiento y validación

Se han divido los datos del dataset en un 80% para el entrenamiento y un 20% para la validación.

Los datos de validación (el 20% de observaciones del dataset) servirán para verificar si al acabar una iteración (epoch) del entrenamiento el modelo se está entrenando de forma correcta. En caso contrario, se pueden detectar a tiempo sobreajustes del modelo (overfitting) al conjunto de entrenamiento, evitando de esta manera fallos de predicción para el resto de observaciones del dataset.

Finalmente, el conjunto de test se reserva para cuando el modelo predictivo haya sido entrenado por completo. Como los datos de test no han influido directamente sobre el proceso de entrenamiento, las predicciones realizadas al aplicar estos datos sobre el modelo entrenado nos dan una estimación real de la calidad del entrenamiento.

Listing 1: División de los datos de entrenamiento en test y validación

```
df['label'] = df['label'].astype(str)
train, valid = train_test_split(df, test_size=0.1, stratify = df['label'])
```

8 Optimizaciones

Se presentan dentro de este capítulo cuatro conceptos adicionales que han sido de enorme utilidad en su aplicación al desarrollo del modelo final de la red neuronal. Son conceptos relativamente innovadores cuyo principal objetivo es conseguir reducir el sobreajuste u overfitting, además de aumentar la precisión de la red y disminuir los tiempos de entrenamiento de la misma.

8.1 Callbacks

Los callbacks son funciones que nos permiten visualizar cómo se está desarrollando el entrenamiento de nuestro modelo y nos permite crear mejores en base a la información que nos dan. En esta práctica hemos usado tres callbacks que son los que se especifican a continuación.

- ModelCheckpoint: Este callback nos permite guardar los pesos de la mejor solución encontrada hasta el momento, lo cual nos permite recuperarlos cuando vayamos a predecir.
- EarlyStopping: Permite que el entrenamiento de la red se pare si no hay mejoras en la métrica que se use (hemos usado tanto la pérdida como la tasa de acierto en el conjunto de entrenamiento) en un determinado número de iteraciones.
- ReduceLROnPlateau: Permite reducir la tasa de aprendizaje al moverse en mesetas de la métrica que estemos monitorizando.

8.2 Dropout

Esta técnica consiste en ignorar o apagar neuronas de la red neuronal, a lo largo del proceso de entrenamiento,con una cierta probabilidad denominada tasa de dropout (dropout rate). Al ir eliminando neuronas de forma aleatoria conseguimos ir probando nuevas topologías de redes neuronales en el entrenamiento, haciendo que nuestra red sea mas robusta frente a pequeñas variaciones en los valores de la capa de entrada. En la figura 7 se muestra el resultado de aplicar dropout a una red.

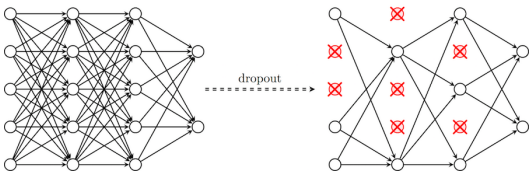


Figure 7: Dropout

8.3 Batch Normalization

Otra técnica surgida recientemente es batch normalization, que permite corregir durante el entrenamiento el problema de la alta variabilidad en la distribución de los inputs para cada neurona de la red. Las ventajas de esta técnica son mejores resultados en menos epochs de entrenamiento (debido a que los creadores de esta técnica probaron que se podían usar tasas de aprendizaje muy altas), mejoras en las predicciones realizadas por las redes y eliminación del problema de sobreajuste (es decir, batch normalization actúa también como regularizador). Además, si en la capa de entrada aplicamos batch normalization, estaremos aprovechando para realizar una normalización de los datos de entrada, en el caso de que no lo estuvieran previamente.
En la figura 8 se muestra el resultado de aplicar Batch Normalization a una red neuronal:

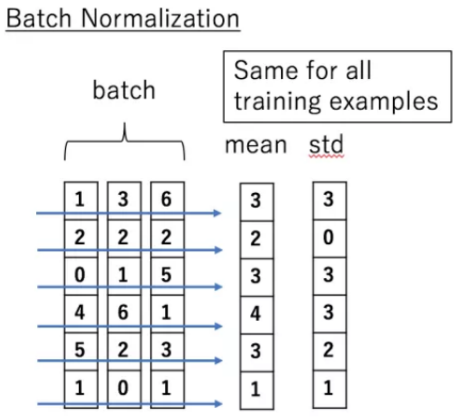


Figure 8: Batch normalization

8.4 ImageDataGenerator

Práctica común dentro de la clasificación de imágenes mediante redes de neuronas es el aumento de los datos presentes en el dataset.

