



Universitat de les Illes Balears

Escuela Politécnica Superior

Memoria del Trabajo de Fin de Grado

Aprendizaje profundo para segmentar espinas dendríticas.

Joaquín Jiménez Morales

Grado de Ingeniería Electrónica Industrial y Automática

Año académico 2016-17

DNI del alumno: 43219613Z

Trabajo tutelado por: Dra. Yolanda González Cid
Departamento de Ciencias matemáticas e informática

Autorizo a la Universidad a incluir este trabajo en el Repositorio Institucional para su consulta de acceso abierto y difusión en línea, con finalidades exclusivamente académicas y de investigación.	Autor		Tutor	
	Sí	No	Sí	No
	X		X	

Palabras clave del trabajo:

Deep Learning, Procesamiento de imágenes, Espinas dendríticas, Matlab.

ÍNDICE

RESUMEN	1
RESUM.....	2
ABSTRACT	3
AGRADECIMIENTOS.....	4
1. CONTEXTUALIZACIÓN	5
2. INTRODUCCIÓN	7
3. LA NEURONA	9
3.1 CEREBRO	9
3.2 MORFOLOGÍA NEURONAL BÁSICA.....	9
3.3 DENDRITAS.....	10
4. APRENDIZAJE AUTOMÁTICO (MACHINE LEARNING)	11
4.1 CONCEPTOS PREVIOS.....	11
5. APRENDIZAJE PROFUNDO (DEEP LEARNING).....	13
5.1 REDES NEURONALES CONVOLUCIONALES	14
6. HERRAMIENTAS PARA EL DESARROLLO	17
6.1 SOFTWARE	17
6.2 HARDWARE	17
7. DESARROLLO DEL PROYECTO	19
7.1 FASE 0: ADQUISICIÓN DE CONOCIMIENTOS PREVIOS	19
7.2 FASE 1: FUSIÓN DE LA INFORMACIÓN Y OBTENCIÓN DE PUNTOS.....	20
7.3 FASE 2: RECONOCIMIENTO DE ESPINAS EN LA IMAGEN	25
7.4. FASE 3: TRANSFERENCIA DE APRENDIZAJE DE UNA RED EXISTENTE.....	28
7.4.1 BASE DE DATOS MNIST.....	28
7.4.2 CONFIGURACIÓN INICIAL	29
7.4.3 RESULTADOS DE LA RED ORIGINAL	30
7.5 FASE 4: ENTRENAMIENTO USANDO SUBVENTANAS 2D COMO INPUT	30
7.6 FASE 5: ENTRENAMIENTO USANDO CUBOS 3D COMO INPUT	35
8. CASOS EXPERIMENTALES CONSIDERADOS	39
8.1 DATASETS.....	39
8.2 DIMENSIONES	40
8.3 DESCRIPCIÓN DE LAS PRUEBAS REALIZADAS.....	40
9. RESULTADOS EXPERIMENTALES.....	43
9.1 RED BIDIMENSIONAL PARA EL DATASET1 CON VENTANAS 17X17.....	43
9.2 RED BIDIMENSIONAL PARA EL DATASET4 CON VENTANAS 17X17.....	45
9.3 RED BIDIMENSIONAL PARA EL DATASET1 CON VENTANAS 33X33.....	48
9.4 RED BIDIMENSIONAL PARA EL DATASET4 CON VENTANAS 33X33.....	49
9.5 RED TRIDIMENSIONAL PARA EL DATASET5 CON VENTANAS 28x28xvar	51
9.6 RED TRIDIMENSIONAL PARA EL DATASET5 CON VENTANAS MÁXIMAS.	52
9.7 RESULTADOS DE LAS PRUEBAS	54
10. ANÁLISIS DE LOS RESULTADOS.....	55
10.1 DATASETS UTILIZADOS.....	55
10.2 DIMENSIONES	56
10.3 2D / 3D	57
11. CONCLUSIONES Y TRABAJO FUTURO	59
BIBLIOGRAFÍA.....	61
ANEXO 1. DESCRIPCIÓN DEL CÓDIGO EN MATLAB.....	63

ÍNDICE DE FIGURAS

Figura 1. 1: Gráfico de las múltiples disciplinas que intervienen en el estudio del cerebro. Fuente: [2].....	6
Figura 3.2. 1: Estructura general de una neurona. Fuente [5].....	9
Figura 3.3. 1: Esquema de la localización de las espinas y lugar de sinapsis. Fuente [7].....	10
Figura 3.3. 2: Imagen del CSIC de una dendrita y sus espinas.	10
Figura 5. 1: Representación de la curva de aprendizaje automático y profundo. Fuente [12]....	13
Figura 5.1. 1: Obtención del mapa de activación tras aplicar la operación de convolución. Fuente [15].....	15
Figura 5.1. 2: Ejemplo de la operación de max-pooling. Fuente [16].	15
Figura 5.1. 3: Operación de ReLu. Fuente [17].....	16
Figura 5.1. 4: Esquema de las capas de una red neuronal convolucional. Fuente [18].	16
Figura 7.2. 1: Fotografías de microscopía tomadas por el CSIC. Directorio api.if6.1.8enero.	20
Figura 7.2. 2: Nube de puntos de la espina 20 generada por uno de los archivos del directorio api.if6.1.8enero.	21
Figura 7.2. 3: Volumen tridimensional de la espina 20 del directorio api.if6.1.8enero. [visualizar_fichero.m].....	22
Figura 7.2. 4: Valores de pixel que se quieren obtener a partir de la información tridimensional (rojo).....	22
Figura 7.2. 5: Representación gráfica del problema a resolver. [visualizacion2.m]	23
Figura 7.2. 6: Planteamiento del problema como la intersección entre planos.....	23
Figura 7.2. 7: Puntos de intersección de un plano con una espina.....	24
Figura 7.2. 8: Vista de la intersección de un conjunto de espinas con los planos 54 y 55.....	24
Figura 7.2. 9: Visión global y ampliada de la intersección de todas las espinas con todos los planos.....	25
Figura 7.2. 10: Visión lateral de la intersección de todas las espinas con todos los planos.....	25
Figura 7.3. 1: Puntos x,y calculados para una espina. [nuevo2.m]	26
Figura 7.3. 2: Representación de los puntos obtenidos de las espinas en la imagen original....	27
Figura 7.3. 3: Segmentación de espinas en la imagen original.....	27
Figura 7.5. 1: Patches de la imagen original que contienen espina.	30
Figura 7.5. 2: Patches de la imagen original que no contienen espina.....	31
Figura 7.5. 3: Patches 17x17 de espina (rojo) y no espina (azul) de la imagen original.....	32
Figura 7.5. 4: Resultado tras la ejecución del proceso de entrenamiento y testeo.....	33
Figura 7.5. 5: Precisión en la clasificación de imágenes en los dos posibles valores de salida...	33
Figura 7.5. 6: Imágenes en las que el valor real no ha coincidido con el predicho.....	33
Figura 7.5. 7: Resultados para el primer directorio expresados en forma de matriz de confusión.....	34
Figura 7.5. 8: Filtros obtenidos para la red entrenada con los datos del directorio api.if6.1.8enero.....	35
Figura 7.6. 1: Proceso de homogenización y creación del nuevo cubo.	36
Figura 7.6. 2: Muestra de las diferentes capas del cubo que contienen información tridimensional de la espina.	37
Figura 7.6. 3: Muestra de las diferentes capas del cubo que no contienen espina.....	37

Figura 7.6. 4: Imágenes extraídas durante el proceso de detección de cubos de espinas (rojo) y no espinas (azul).....	38
Figura 7.6. 5: Muestra de la vista tridimensional de tres espinas y una región de no espina.	38
Figura 8. 2: Esquema de los diferentes escenarios experimentales.....	41
Figura 9.1. 1: Matriz de confusión, ratio de acierto y confianza de la primera red.	43
Figura 9.1. 2: Filtros de la primera red una vez entrenada.	44
Figura 9.1. 3: Errores durante el entrenamiento para la primera red	44
Figura 9.1.4: Matrices de confusión de la primera red testeada con dataset2 (izquierda), dataset3 (derecha) y dataset4 (abajo).....	45
Figura 9.2. 1: Matriz de confusión, ratio de acierto y confianza de la segunda red.	46
Figura 9.2. 2: Filtros de la segunda red una vez entrenada.	46
Figura 9.2. 3: Errores durante el entrenamiento para la segunda red	46
Figura 9.2. 4: Matrices de confusión de la primera red testeada con dataset1 (izquierda), dataset2 (derecha) y dataset3 (abajo).....	47
Figura 9.3. 1: Matriz de confusión, ratio de acierto y confianza de la tercera red.	48
Figura 9.3. 2: Errores durante el entrenamiento para la tercera red.	48
Figura 9.3. 3: Matriz de confusión de la tercera red testeada con dataset4.....	49
Figura 9.4. 1: Matriz de confusión, ratio de acierto y confianza de la cuarta red.....	49
Figura 9.4. 2: Filtros de la cuarta red una vez entrenada.....	50
Figura 9.4. 3: Errores durante el entrenamiento para la cuarta red.....	50
Figura 9.4. 4: Matriz de confusión de la cuarta red testeada con dataset1.....	50
Figura 9.5. 1: Datos entrenamiento y testeo de la red.	51
Figura 9.5. 2: Matriz de confusión para la quinta red.	51
Figura 9.5. 3: Errores y ratio de acierto de la quinta red.	52
Figura 9.5. 4: Matrices de confusión de la quinta red testeada con dataset6 (izquierda) y dataset7 (derecha).....	52
Figura 9.6. 1: Matriz de confusión, ratio de acierto y confianza de la sexta red.	53
Figura 9.6. 2: Errores durante el entrenamiento para la sexta red.	53
Figura 9.6. 3: Matrices de confusión de la sexta red testeada con dataset6 (izquierda) y dataset7 (derecha).....	53

ÍNDICE DE TABLAS

Tabla 1: Resultados de las pruebas bidimensionales.....	54
Tabla 2: Resultados de las pruebas de la red tridimensional.....	54

RESUMEN

El estudio del cerebro es necesariamente un área multidisciplinar que requiere tanto de expertos en biología, farmacología o bioquímica como de expertos en el tratamiento, análisis y procesamiento de imágenes.

Actualmente, los neurocientíficos de diferentes instituciones buscan el apoyo de la tecnología para automatizar ciertos procesos, como puede ser el reconocimiento de algunos elementos en imágenes cerebrales. También utilizan nuevas técnicas para la búsqueda de patrones que no pueden ser detectados a simple vista.

Así, el objeto del presente proyecto es el de analizar las imágenes tomadas del córtex cerebral para, mediante técnicas de aprendizaje profundo, poder reconocer y detectar unos elementos presentes en las neuronas cerebrales conocidos como espinas dendríticas.

La finalidad última del proyecto consiste en que una vez entrenada la red artificial profunda, ésta sea capaz de distinguir en las imágenes de microscopía del cerebro entre regiones que contengan alguna espina y regiones que no. Esta tarea, puede resultar de utilidad para neurocientíficos que actualmente se encuentran estudiando la relación de estas dendritas neuronales con deficiencias en las capacidades cognitivas e intelectuales del propio individuo. A día de hoy, se conoce que precisamente estas dendritas son, en gran medida, las responsables de la generación y propagación de los impulsos eléctricos entre neuronas.