La eficacia de esta técnica reside en la forma en que las redes neuronales entienden las imágenes y sus caracteísticas. Si una imagen es modificada ligeramente es percibida por la red como una imagen completamente distinta perteneciente a la misma clase. Esto disminuye las probabilidades de que la red se centre en orientaciones o posiciones mientras mantiene la relevancia de las características deseadas.

Se ha realizado un aumento del conjunto de entrenamiento siguiendo el siguiente criterio:

- Rescale: 1./255
- Vertical flip: True

- Horizontalflip: True

En la figura 9 podemos apreciar un ejemplo de las transformaciones que se llevaran a cabo durante el proceso de entrenamiento de la red. Es importante destacar que no se han llevado más transformaciones ya que por el tipo de imagen que estamos tratando, padecían distorsiones que causaban peores resultados. Además como nuestra base de datos es extensa, no se veía la necesidad de aumentarla aún más.

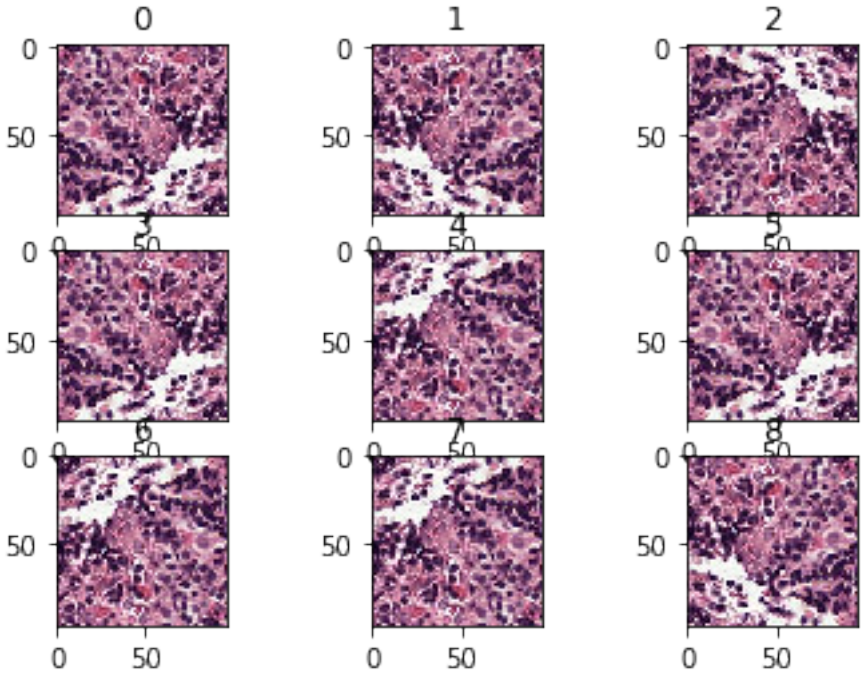


Figure 9: Ejemplo transformacion para aumentar el dataset

9 Preprocesamiento de los datos

Siempre se recomienda normalizar todos los datos restando cada dato por lamedia y dividiendo por su desviacioó típica, ya que normalmente aumenta el rendimiento de la red neuronal y facilita los cálculos. Sin embargo, en este caso en particular, esta normalización suponía una clara desventaja ya que los resultados eran peores. Se probo con dividir todos los datos por un valor constante de 255, que es el que se suele usar en tratamiento de imágenes para limitar las intensidades de los pixeles entre 0 y 1, obteniendo mejores resultados que con la normalización estándar y sin normalización.

10 Métricas de evaluación

En la clasificacion de imágenes la salida de la red etiqueta a esta imagen con una etiqueta donde predice con un porcentaje el acierto de la misma. En este trabajo se ha utilizada la métrica accuracy donde

se evalúa el porcentaje de aciertos sobre el total de muestras.

Además hemos utilizado la curva ROC que es una herramientas que se utilizada también en la evaluación del rendimiento de clasificadores binarios. Esta curva nos indican de manera visual la relación entre la precisión y la sensibilidad de nuestro modelo.

11 Modelos propuestos

El primer escenario se basaba en desarrollar un modelo que permita detectar si una imagens tiene cancer o no a partir de una base de la datos.
El modelo que se ha propuesto es el de la imagen 10, 2 redes convoluciones seguidas cada una por una red Pooling para reducir las dimensiones, una red ForwardPass de 256 neuronas con funciones de activación "relu", seguida de una output layer de 1 neuronas con función de activación "sigmoid".
Al no existir una fórmula que permita obtener la configuración óptima de la red, el número de capas convolucionales, pooling layers, etc es necesario realizarla a través de una aproximación puramente experimental. Para ello jugaremos con las variables que nos permiten configurarla:

- Número de "epochs", es decir número de veces que la red es entrenada con los datos disponibles. Hay que tener en cuenta que, aunque a priori parezca una buena idea un número de epochs excesivamente alto puede acarrearnos overfiting en el modelo.
- Número de redes convolucionales y polling layers, además de su distribución y orden.
- Número de redes ForwardPass , numero de neuronas de cada una y su función de activación.
- Dropouts y BatchNormalization

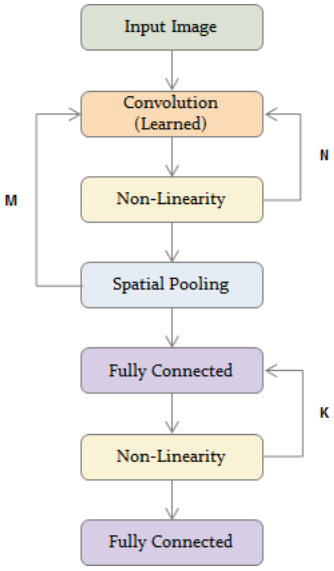


Figure 10: Arquitectura CONVNET

11.1 Modelo 1 y 2

Como se ha comentado en el capítulo 5.1, los modelos en Keras se definen como una secuencia de capas. La arquitectura descrita en este apartado será la utilizada en el proyecto, y en el apartado siguiente se realizaran numerosas variaciones y análisis para decidir la arquitectura y los parámetros óptimos para la red neuronal.
Como versión inicial y tal y como hemos mencionado en el capítulo 11, se probó con la arquitectura definida en la tabla 2. Como primer modelo se decidio dividir la red en 3 bloques para facilitar la explicación:

- Primer bloque: El primer bloque es un bloque convolucional típico formado por una capa convolucional de 32 filtros de tamaño 3x3. Posteriormente hay una función de activación rectificadora (RELU) y posteriormente se aplica una capa que realiza una operación de max-pooling con un entorno de actuación de 2x2.

- Segundo bloque: El segundo bloque es muy parecido al primero sin embargo se duplica el número de filtros a 64, el cual sigue siendo número pequeñoo. Para finalizar se aplica una capa de max-pooling. Po último lugar, existe una capa (Flatten) que convertirá la matriz de 2 dimensiones en un vector, permitiendo de esta forma que la salida pueda ser procesada por una capa totalmente conectada

- Tercer bloque: Las características obtenidas por la capas anteriores pasan después a este bloque formando por dos capas convencionales o completamente conectadas que elaboran la predicción final. Esta predicción es pasada a una capa final con 1 unidades, con una función de activació especial denominada sigmoid.

Configuración modelo 1	
Arquitectura	[CONV -> RELU -> POOL] * 2 ->FC -> RELU -> FC
Batch_size	32
Iteraciones	Número de muestras (80000) / batch_size (32)
Optimizador	Adam

Table 2: Configuración del modelo 1

Como era de esperar, el modelo tiene una arquitectura tan simple, que no fué capaz de aprender la relación entre los datos de entrada y los de salida. En el gráfico 11 podemos ver un claro ejemplo de underfitting, la red no termina de converger.