RESUM

L'estudi del cervell és necessàriament un camp multidisciplinari que requereix tant d'experts en biologia, farmacologia o bioquímica com d'experts en el tractament, anàlisi i processament d'imatges.

Actualment, els neurocientífics de diferents institucions cerquen el recolzament de la tecnologia per poder automatitzar processos com pot ser el reconeixement d'alguns elements a imatges cerebrals. També utilitzen noves tècniques per a la recerca de patrons que no puguin ser detectats a simple vista.

Així, l'objectiu del present projecte és el d'analitzar les imatges preses del còrtex cerebral per, mitjançant tècniques d'aprenentatge profund, poder reconèixer i detectar uns elements presents a les neurones cerebrals coneguts com a espines dendrítiques.

La finalitat última del projecte consisteix en que un vegada entrenada la xarxa artificial profunda, aquesta sigui capaç de distingir en les imatges de microscòpia del cervell entre regions que contenguin alguna espina i regions que no. Aquesta tasca, pot resultar d'utilitat per neurocientífics que actualment es troben estudiant la relació d'aquestes dendrites neuronals amb deficiències en les capacitats cognitives i intel·lectuals del propi individu. A dia d'avui, se coneix que precisament aquestes dendrites són, en gran mesura, les responsables de la generació i propagació d'impulsos elèctrics entre neurones.

ABSTRACT

The study of the brain is essentially a multidisciplinary field which requires from biology, pharmacology or biochemistry experts as well as from data analysts or image processing experts.

Currently, neuroscientists from different institutions are automating processes like the recognition of certain objects from an image. They are also using new techniques to detect patterns that cannot be detected at first sight.

Thus, the purpose of this project is to analyze cortex images with Deep learning techniques to be able to recognize some elements called dendritic spines.

The main goal of this project is to be able to distinguish whether we have spine or not in a specific region of the brain once the deep neural network is trained. This task, can be helpful to neuroscientists who are studying the relationship between spines and cognitive abilities or intellectual dysfunction in humans. We already know that these elements are vital in neuron communication and have a strong impact on electric pulses propagation.

AGRADECIMIENTOS

A Yolanda, por darme la oportunidad de colaborar con un proyecto a gran escala y por la atención y guía recibida a lo largo de estos meses de trabajo.

A José Miguel Espadero, de la Universidad Rey Juan Carlos, por su gran ayuda a la hora de organizar los datos de entrada a la red neuronal.

1. CONTEXTUALIZACIÓN

Uno de los principales objetivos de la neurociencia es el de entender los mecanismos biológicos responsables de la actividad mental humana. Concretamente, el estudio del córtex cerebral es, sin lugar a dudas, uno de los mayores retos científicos de las próximas décadas. Esto se debe a que es precisamente la actividad de nuestra corteza cerebral la que nos da las capacidades y habilidades que nos distinguen del resto de mamíferos. Gracias al desarrollo y evolución de esta corteza somos capaces de realizar tareas complejas y actividades propias del ser humano como escribir un libro, componer una canción o desarrollar e inventar todo tipo de artilugios.

Por este motivo, se inició el *Blue Brain Project* en 2005 cuando, con la colaboración conjunta de IBM y *l'Ecole Polytechnique Fédérale de Lausanne (Suiza)*, se decidió crear un modelo de cerebro funcional. Para ello, utilizarían un método conocido como ingeniería inversa mediante el cual, a partir de las diferentes simulaciones y resultados, intentarían comprender el funcionamiento del cerebro. A finales del 2006, el *Blue Brain Project* creó el modelo de la unidad básica del cerebro, la columna neocortical. Sin embargo, debido a los objetivos impuestos por el ambicioso proyecto, éste acabó convirtiéndose en una iniciativa a nivel internacional. La contribución española a este proyecto internacional, el *Cajal Blue Brain Project*, empezó en enero del 2009 dirigida por la Universidad Politécnica de Madrid (UPM) y el Consejo Superior de Investigaciones Científicas (CSIC).

Actualmente, la doctora Yolanda González del departamento de Ciencias Matemáticas e Informática está colaborando desde 2016 en este proyecto, de la mano de investigadores del CSIC y varios investigadores de diversas Universidades de la Comunidad de Madrid y en el que, durante este curso académico, he tenido la oportunidad de colaborar [1].

Por lo tanto, este Trabajo Final de Grado forma parte de un estudio de investigación mucho más amplio y es solo un pequeño paso para conseguir algunos de los objetivos globales como pueden ser:

- Entender el mapa de conexiones sinápticas para poder reconstruir todos sus componentes.
- Entender mejor las enfermedades cerebrales como el Alzheimer.
- Idear nuevos métodos para procesar y analizar datos experimentales.
- Desarrollar la tecnología necesaria para graficar o visualizar y poder así estudiar mejor las funciones neuronales.

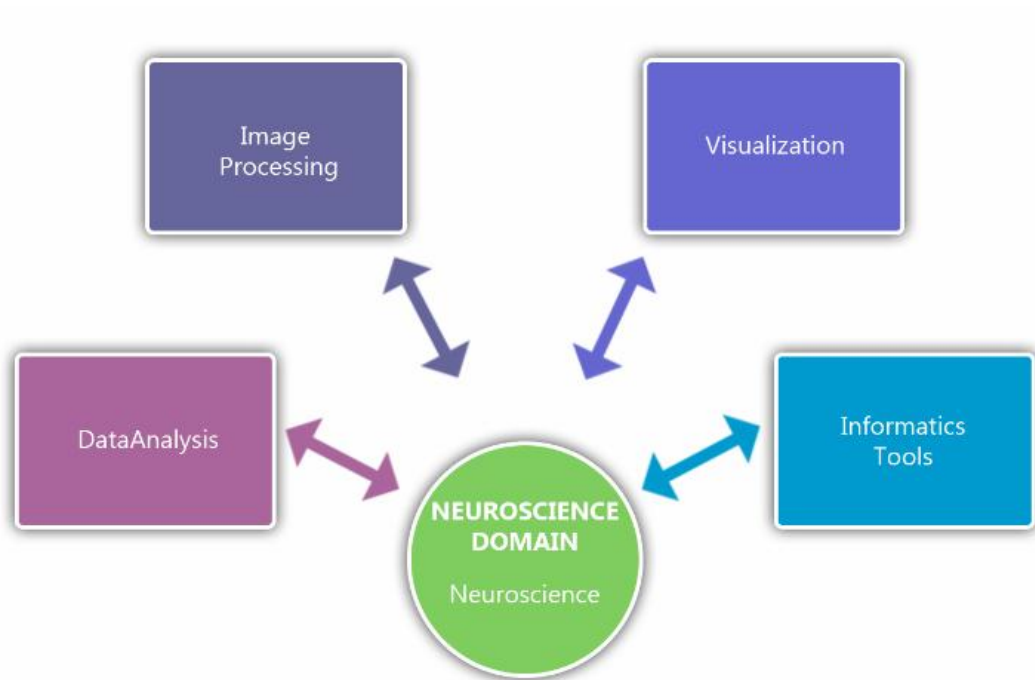


Figura 1. 1: Gráfico de las múltiples disciplinas que intervienen en el estudio del cerebro. Fuente: [2]

2. INTRODUCCIÓN

El Trabajo Final de Grado que se detalla en este informe tiene como objetivo reconocer espinas dendríticas en imágenes digitales. El fin último es el de automatizar el proceso de detección de espinas y facilitar así el estudio de las neuronas para, en un futuro, compararlas más fácilmente y detectar más rápidamente posibles anomalías que se relacionen con enfermedades o deficiencias cognitivas.

El conjunto de imágenes sobre las que se trabajará son imágenes de microscopía de cortezas cerebrales reales proporcionadas por el propio CSIC. Además de estas imágenes en escala de grises, los únicos datos de los que partimos son unos puntos tridimensionales de cada espina a partir de los cuales hemos de ser capaces de entrenar una red neuronal artificial para que aprenda a distinguir lo que es y lo que no es una espina dendrítica.

El proceso de aprendizaje se llevará a cabo mediante técnicas de aprendizaje profundo, también conocido como *Deep Learning* que en los últimos años ha tenido y va a tener una gran importancia, especialmente en el desarrollo de la inteligencia artificial. De esta manera, si alimentamos a nuestra red con las imágenes necesarias, podemos lograr que ésta aprenda a detectar los elementos de interés de manera autónoma e interpretar futuras imágenes en base al entrenamiento que ha recibido.

Para poder hacer que la red “aprenda”, se deberán generar unos datos de entrada, que serán subventanas de la imagen también conocidos como *patches*, y unos datos de salida que contienen información sobre la respuesta correcta que debería obtener la red, también conocidos como valores de *ground truth*. Por lo tanto, se deberá realizar un trabajo previo de preparación de los datos para conseguir relacionar la información de los puntos tridimensionales con cada una de las imágenes y así obtener estos valores de entrada y salida que permitan entrenar la red.

Una vez entrenada, se podrá utilizar la red para reconocer si un patch contiene o no una espina con una cierta probabilidad. Cuantas más muestras se le presenten en el proceso de entrenamiento, más precisa y fiable será la red. Sin embargo, introducir un número demasiado elevado puede suponer un coste computacional también elevado para un incremento mínimo en el porcentaje de acierto.

3. LA NEURONA

A continuación, se introducen algunos conceptos sobre el funcionamiento de las neuronas cerebrales. Conocer su estructura y forma de interactuar no solo será útil para entender las imágenes que estamos tratando, sino que también forma parte de un punto fundamental del proyecto como es el aprendizaje profundo mediante redes neuronales convolucionales, que se explicará más adelante.

3.1 CEREBRO

El cerebro es el órgano más complejo del cuerpo humano. En un humano típico, la corteza cerebral (la parte más grande) se estima que contiene 10 mil millones de neuronas y todo el cerebro se estima que contiene entre 86 y 100 mil millones de neuronas. Las neuronas son en realidad un tipo de células especializadas del sistema nervioso. Su función principal es la recepción y conducción de estímulos en forma de impulsos nerviosos. [3] y [4].

3.2 MORFOLOGÍA NEURONAL BÁSICA

Las neuronas se pueden dividir de manera general en tres partes: un cuerpo celular (soma); una o varias prolongaciones cortas que generalmente transmiten impulsos hacia el soma celular (dendritas); y una prolongación larga (axón), que conduce los impulsos desde el soma hacia otra neurona.