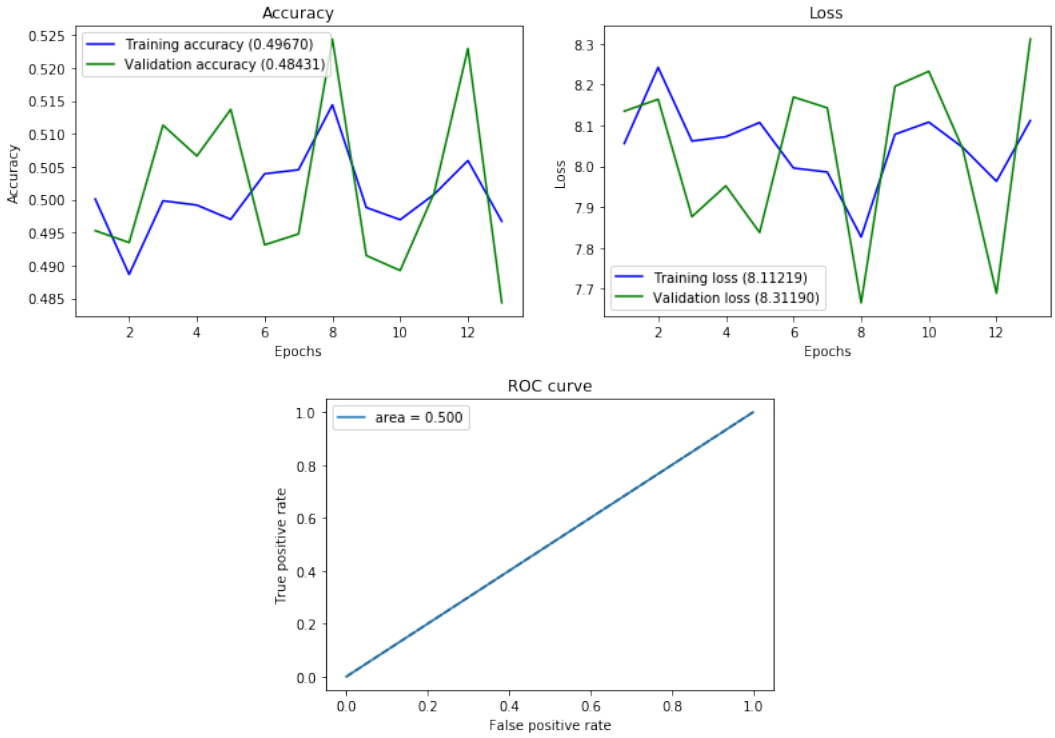


Figure 11: Resulatdos del modelo 2

Como mejora aplicamos Dropdout y Batch Normalization como mecanismos de optimización a cada una de nuestras capa convolucionadas para así, poner freno al underfitting producido anteriormente y a la vez prevenir un posible caso de overfitting. Se ha configurado la capa Dropdout para excluir aleatoriamente el 30% de las neuronas de la capa.

Una vez aplicado Dropdout y Batch Normalization obtuvimos la siguiente estructura:

Configuración modelo 2	
Arquitectura	[CONV -> BN -> RELU -> POOL -> Dropout(0.3)] * 2 ->FC -> RELU -> FC
Batch_size	31
Iteraciones	Número de muestras (80000) / batch_size (32)
Optimizador	Adam