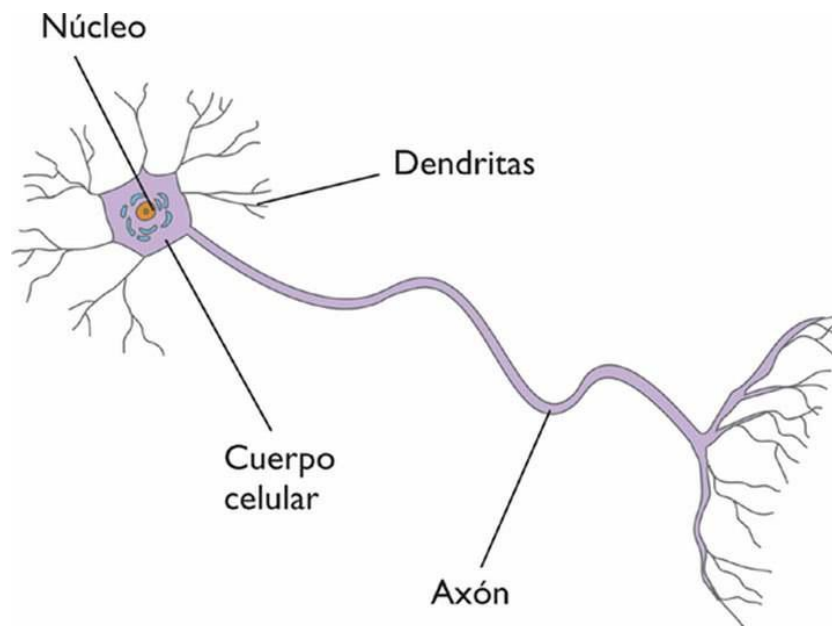


Figura 3.2. 1: Estructura general de una neurona. Fuente [5].

3.3 DENDRITAS

Las dendritas neuronales son unas cortas prolongaciones dedicadas a la recepción de estímulos. Más concretamente, sirven como receptores de impulsos provenientes de un axón perteneciente a otra neurona. Su principal función es recibir los impulsos de otras neuronas y enviarlos hasta el soma de la neurona. A lo largo de las dendritas existen las espinas dendríticas, pequeñas prolongaciones citoplasmáticas, que es donde se produce la sinapsis. Son precisamente estas pequeñas prolongaciones que aumentan la superficie y que se encargan de recibir la información las que nos interesa detectar, ya que su atrofia o desarrollo deficiente está relacionado con déficits cognitivos como el síndrome de Down. [6]

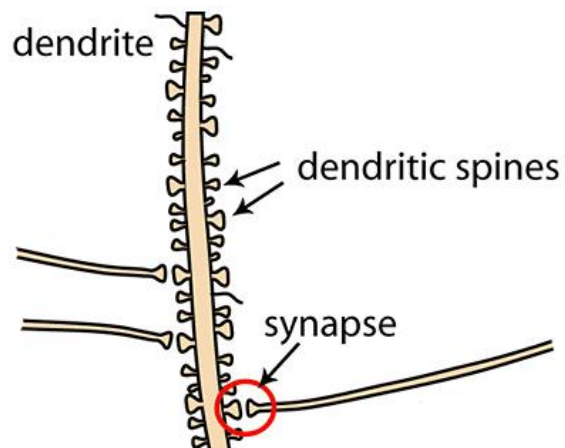


Figura 3.3. 1 Esquema de la localización de las espinas y lugar de sinapsis. Fuente [7].

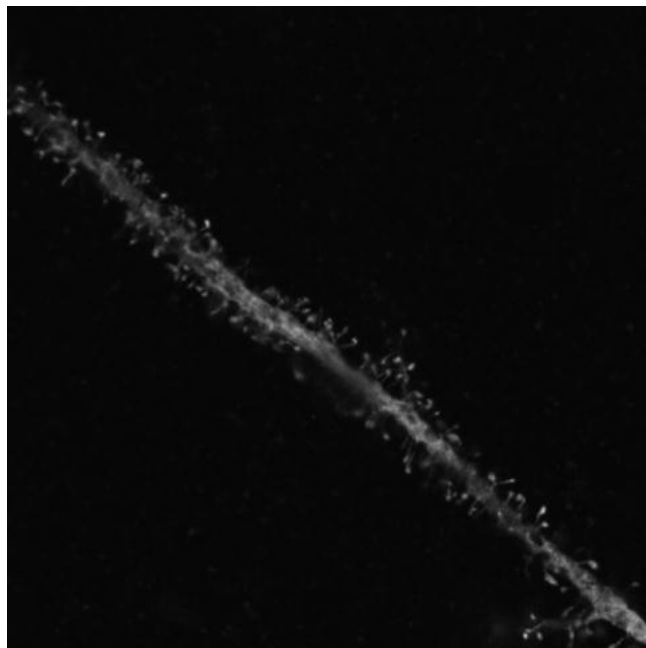


Figura 3.3. 2: Imagen del CSIC de una dendrita y sus espinas.

4. APRENDIZAJE AUTOMÁTICO (MACHINE LEARNING)

El término *machine learning* se traduce al español como aprendizaje automático ya que pretende crear sistemas que aprendan automáticamente. Por aprender, nos referimos a que la máquina es capaz de detectar patrones o características comunes de entre los numerosos datos que se le proporcionan. Es decir, que, en realidad, lo que la máquina aprende es un algoritmo que será capaz de revisar los datos para predecir comportamientos futuros. Por automático, nos referimos a que estos sistemas mejoran y se van perfeccionando si incrementamos el número de datos, pero no es necesaria la constante intervención humana.

Según autores como Arthur Samuel, *machine learning* es el campo de estudio que da a los ordenadores la habilidad de aprender sin haber sido explícitamente programados. Otros, como Tom Mitchell, dan una definición más moderna y lo definen de la siguiente manera: Se dice que un programa de ordenador aprende de la experiencia (E) con respecto a una serie de tareas (T) y con un rendimiento (P) si su rendimiento en T, medido por P, mejora con E.

En cierto modo, lo que se pretende es imitar el proceso de aprendizaje de los humanos mediante máquinas que puedan aprender de la experiencia de una manera más rápida y precisa.

En estos últimos años, debido a la facilidad de generar esta inmensa cantidad de datos (Big Data), estas técnicas son cada vez más útiles y se utilizan actualmente para detectar fraudes en transacciones, predecir fallos en equipos tecnológicos, hacer prediagnósticos médicos basados en síntomas del paciente, predecir el tráfico urbano, prever qué empleados serán más rentables el año que viene, por citar algunos ejemplos. [8]

4.1 CONCEPTOS PREVIOS

Este trabajo se basa en gran parte en una disciplina relativamente novedosa por lo que muchos de los términos y conceptos que se utilizarán posteriormente pueden resultar ambiguos si no se definen y se establecen previamente. [9].

Además, la explicación de estos términos facilitará la comprensión del método de trabajo utilizado, así como los pasos seguidos y el funcionamiento del proyecto en sí mismo.

Data set: Es el conjunto de datos utilizados para entrenar al sistema. En nuestro caso, siempre que hablemos de data set nos referiremos a las imágenes de las dendritas proporcionadas por el CSIC.

Inputs: Entrada de datos al sistema que queremos entrenar, como por ejemplo imágenes.

Outputs: Predicción o valor que nuestro sistema ha generado para una entrada concreta.

Training: Proceso de entrenamiento de la red mediante el cual, a partir de los datos de entrada y el resultado esperado generaremos un sistema capaz de realizar predicciones acertadas.

Durante este proceso se producen errores en la predicción. En base a estos errores, se generará el modelo final de forma que, para cada input, se obtiene un output que, en caso de no corresponder con el esperado, provoca una penalización o ajuste de los parámetros de la red (**backpropagation**).

Testing: Proceso mediante el cual comprobamos la eficiencia del proceso de aprendizaje contrastando los valores obtenidos con los reales y calculando el número de aciertos o precisión de nuestra máquina. Generalmente, el número de muestras destinadas al proceso de entrenamiento es dos o tres veces mayor al de testeo. El porcentaje exacto depende del problema a resolver, así como de la experiencia y los resultados que obtenga el investigador.

Peso: es el valor que tiene una neurona artificial asociada a una conexión con otra neurona de la red neuronal y que determina la contribución que tiene ésta al valor de salida obtenido. El valor de los pesos es variable y se irá ajustando durante el proceso de entrenamiento para obtener siempre el mínimo error posible en la predicción.

Sesgo: valor de desplazamiento vertical que, al igual que el peso, se ajusta durante el proceso de entrenamiento para dar preferencia a ciertos outputs en función de los inputs recibidos durante el proceso de especialización y aprendizaje.

Por otro lado, es importante conocer que el machine learning se divide en dos áreas generales:

Aprendizaje supervisado (clasificación/regresión)

En el aprendizaje supervisado la máquina producirá un algoritmo que relacione las entradas y salidas del sistema que son previamente conocidas. Se conocen los datos de entrada y la salida/resultado que debería obtener nuestra máquina. Cuantos más datos se aporten, más probabilidades de éxito tendrá de acertar con la predicción resultante.

Dentro del aprendizaje supervisado diferenciamos dos tipos de problemas a resolver: problemas de regresión y problemas de clasificación.

En los problemas de regresión el objetivo es predecir los resultados en base a una función continua, mientras que en los problemas de clasificación tratamos de predecir resultados discretos de salida. En este último caso la salida será una categoría a la que pertenece el dato de entrada.

Aprendizaje no supervisado

El aprendizaje no supervisado resulta muy útil para abordar problemas en los que no sabemos cómo han de ser nuestros resultados. La estrategia consiste en deducir patrones o estructuras a partir de los datos de entrada sin necesariamente conocer qué efectos tendrán a priori en la salida. En este tipo de aprendizaje no hay retroalimentación sobre las predicciones obtenidas.

Este **proyecto** se centra en un tipo de **aprendizaje automático supervisado** ya que disponemos de los valores reales de salida (respuesta correcta) durante la fase de entrenamiento. Se tratará como un problema de clasificación ya que deberemos distinguir en las imágenes entre si hay espina o si no hay, que corresponderá a los valores 1 y 0 respectivamente.

5. APRENDIZAJE PROFUNDO (DEEP LEARNING)

El aprendizaje profundo, también conocido como *Deep Learning* es una de las vertientes del aprendizaje automático. Siendo éste mucho más sofisticado y autónomo en cuanto al método de aprendizaje.

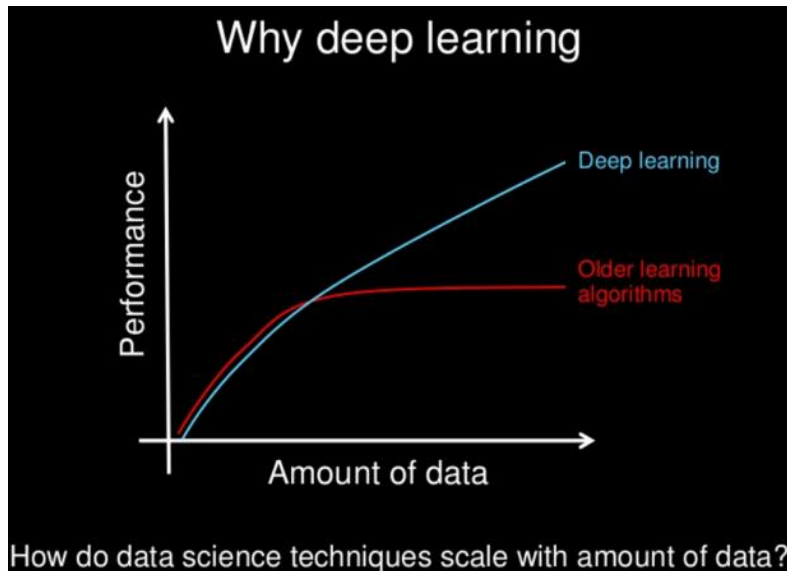


Figura 5.1: Representación de la curva de aprendizaje automático y profundo. Fuente [12].