Table 3: Configuración del modelo 2

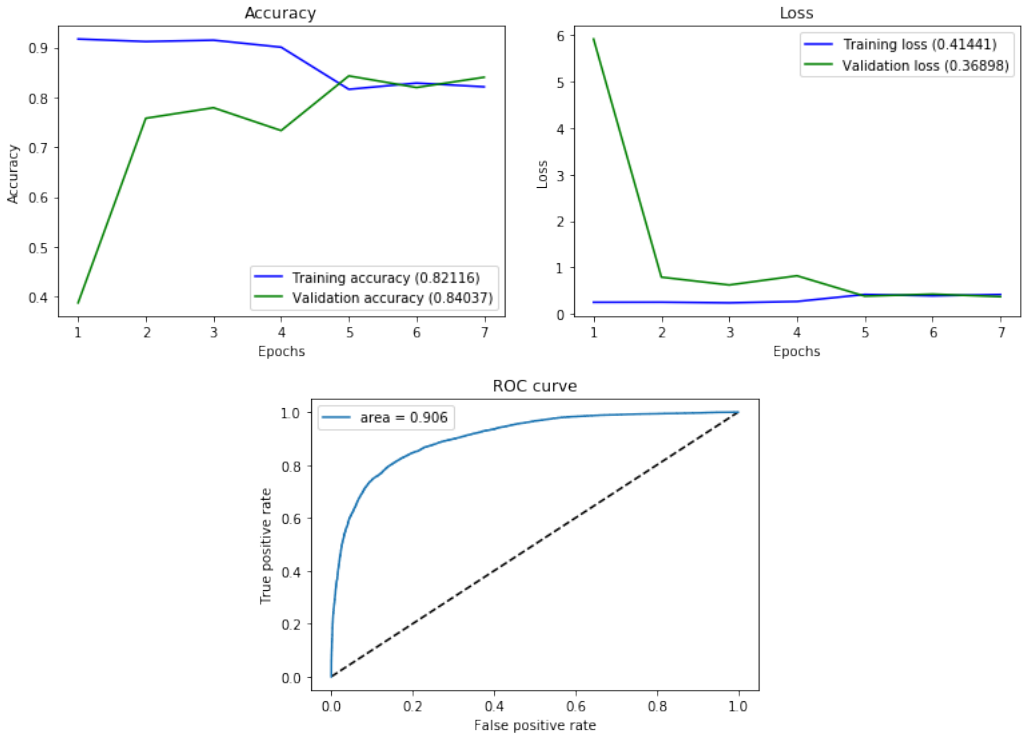


Figure 12: Resulatdos del modelo 2

Como podemos observar en la propia grafica de entrenamiento 12, las perdidas han dejado de converger apartir del epoch 5, quedandose ya muy atrás respecto a las perdidas en el aprendizaje del training set. Técnicamente no ha llegado a ver overfitting ya que la precisión en la validación no ha sufrido una reducción, pero si estamos en claro riesgo de sufrirlo a mayor número de iteracciones puesto que la ganancia de precision real en el modelo parece haber llegado a su límite con esta configuración.
Aún así obtuvimos una mejora bastante destacable respecto al modelo inicial. Con este cambio aumentamos el porcentaje al 88% de acierto

11.2 Modelo 3

Como el modelo definido en el capítulo 2 seguía siendo demasiado simple y no era capaz de aprender los detalles de las imagenes se decidió aumentar el número de capas ocultas para incrementar la capacidad de aprendizaje. Además de aumentar la profundiad de la neurona hicimos la red más ancha.

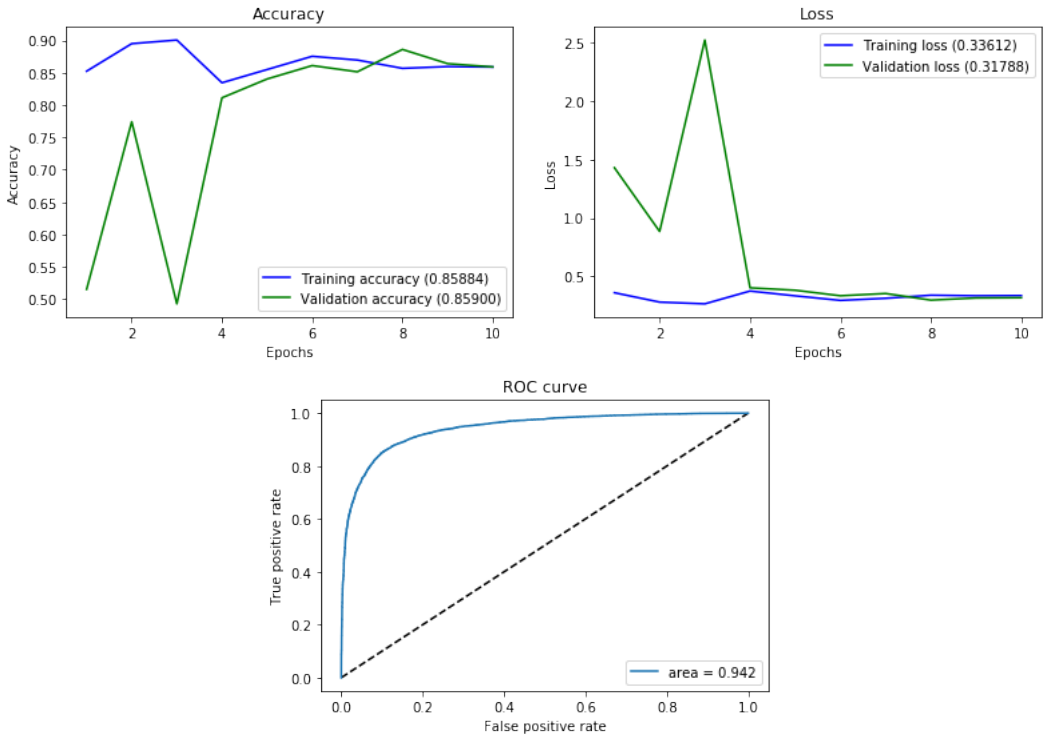
En esta versión se probó con la arquitectura definida anteriormente pero añadiendo un bloque convolucional nuevo. Este bloque es muy parecido al segundo donde simplemente de duplicó el número de filtros a 128. Además es cada todos los bloques convolucionales pasamos de tener 1 capa convolucional a tener dos por bloque. En todas la capas se sigue utilizando la función de activación ReLU.