Tanto con *machine learning* como *deep learning* se pretende detectar reglas o patrones a partir de un gran número de datos de entrada para que finalmente, el software sea capaz de generar nuevas reglas y asociaciones cuando se introduce un input desconocido. En *Deep Learning*, el método de aprendizaje intenta imitar el comportamiento del cerebro humano para el cual utiliza una estructura basada en un conjunto de capas o unidades neuronales.

Este conjunto de unidades neuronales funciona como una sucesión o cadena donde a partir de unos inputs dados se genera una respuesta. Esta respuesta está basada en el método de ponderación que consiste en asociar diferentes pesos para cada entrada. El resultado o salida de esa capa constituye la entrada de la siguiente capa que también reajustará sus parámetros en función de los inputs correspondientes.

Este método de aprendizaje, junto con la gran cantidad de datos que requiere, permite reducir considerablemente el margen de error y aumentar la precisión en sus conclusiones. Cuantas más capas presente el sistema, más complejo y preciso será. Sin embargo, si sobredimensionamos el número de capas necesarias, el sistema aprenderá demasiado específicamente para esos inputs y no generará respuestas acertadas cuando generalicemos y utilicemos imágenes nuevas (*overfitting*). Si el número de capas es muy reducido, no tendremos la certeza suficiente y obtendremos una respuesta errónea con más frecuencia. Por eso, el número de capas a utilizar no viene determinado o impuesto, sino que más bien, debe ser algo que, como muchos otros parámetros, vayamos ajustando y perfeccionando en función de los resultados obtenidos.

5.1 REDES NEURONALES CONVOLUCIONALES

De los múltiples tipos de redes que engloba el *Deep Learning*, en este proyecto se trabajará con las conocidas como redes neuronales convolucionales (CNN, en inglés de *Convolutional Neural Network*). El motivo es que, aunque existen otros tipos de redes como las recurrentes o las de creencia profunda, se ha demostrado que las CNNs producen muy buenos resultados al trabajar con imágenes o formas estructurales espaciales.

Uno de los motivos por los que funcionan tan bien es porque, a diferencia de otras redes, las redes neuronales convolucionales suponen explícitamente que su entrada son imágenes, lo que permite codificar ciertas propiedades en la arquitectura ganando así eficiencia y reduciendo considerablemente la cantidad de parámetros de la red.

En el caso de redes neuronales ordinarias se observa que no escalan bien para imágenes de mucha definición. Por ejemplo, en el problema de MNIST, las imágenes son de 28x28; por lo que una sola neurona plenamente conectada en una primera capa oculta de una red neuronal ordinaria tendría $28 \times 28 = 784$ pesos. Esta cantidad todavía parece manejable, pero es evidente que esta estructura totalmente conectada no funciona bien con imágenes más grandes. Si tomamos el caso de una imagen de mayor tamaño, por ejemplo, de 200x200 con 3 canales de color RGB, daría lugar a neuronas que tienen $200 \times 200 \times 3 = 120.000$ pesos. El contar con tal cantidad de parámetros supondría un desperdicio de recursos y conduciría rápidamente al sobreajuste.

Las CNN trabajan modelando de forma consecutiva pequeñas piezas de información, y luego combinando esta información en las capas más profundas de la red. Una manera de entenderlas es que la primera capa intentará detectar los bordes y establecer patrones de detección de bordes. Luego, las capas posteriores tratarán de combinarlos en formas más simples y, finalmente, en patrones de las diferentes posiciones de los objetos, iluminación, escalas, etc. Las capas finales intentarán hacer coincidir una imagen de entrada con todos los patrones y llegar a una predicción final como una suma ponderada de todos ellos. De esta forma las redes neuronales convolucionales son capaces de modelar complejas variaciones y comportamientos dando predicciones bastantes precisas. [13]

5.1.1 ESTRUCTURA

Como ya se ha mencionado anteriormente, su funcionamiento está basado en un conjunto de capas de filtros convolucionales de una o más dimensiones.

Durante el proceso aparecen dos fases claramente diferenciadas. Por un lado, la fase de extracción de características que se compone por las neuronas convolucionales y de reducción de muestreo. Por otro lado, una fase final donde se realiza la clasificación final en base a las características obtenidas en la fase previa.

La dimensión de los datos disminuye a medida que éstos avanzan en la red. Esto provoca que la perturbación que reciben las neuronas de las últimas capas debida a los datos de entrada

sea menor. Dicho de otro modo, las neuronas de las últimas capas son menos sensibles a los datos iniciales, pero son activadas por características más complejas. [14]

Capa convolucional

Como su propio nombre indica, en esta capa se realiza la operación de convolución que recibe como entrada la imagen y posteriormente aplica sobre ella un filtro, también conocido como *kernel*, que devuelve un mapa de características de la imagen original. Se dispondrá de tantos mapas como filtros se apliquen. De esta manera, se reduce el tamaño de los parámetros de la red. [15]

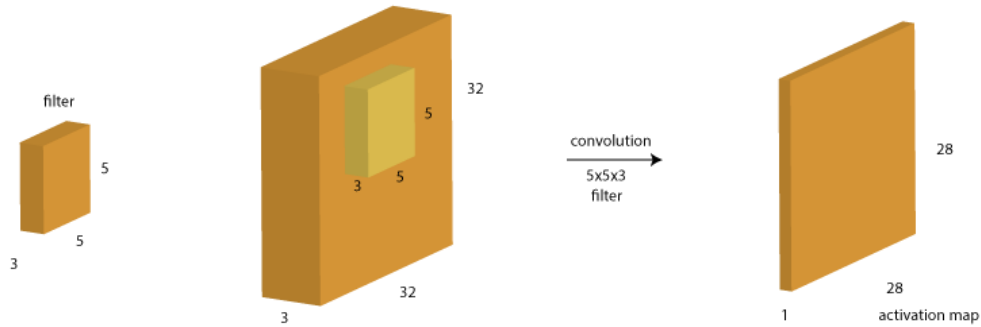


Figura 5.1. 1: Obtención del mapa de activación tras aplicar la operación de convolución. Fuente [15].

Capa de reducción (pooling)

Generalmente, esta capa se encuentra después de la convolucional. Su principal función consiste en reducir las dimensiones del volumen de entrada para la siguiente capa convolucional. Por este motivo, también se la denomina como de reducción de muestreo. Aunque se produzca una ligera pérdida de información en este proceso, resulta beneficioso realizarla porque al tener un menor tamaño provocará una menor sobrecarga en las próximas capas de la red y además contribuye a reducir el sobreajuste.

Encontramos diferentes tipos de operaciones de pooling, aunque la más utilizado suele ser el *max-pooling* que reduce la información de la ventana seleccionada al valor máximo que hay en esta.

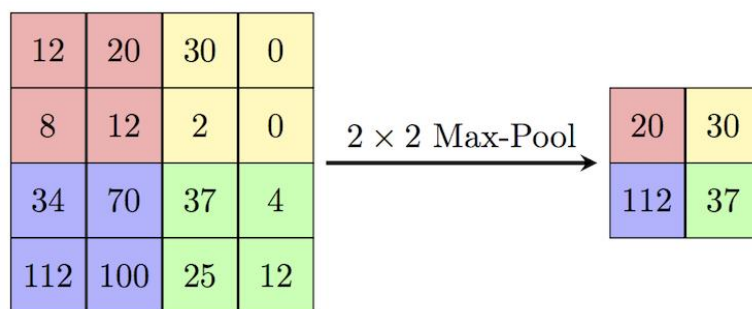


Figura 5.1. 2: Ejemplo de la operación de max-pooling. Fuente [16].

ReLU

Es muy habitual encontrar un proceso de rectificación llamado ReLU (*Rectified Linear Unit*) que consiste en un proceso en el cual se reemplaza el valor de aquellos píxeles negativos por cero. De esta manera, podemos introducir no linealidades a nuestro sistema que es más realista y se corresponde mejor con los problemas que encontramos habitualmente.

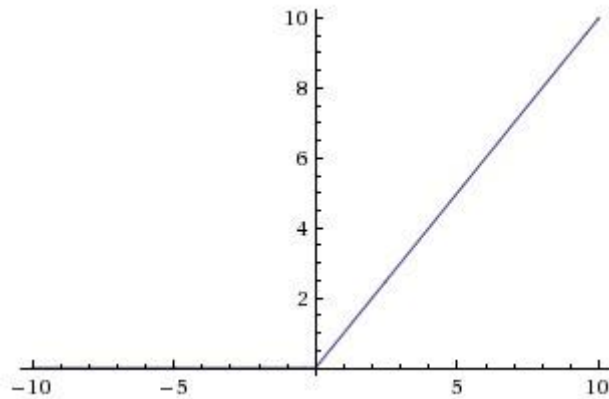


Figura 5.1. 3: Operación de ReLu. Fuente [17].

Capa clasificadora totalmente conectada (*Fully Connected layer*)

Al final de las capas convolucional y de pooling, las redes utilizan generalmente capas completamente conectadas en la que cada píxel se considera como una neurona separada. Que esté completamente conectada significa que cada neurona de la capa anterior está conectada a cada neurona de la capa siguiente. La función de esta capa es utilizar las características obtenidas en las capas anteriores de pooling y convolución para clasificar la imagen de entrada en varias clases.

Es decir, que para resumirlo de manera sencilla y visual la estructura de la red presentaría el siguiente aspecto, siendo el tipo y número de capas variables en cada caso:

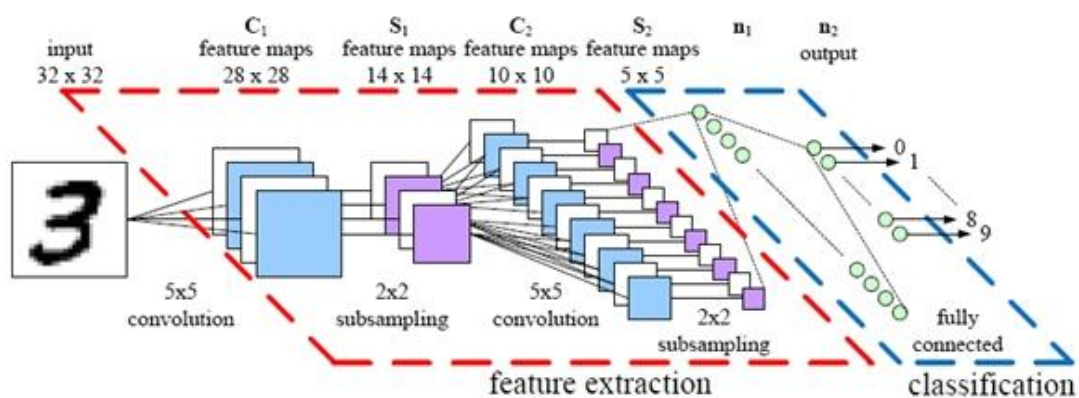


Figura 5.1. 4: Esquema de las capas de una red neuronal convolucional. Fuente [18].