Esto permitió en primer lugar extraer las características más importantes de la imagen y posteriormente en el siguiente nivel de profundidad suavizar la extracción para no perder información relevante.

En el siguiente apartado se evaluará el efecto de añadir a nuestra arquitectura a capa oculta el doble de neuronas de las que tiene actualmente.

Configuración modelo 3	
Arquitectura	[[CONV -> BN -> RELU] * 2 -> POOL -> Dropout] * 3 ->[FC -> RELU]*1 -> FC
Batch_size	31
Iteraciones	Número de muestras (80000) / batch_size (32)
Optimizador	Adam

Table 4: Configuración del modelo 3



Lo primero que se observa es la gráfica 11.2 de la evolución de la precisión en las predicciones para los datos de entrenamiento donde se observan tres fases relativamente diferenciadas. Las diferencias entre fases se observan con cambios en la curvatura de las gráficas, así como cambios bruscos en las pendientes.
En la primera fase la red se encuentra en un estado de inicialización, donde se puede apreciar un grave caso de overfitting o sobreaprendizaje. La primera fase dura aproximadamente desde el principio del entrenamiento hasta la época 4.

En la siguiente fase la red va aprendiendo a mejor ritmo a clasificar las distintas imagenes de cancer, observándose un aumento de pendiente, pudiendo situar esta fase entre las épocas 4 y 8 aproximadamente.
De la época 8 en adelante se observa que la gráfica vuelve a experimentar un cambio de curvatura, y la red estaría empezando a experimentar un underfitting, donde se ve que la precisión de entrenamiento disminuye. Este cambio nos lleva a pensar que el aumento de la densidad de la red en él último bloque convolucional, aporte más detalle al aprendizaje y así evitemos este caso de underfitting.