6. HERRAMIENTAS PARA EL DESARROLLO

6.1 SOFTWARE

Microsoft Word 2016

Para la redacción de la memoria se ha utilizado el programa de office Microsoft Word en su versión 2016.

Matlab 2017a

Para el proceso de desarrollo del algoritmo y uso de redes convolucionales se ha utilizado el programa Matlab en su versión completa del año 2017a. También pueden utilizarse versiones anteriores como la 2016b que ya incluyen la *Toolbox* CNN necesaria para el uso de redes neuronales convolucionales.

Inicialmente, se hicieron pruebas con versiones de Matlab 2011a y 2014b, pero finalmente se optó por las versiones más modernas que incluían nuevas funciones y librerías necesarias para el desarrollo del proyecto.

También es necesaria una tarjeta gráfica que permita la ejecución del programa (CUDA 3.0 o superior para tarjetas NVIDIA).

Estos programas se han utilizado sobre el sistema operativo Windows 7 Pro-64 bits.

6.2 HARDWARE

La máquina utilizada para la ejecución del programa desarrollado en Matlab es un ordenador portátil Lenovo Z500. Algunas características de interés pueden ser:

Procesador: Intel® Core™ i7-3612QM CPU 2.10GHz

Memoria(RAM): 8 GB

Tarjeta gráfica: GeForce GT 635M - versión 306.97

El tipo y características del ordenador no son relevantes para las fases iniciales del proyecto y pruebas a pequeña escala. Sin embargo, realizar el entrenamiento de la red para un número elevado de imágenes puede suponer un tiempo considerable y la utilización de un ordenador superior puede agilizar bastante este proceso. Sin embargo, una vez entrenada ya podrá usarse con normalidad sin tener que esperar varios minutos

7. DESARROLLO DEL PROYECTO

En este capítulo, se explicará el proceso de desarrollo del proyecto que se ha estructurado en cinco fases. En cada una de ellas, se describen, de manera secuencial, los pasos, problemas detectados, soluciones encontradas y resultados obtenidos.

Para conseguir una mejor comprensión, también se mostrarán algunos resultados parciales que permitan ver de una forma clara el momento o fase del proyecto en que nos encontramos. Además, se añadirán las pruebas necesarias para justificar las soluciones finalmente adoptadas.

El conjunto de figuras mostradas en este capítulo no pretende mostrar el funcionamiento final del proyecto (que se incluye en el apartado de pruebas) sino demostrar la lógica y el razonamiento seguido hasta llegar a la solución finalmente implementada.

7.1 FASE 0: ADQUISICIÓN DE CONOCIMIENTOS PREVIOS

Esta etapa inicial corresponde a un largo proceso de formación y búsqueda de información sobre un tema, hasta entonces desconocido por el estudiante, como es el aprendizaje automático.

Antes incluso de comenzar a resolver el problema, era necesario adquirir una formación básica e indispensable. Durante esta etapa, se finalizaron exitosamente los bloques iniciales del curso “Machine Learning” de Coursera dirigido por el experto Andrew Ng [19]. También, se cursó otro más específico de la rama de aprendizaje profundo utilizando redes neuronales convolucionales para el reconocimiento visual [20].

Al mismo tiempo, se consultó numerosa bibliografía y libros como “Deep Learning” de Ian Goodfellow [21] o “Hello-world in Tensorflow” [22] de Jordi Torres. No hay que olvidar que la inmensa mayoría de la información sobre este tema se encuentra en inglés, por lo que es más recomendable consultarla directamente en ese idioma que de fuentes traducidas donde hay mucha menos información. En especial, resultaron muy instructivas las lecciones o charlas en modo video de todo tipo de autores o universidades como la de California.

Además de la información teórica sobre aprendizaje automático y redes neuronales, también se comenzó a investigar sobre qué plataformas se podrían utilizar para desarrollar el proyecto. Los candidatos que mejor se ajustaban a las necesidades del proyecto fueron Matlab y Tensorflow (Python).

Finalmente, tras investigar y ver las librerías y trabajos previos con Matlab, se decidió utilizar esta opción con la que resultaría más cómodo e intuitivo trabajar, dada la experiencia previa del alumno con este software.

7.2 FASE 1: FUSIÓN DE LA INFORMACIÓN Y OBTENCIÓN DE PUNTOS

Una vez estudiado el caso y la posibilidad de realizar el proyecto en Matlab comienza un proceso de pruebas y aproximaciones a lo que era un problema de grandes dimensiones. Es decir, en lugar de plantear el objetivo final de reconocer espinas dendríticas, se van abordando objetivos pequeños como, por ejemplo, el de interpretar y mostrar la información de la que se disponía inicialmente.

En realidad, la información se encuentra organizada en varios directorios. Cada directorio contiene el conjunto de muestras pertenecientes a una dendrita. La información inicial de la que disponemos es la siguiente:

- Conjunto de imágenes de microscopía.
- Puntos xyz que forman el volumen de la espina.

A diferencia de otros proyectos de detección de objetos donde en una imagen 2D es suficiente para representar toda la información disponible, en este caso, al tratar con elementos tan pequeños y en un espacio tridimensional, será necesario utilizar un conjunto de varias imágenes para describir un solo conjunto de trabajo. Es decir, las imágenes con las que trabajamos son fotos tomadas a diferentes alturas sobre una misma dendrita. A continuación, se muestra un pequeño ejemplo:

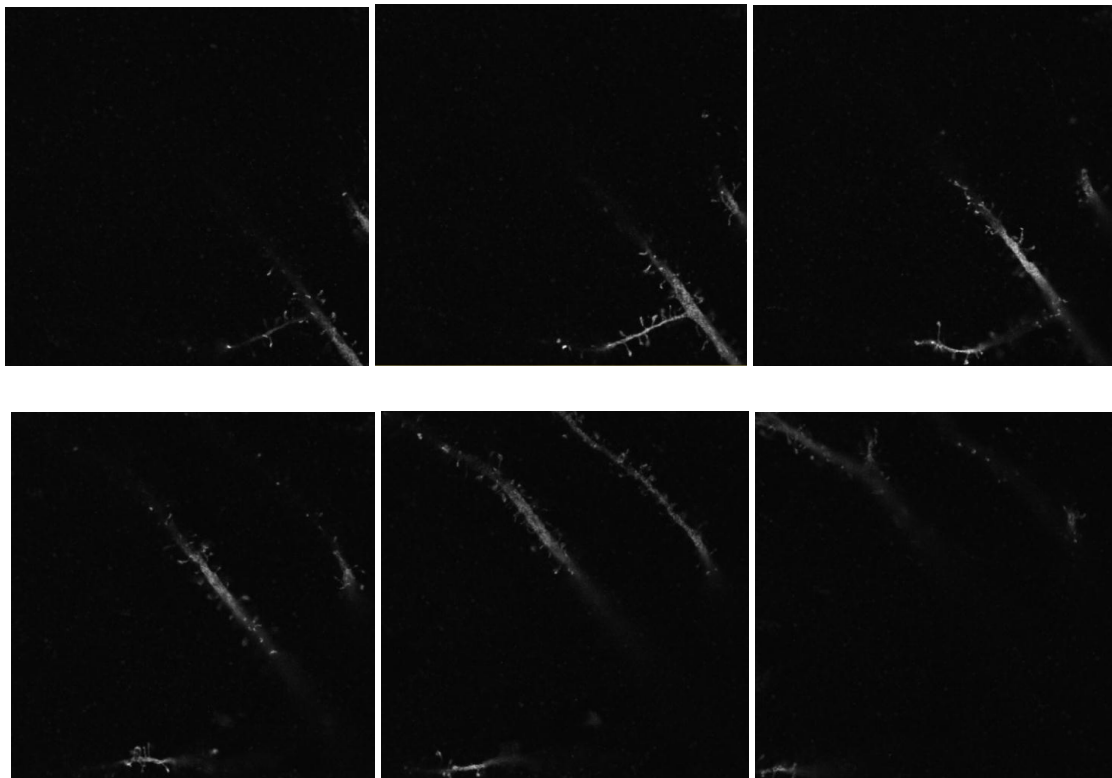


Figura 7.2.1: Fotografías de microscopía tomadas por el CSIC. Directorio api.if6.1.8enero.

En el ejemplo anterior, se han elegido tan solo 6 imágenes del conjunto de las 56 disponibles para el directorio de esa dendrita con el objetivo de ilustrar el concepto que se citaba anteriormente. Es decir, que las imágenes de un conjunto no son independientes, sino que contienen información sobre diferentes capas de una misma muestra.

Para este caso concreto, se han utilizado 56 fotografías o capas de la espina pero, para cada dendrita habrá un directorio. Esto supone que se trabajará con un número variable de imágenes que definan las diferentes dendritas. Habitualmente, este número se encuentra entre 30 y 70 planos de corte. De igual forma, el número de espinas presentes en cada directorio también será diferente, por lo que el código finalmente generado debe tener un carácter generalista y obtener la información de cada directorio automáticamente leyendo previamente el conjunto de imágenes correspondientes.

El segundo tipo de datos disponible es el conjunto de puntos que la estructura volumétrica de cada espina. Las primeras pruebas se realizaron sobre el directorio "api.if6.1.8enero" donde encontramos un número total de espinas de 180, o lo que es lo mismo, 180 archivos diferentes con varios cientos de puntos 3D. El número total de puntos (x, y, z) asciende a unos 96000 solo para ese directorio. Este valor variará en función de la dendrita ya que tendrá un número diferente de espinas (o archivos) de diferente tamaño en función del número de puntos que se utilicen para definirlos.

De una manera más explícita y por poner un ejemplo que facilite la comprensión, para el caso de la espina 0 del respectivo directorio tendremos 320 puntos que definan su volumen, para la espina 1 dispondremos de 483 puntos que la definan, para la espina 20 tendremos 479 puntos...

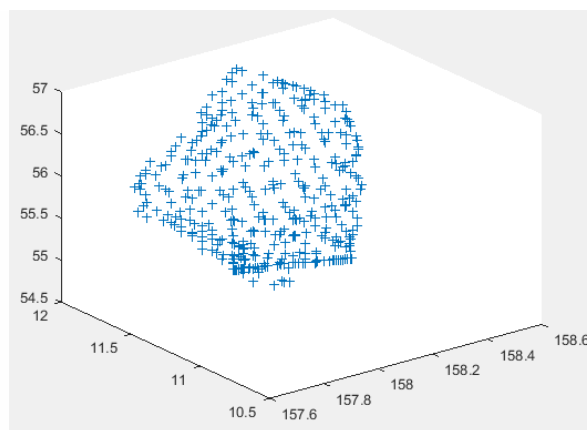


Figura 7.2. 2: Nube de puntos de la espina 20 generada por uno de los archivos del directorio api.if6.1.8enero.

Se comenzará visualizando la espina en Matlab a partir de la nube de puntos que se ha mostrado previamente. Además de la información de los puntos fue necesaria la elaboración de un nuevo fichero que no solo tuviera información de los puntos sino de cómo éstos estaban relacionados. Para nuestro caso concreto, la superficie estaría definida por triángulos por lo que se necesitaba un fichero que relacionara los 3 puntos que formaban cada uno de estos triángulos. A esta información se le denominó índices de los triángulos. A modo de ejemplo, si la primera línea de índices fuera 1, 6, 8 implica que tenemos una primera superficie triangular definida por los puntos 1, 6 y 8. Cada uno de estos puntos se encuentra en otro fichero con su

información xyz correspondiente. De esta manera, bastaría con leer los datos de este fichero con una función (que no es propia de Matlab) a la que llamamos `read_off`. El resultado, tras leer y representar estos puntos para una espina concreta se muestra a continuación:

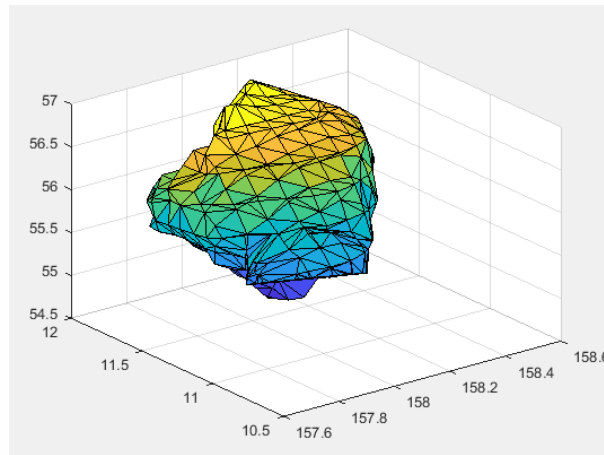


Figura 7.2. 3: Volumen tridimensional de la espina 20 del directorio `api.if6.1.8enero`. [`visualizar_fichero.m`]

El primer **objetivo** que se plantea a continuación, es encontrar los valores de los puntos de cada espina que constituyen su contorno sobre las imágenes de microscopía. Dicho de otro modo, interesa encontrar los puntos de la periferia de cada espina en cada plano para el que se ha tomado una imagen. De una manera visual, consistiría en encontrar las coordenadas de los píxeles en rojo:

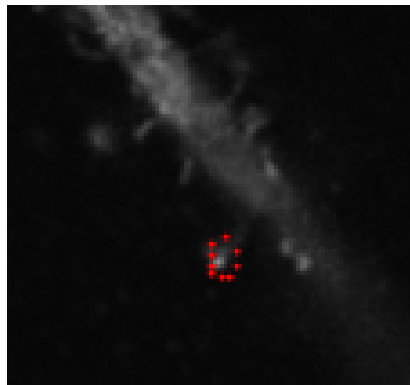


Figura 7.2. 4: Valores de pixel que se quieren obtener a partir de la información tridimensional (rojo).

El reto era claro, poder unificar la información tridimensional y poder relacionarla con las imágenes de microscopía.

Para ello se probaron diferentes estrategias: la primera consistió en intentar generar la superficie de un plano y la espina superpuestas para poder obtener así los puntos de intersección buscados. No obstante, no existía ningún método en Matlab para hacer esta operación. El motivo es sencillo: tenemos un plano a una altura determinada y una nube de puntos en un volumen tridimensional. La función "`trisurf`" de Matlab no genera más puntos, sino que simplemente une los que hay y grafica una superficie entre estos.

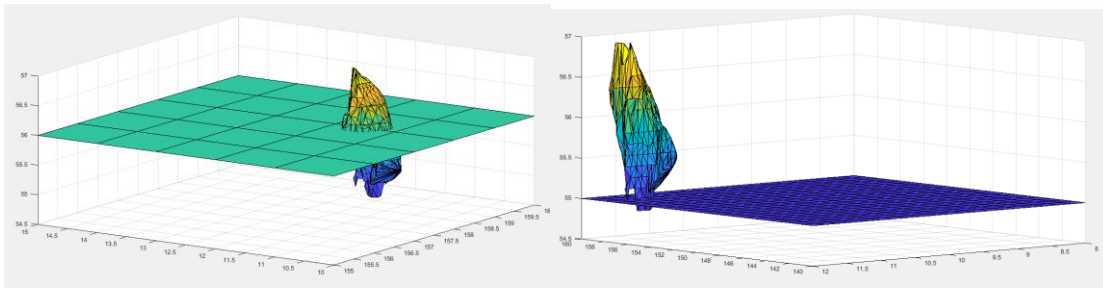


Figura 7.2.5: Representación gráfica del problema a resolver. [visualizacion2.m] y [unplano_unaespina.m].

A nivel visual se graficaba correctamente, pero no era posible obtener la información de los puntos de la espina sobre ese plano de corte directamente, que es realmente la información que se necesitaba para poder trabajar con las imágenes (puntos rojos de la figura 7.2.4). Por lo tanto, se propuso una alternativa diferente.

Siguiendo con el mismo enfoque y en vista de la necesidad de tener el valor concreto de dichos puntos, se planteó el problema como una intersección entre planos. La intersección sería entonces entre un plano fijo a una altura determinada y los diferentes triángulos que definen el volumen de la espina, delimitados por cada conjunto de tres puntos. Como resultado, se obtendría una recta que debería acotarse al interior de cada triángulo. De manera más visual:

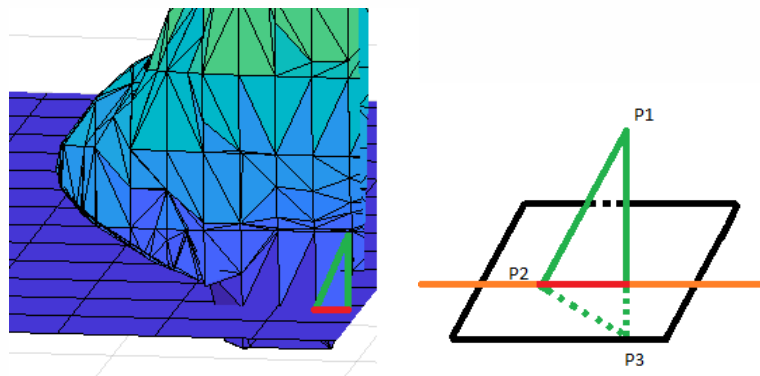


Figura 7.2. 6: Planteamiento del problema como la intersección entre planos.

Para calcular el segmento de recta rojo, que representa la información útil, se debería calcular en primer lugar la intersección entre el plano y el triángulo de la espina, representada por la recta naranja. A continuación, deberíamos limitar esa recta a los puntos que formen parte de la espina. Para ello, será necesario calcular la intersección entre la recta naranja y las rectas P1-P2, P2-P3, P3-P1. De esta manera, obtendríamos dos soluciones que corresponderían a los puntos que limitan los valores de la recta que pertenecen a esa espina y que son solución.

Como se puede intuir, este método tampoco fue el que finalmente se utilizó ya que era computacionalmente costoso y bastante ineficiente. Esto se debe a que habría que calcular una intersección entre el plano fijo y la superficie triangular además de las tres intersecciones con las rectas que lo delimitan. Este proceso se debía repetir para cada uno de los pequeños triángulos que forman la superficie de esta espina y después con el conjunto de las espinas de esa dendrita, para finalmente tener las intersecciones de las espinas de cada una de las varias dendritas que debemos procesar.

Finalmente, una vez descartada la opción anterior se volvió a enfocar el problema basándonos únicamente en la información disponible, una nube de puntos. De este modo surgió una nueva idea. La solución sería interpolar. Así, para cada par de puntos que se encontraba en el rango de altura indicado, obtendríamos sus valores interpolados x e y para una altura conocida (altura del plano).

Este cálculo era mucho más rápido y simple de realizar y comenzó a dar buenos resultados. Primero se probó para una espina individual. Después, se probó representar todas las espinas y mostrar la intersección con un solo plano. Finalmente, se obtuvo la versión más general donde se apreciaban los puntos de intersección de todas las espinas con todos los planos. Los resultados pueden visualizarse a continuación:

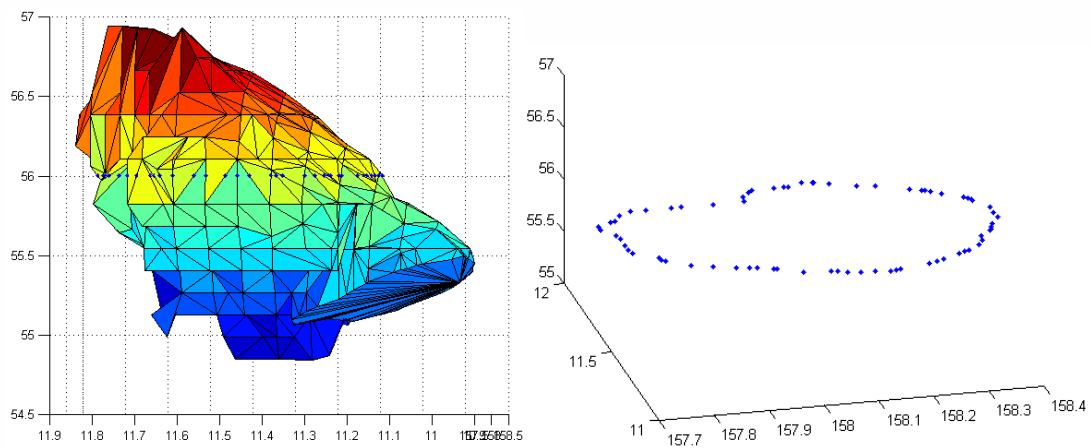


Figura 7.2. 7: Puntos de intersección de un plano con una espina

Los nuevos puntos azules calculados son los que finalmente se utilizarán ya que corresponden a los puntos de borde de cada espina de las imágenes de microscopía.

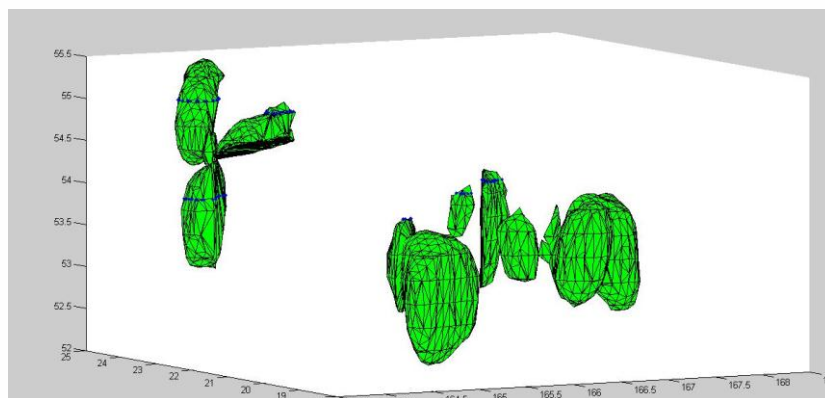


Figura 7.2. 81: Vista de la intersección de un conjunto de espinas con los planos 54 y 55.