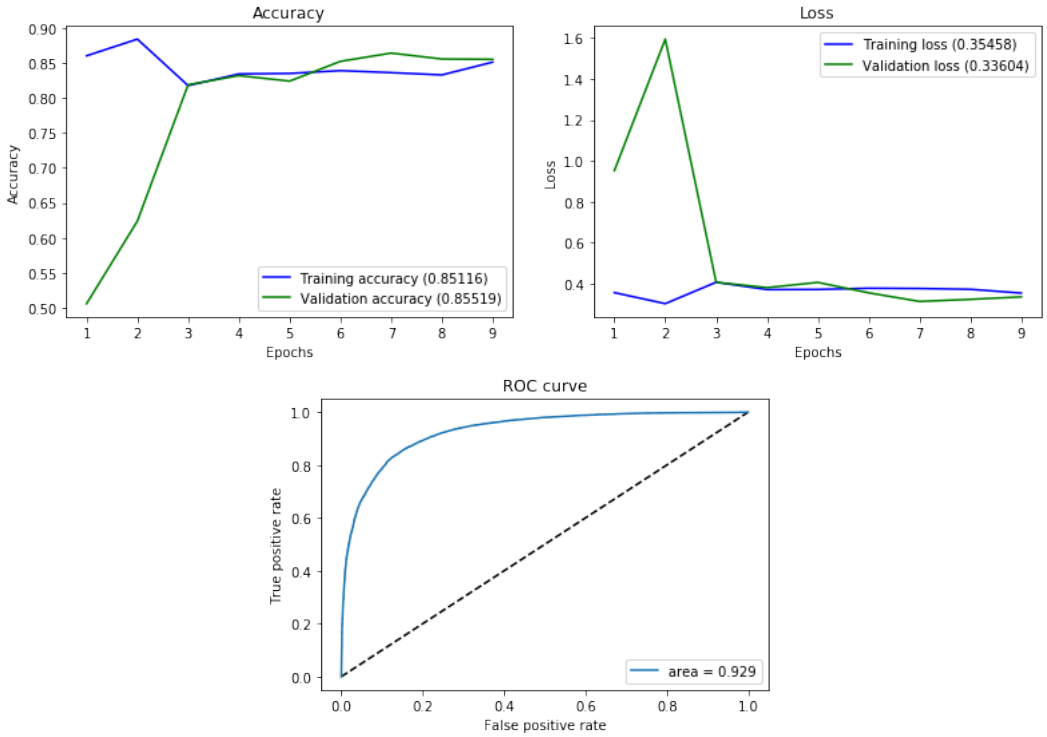
11.3 Modelo 4

El modelo que se ha propuesto en último lugar es el de la tabla 5. Se siguio con la idea de preservar el número de bloques convolucionales pero con la diferencia que vamos aumentando en cada bloque el número de capas convolucionales.

Configuración modelo 4	
Arquitectura	[[CONV -> BN -> RELU] * [1, 2, 3] -> POOL -> Dropout] * 3 ->[FC -> RELU]*1 -> FC
Batch_size	31
Iteraciones	Número de muestras (80000) / batch_size (16)
Optimizador	Adam

Table 5: Configuración del modelo 3

A pesar de aña



La gráfica 11.3 de la evolución de la precisión muestra un empeoramiento en el aprendizaje respecto al modelo3. El hecho de añadir progresivamente capas convolucionales a cada bloque no implico una mejora en el porcentaje de acierto final.

12 Resultados

Las arquitecturas descritas se han puesto en práctica sobre el conjunto dato previamente normalizado. Para el proceso de aprendizaje, se ha escogido el optimizador adam, completando el entrenamiento en 15 épocas, con un tamaño del batch de 32 muestras. Todos los modelos evaluados han sido configurados con el mismo optimizador, la misma función de activación, el mismo imageDataGenerator. En la siguiente tabla se observan las configuraciones con sus resultados respectivamente:

Como se aprecia en la Tabla 6, los tres modelos aportan buenos resultados, destacando especialmente el último modelo. Adicionalmente, se incluyen en esta misma tabla los resultados de la predicccción.

MODELO	**PRIVATE SCORE**
Modelo 1	0.52
Modelo 2	0.88
Modelo 3	0.92
Modelo 4	0.90

Table 6: Tabla comparativa con las diferentes precisiones en los datos de test para cada modelo

Comparando los resultados con los obtenidos se observa que los resultados del presente trabajo son relativamente buenos, siendo igual o mejor que cualquier resultado obtenido por cualquier modelo preentrenado vistos en la competición. Sin embargo, la comparación no es del todo justa, ya que en el presente trabajo me he visto limitado por la falta de tiempo por probar nuevas configuraciones con el modelo3 y en la competiciones han hecho uso de redes pre-entrenadas tales como Resnet50.

13 Conclusiones

Como se ha podido comprobar, el uso de las redes neuronales en ámbitos comunes es ya una realidad. A día de hoy, el mundo de las redes neuronales es una de las mayores líneas de investigación tanto en informática como en el estudio y recreación del cerebro humano, y merece la pena entender su funcionamiento básico.

En cuanto a los objetivos de este trabajo se han cumplido todos ellos. Primero, se llevó a cabo un primer acercamiento al mundo de las redes neuronales, describiendo su funcionamiento, arquitectura, y características básicas. Después, se presentó Keras, una librería Python que se ejecuta sobre Tensorflow o Theano (dos de los backend más conocidos de redes neuronales) y que facilita mucho la programación de los modelos de redes. Seguidamente, mediante cuatro ejemplos, se aplicaron todos los conocimientos descritos en este documento para demostrar la facilidad que existe hoy en día para

programar cualquier tipo de red neuronal.

Con el desarrollo de este proyecto podemos llegar a la conclusión de que entrenar una red neuronal para la clasificación de pocas clases puede ser rápida y menos laboriosa, en cambio, si queremos entrenar una red para la clasificación de mas de 100 clases o con imagenes que tienen muchos detalles puede ser más costosa y obtendremos menor porcentaje de acierto. Es por esto que utilizar una red pre-entrenada pueda ser una mejor opción en estos caso.