Finalmente, se generalizó el código para mostrar todas las espinas y todos las intersecciones con cada uno de los planos a los que se tomaron las imágenes. Representadas de manera global se obtiene la siguiente figura:

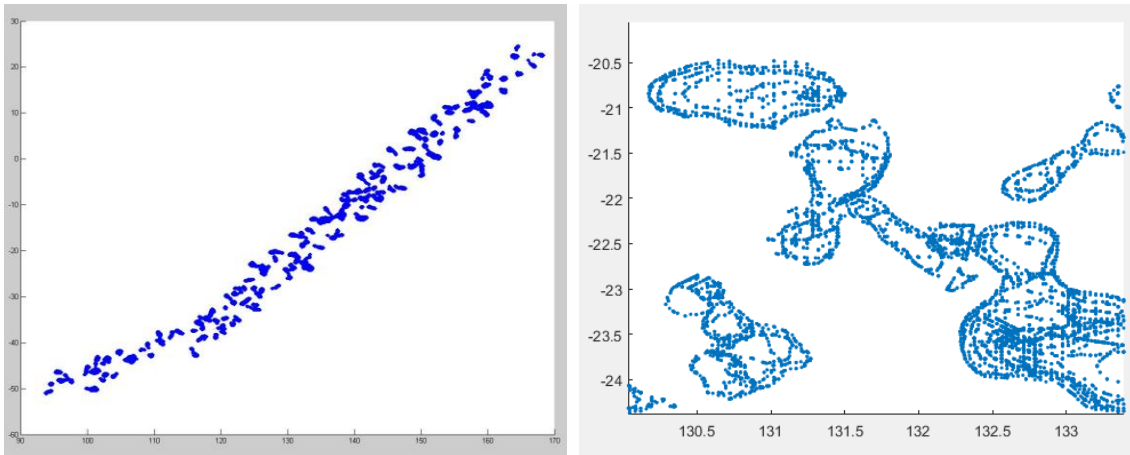


Figura 7.2.92: Visión global y ampliada de la intersección de todas las espinas con todos los planos.

Conviene aumentar la imagen para realmente ver que los puntos calculados coinciden con los buscados.

Es precisamente en esta última imagen donde se puede apreciar perfectamente cómo a cada espina le corresponde un contorno de puntos. Es en cierto modo similar a un mapa topográfico donde podemos encontrar curvas de nivel que representan diferentes alturas.

Finalmente, se muestra una vista lateral de las intersecciones de espinas y cada uno de los planos. Para este ejemplo, se pueden ver 56 alturas claramente diferenciadas, pero como ya se ha comentado antes, el número de alturas y espinas dependerá del directorio.

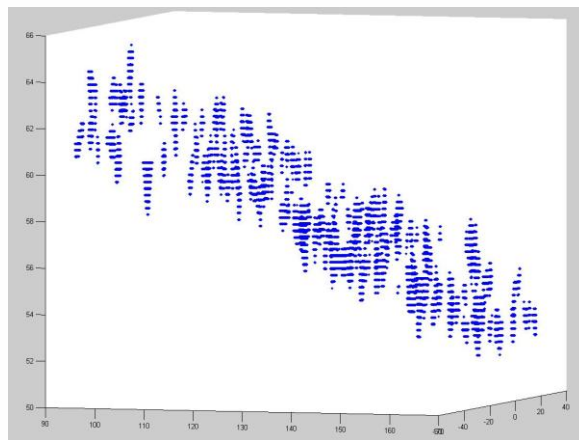


Figura 7.2. 103: Visión lateral de la intersección de todas las espinas con todos los planos.

7.3 FASE 2: RECONOCIMIENTO DE ESPINAS EN LA IMAGEN

Una vez se han obtenido los valores de los puntos que corresponden a las espinas de cada imagen, habrá que aplicar la correspondiente transformación para poder ubicarlos en la posición de píxel adecuada.

Los datos en forma de imágenes y los ficheros de puntos tridimensionales son de la misma dendrita pero no están directamente relacionados. Es decir, tendremos información tridimensional de partes de la espina que no se han fotografiado. Esto es así porque

disponemos únicamente de 56 imágenes en los ficheros png (56 alturas) y sin embargo encontramos 32440 valores diferentes de z (alturas). Los valores z están comprendidos entre 51.6266 y 65.4991 para el fichero que hemos tomado de ejemplo. Esto supone, que si la distancia a la que se han tomado las fotografías es esquidistante podemos tomar el valor de altura 51.6266 como la altura del primer plano del que tenemos información (imagen 000) y 65.4991 como la altura de la última imagen (nº 55). Así pues, la altura de los sucesivos planos se calculará como la altura mínima considerada más un pequeño incremento. El valor de este incremento viene dado simplemente como la diferencia de las alturas z entre los pasos (nº imágenes) disponibles.

El valor x no corresponde a la posición x en píxeles de la imagen y sucede lo mismo con la variable y. Es por esto, que se establecieron unas fórmulas que permitan transformar puntos volumétricos a coordenadas de la imagen y viceversa así como alturas del plano con su respectivo número de imagen. Al realizar la transformación, se apreciaba que existía una gran cantidad de puntos concentrados en una variación muy pequeña de la imagen. Así que cada par de valores (x, y) se redondeó al píxel más próximo obteniendo una forma aproximada de la espina.

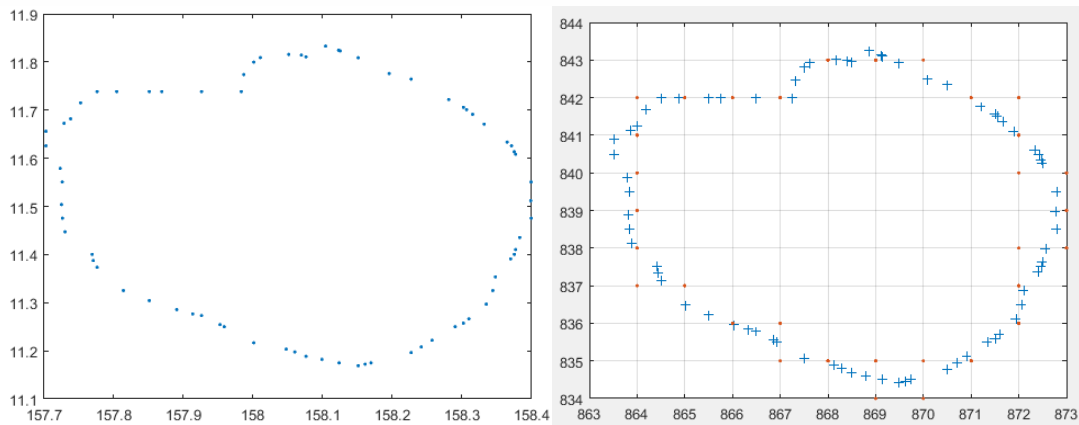


Figura 7.3. 1: Puntos x, y calculados para una espina. [nuevo2.m]

Como se puede ver, los puntos rojos correspondiente a los píxeles de la imagen no corresponden al 100% con la forma de ésta perdiéndose algo de precisión. Sin embargo, son bastante similares y nos dan una idea de la forma final que nos permite seguir avanzando en el proyecto.

Ahora se debe realizar un nuevo código que nos permita superponer los puntos calculados con las imágenes disponibles y así verificar que éstos son correctos. Para poder visualizarlo mejor, se transformará la imagen inicial que se encuentra en escala de grises al canal verde de la imagen RGB. Por otro lado, los puntos de intersección se trasladarán a la capa roja. De esta manera, podemos distinguir más fácilmente los pequeños puntos en la imagen y no confundirlos con el fondo ni la propia dendrita.

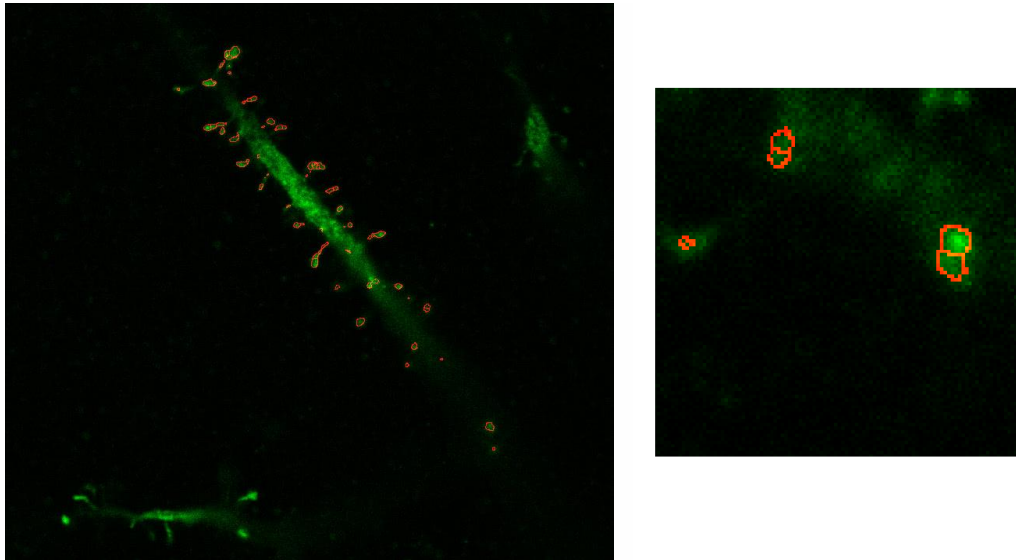


Figura 7.3. 2: Representación de los puntos obtenidos de las espinas en la imagen original.

De los resultados obtenidos, se validó el proceso utilizado hasta ahora. Cabe destacar por otro lado que, en algunos casos, en lugar de obtener un contorno más o menos circular para cada espina se obtenían círculos interiores o formas de “8” como las que se ven en la figura 7.3.2. Tras analizar estas espinas de manera individual, se descubrió que, en realidad, la forma de las espinas no es siempre regular en forma esférica sino que a veces presenta irregularidades. El resultado de la intersección de cada uno de estos picos con el plano produce contornos diferentes que, en caso de ser muy próximos producen formas como las descritas anteriormente.

A fin de que cada espina presentara una única estructura y con el fin de cerrar los contornos que pudieran quedar abiertos, se procedió a realizar un conjunto de operaciones morfológicas básicas (imclose). Esta operación, no es más que el resultado de aplicar una abertura a la imagen (conocida en inglés como *dilation*) seguida de un proceso de erosión (*erosion*). En cada una de las operaciones anteriores se pasa un filtro por la imagen y como resultado final se obtienen los objetos más o menos cerrados en función del tamaño del elemento estructural utilizado. El resultado final de esta etapa tendría el aspecto que se muestra en la figura 7.3.3 (izquierda).

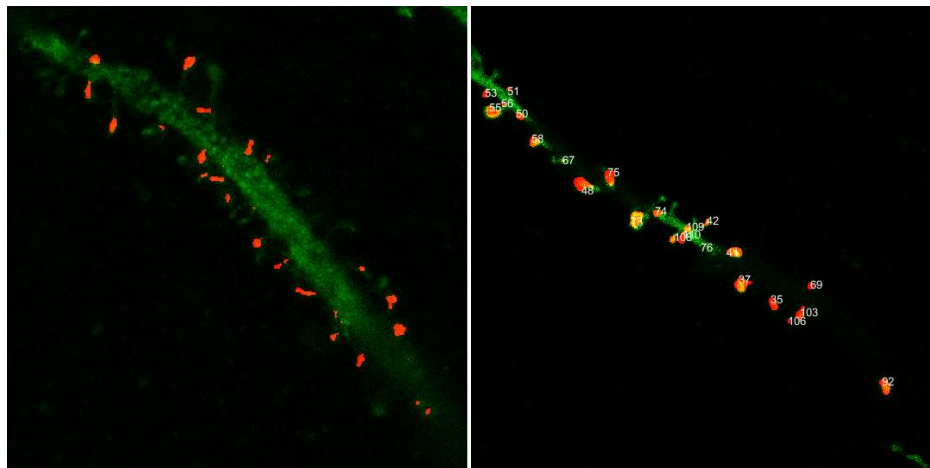


Figura 7.3. 3: Segmentación de espinas en la imagen original.

En la figura 7.3.3 (derecha), se puede apreciar que el conjunto de puntos tridimensionales de los que disponemos en los archivos y posteriormente representamos, no corresponden a la totalidad de los puntos reales de las espinas dendríticas. Esto supone que en algunas ocasiones durante la fase de pruebas, se obtengan erróneamente ventanas de inputs como no espina cuando en realidad sí lo serían. No obstante, el número de espinas de las que no disponemos información representan una minoría frente al conjunto total de espinas y no producirán grandes desviaciones para el funcionamiento normal de la red.

Una vez detectadas las espinas en la imagen, podemos dar por concluidas las dos primeras etapas cuyo objetivo era preparar e interpretar la información de la que se disponía inicialmente.

En las posteriores etapas, se utilizarán métodos de aprendizaje en base a las espinas detectadas que, junto con la información disponible y el conocimiento adquirido sobre redes neuronales, permitirán desarrollar el proyecto de una manera más sofisticada.

7.4. FASE 3: TRANSFERENCIA DE APRENDIZAJE DE UNA RED EXISTENTE

En esta nueva fase, nuestro objetivo es el de investigar y valorar los diferentes tipos de redes convolucionales que se podrían utilizar.

Llegados a este punto, construir y entrenar una nueva red desde cero requiere una gran cantidad de tiempo y no solo eso, sino que además, es un método inefectivo basado en ensayos de prueba y error que en gran medida depende del azar. Una opción más inteligente, es la de utilizar una red preexistente y reentrenarla cambiando los parámetros adecuados para la nueva aplicación o problema a resolver. Utilizando capas de la red entrenada previamente y reentrenándola con los nuevos datos se agiliza el proceso de manera considerable.

7.4.1 BASE DE DATOS MNIST

En este proyecto, la red neuronal convolucional utilizada se basará en una red ya existente. La base de datos utilizada se conoce como MNIST y está formada por imágenes de dígitos escritos a mano. Estos datos provienen originalmente de otras bases de datos NIST llamadas Special Database 1 (SD-1) y Special Database 3 (SD-3). Éstas también contienen una serie de dígitos escritos a mano por estudiantes en el primer caso y por trabajadores del censo estadounidense en el segundo caso. Inicialmente, se utilizó SD-1 para el entrenamiento de la red y posteriormente SD-3 para su testeo. Al provenir de fuentes diferentes los resultados no eran congruentes así que se optó por crear una nueva base de datos que mezclara las anteriores obteniendo así un conjunto variado de muestras para el entrenamiento y para el testeo. Es a esta nueva base de datos a la que nos referimos como MNIST. [23]

El conjunto de datos del MNIST se encuentra formado entonces por 60000 imágenes para la fase de entrenamiento (30000 del SD-1 y 30000 del SD-2) y 10000 imágenes para la fase de testeo (5000 del SD-1 y 5000 del SD-3). Asimismo, los dígitos del 0 al 9 formarán las 10 clases

disponibles para su clasificación. Con el objetivo de reconocer patrones generales estos dígitos fueron escritos por más de 250 personas diferentes para la creación de la base de datos.

Estos datos son almacenados en forma de matriz de dimensiones 28 x 28 en un cell array de dimensiones 1 x 60000 para las imágenes de entrenamiento (I), uno de 60000 x 1 que contendrá el valor real del input o ground truth (labels). Para el test se mantiene la misma estructura pero de dimensiones más reducidas: 1x10000 (I_test) y 10000 x 1 (labels_test).

7.4.2 CONFIGURACIÓN INICIAL

A continuación, se mostrarán algunas características de la red utilizada como base que será reentrenada para nuestra aplicación concreta. El autor de esta red modelo o guía es Hagay Garty y permite procesar inputs de hasta tres dimensiones [24].

Para el caso de inputs bidimensionales, la red está formada por cuatro capas principales:

Capa 1- Capa convolucional: 12 filtros de 5x5 con avance 1x1 y relleno 2x2.

Capa 2- Capa convolucional: 24 filtros de 13x13 con avance 1x1 y relleno 1x1.

Capa 3- Capa fully connected: 128 filtros de 5x5 con avance 1x1 y relleno 1x1.

Capa 4- Capa fully connected: 10 filtros de 5x5 con avance 1x1 y relleno 1x1.

Para poder utilizar esta red, se debe ajustar una serie de parámetros de la CNN original. Este proceso se conoce en inglés como fine-tuning o ajuste fino. La modificación de los parámetros adecuados permitirá que se pueda realizar una transferencia de aprendizaje y utilizar la red preentrenada para resolver un problema diferente. Habrá valores que dependan de la forma de los datos de entrada y salida de la red y deban configurarse de acuerdo con ellos. Otros, están basados en la prueba y error así como en la experiencia del programador. Algunos ejemplos serían la tasa de aprendizaje, ciertos valores iniciales y finales, el número de épocas o el número de imágenes de entrenamiento antes de evaluar la red. En los siguientes puntos, se explicará el ajuste de los parámetros que dependen de las entradas/salidas de la red y que son más objetivos.

En primer lugar, interesa ajustar el tamaño de los datos de entrada con el tamaño de las imágenes que para las que se ha configurado la red. Hay dos aproximaciones posibles: o ajustar los parámetros para que la red sea capaz de aceptar un nuevo tamaño de input/entrada. O reescalar previamente nuestras imágenes de entrada para que se ajusten al tamaño que la red tiene configurado por defecto. Aunque el segundo caso es una opción válida, puede implicar una pérdida de información en la conversión por lo que obtendremos por la primera opción modificando los parámetros de las capas pertinentes.

En segundo lugar, se deben clasificar las nuevas imágenes en solo dos categorías en función si es o no espina en lugar de en 10 salidas como los posibles dígitos del 0 al 9. Para ello, deberemos cambiar la configuración de la última capa de la red (capa de clasificación) para ajustarla a únicamente dos clases. Utilizaremos el valor 1 para definir aquellas imágenes que sí sean espina y 0 para las que no lo sean.

7.4.3 RESULTADOS DE LA RED ORIGINAL

Con esta red neuronal altamente optimizada se llega a obtener un 94% de acierto para clasificar los datos del MNIST. Este resultado se obtiene con tan solo 15 iteraciones (varios minutos). Si se somete a un mayor proceso de entrenamiento (varias horas) puede alcanzar hasta el 99.2% de éxito lo que se considera un muy buen resultado ya que, a veces, resulta difícil incluso para el propio humano distinguir un dígito de otro.

7.5 FASE 4: ENTRENAMIENTO USANDO SUBVENTANAS 2D COMO INPUT

Una vez configurada la red original para adaptarla a nuestro caso concreto, se comenzará un proceso de pruebas bidimensional para el conjunto de imágenes disponibles. El primer objetivo es obtener los inputs necesarios para poder entrenarla. Estos inputs estarán constituidos por las imágenes de entrada (I) y los valores de espina o no espina asociados a cada una de estas imágenes (labels). Es decir, se debe combinar la información calculada en las primeras fases para detectar y seleccionar las espinas de manera individual en las imágenes.

Para ello, se detectará el centroide de las espinas y se hará un recorte de la imagen original de unas dimensiones fijas de 17x17 que contendrán toda o casi toda la espina. El tamaño de la ventana es arbitrario y no debe sorprender que no sea una potencia de dos porque para calcular dicho tamaño se han cogido 8 píxeles en cada dirección además de la posición del centroide, lo que hace un total de 17 píxeles. El resultado obtenido es el siguiente:

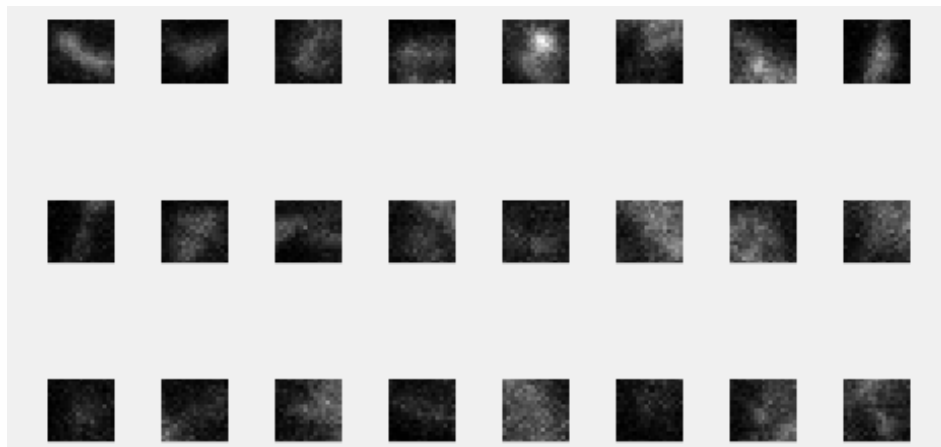


Figura 7.5. 1: Patches de la imagen original que contienen espina.

Este proceso deberá repetirse para cada una de las imágenes de las que disponemos en el directorio. Las imágenes se guardarán en forma de array y se creará un nuevo vector que contenga un 1 (hay espina) en las posiciones donde se guardan las respectivas imágenes. De esta manera, tendremos en la primera posición del array de imágenes una submatriz (pequeña porción o recorte de la imagen original) correspondiente a un trozo o la espina completa. Por otro lado, la primera posición del vector indicará con un 1 que esa primera imagen tiene información de una espina. Este proceso, también es conocido como elaboración del ground truth.

Ahora, se deben introducir también imágenes de regiones que no contengan espina. Teóricamente, si se han detectado las posiciones de las espinas, cualquier otra posición debería ser considerada como no espina. Sin embargo, en algunas de las imágenes aparecen diferentes dendritas pero solo tenemos información de una de ellas (figura 7.2.2). Esto implica que no podamos coger cualquier región al azar. Debemos asegurarnos que el patch que escogemos efectivamente no contenga ninguna espina.

Una de las posibles estrategias y que utilizaremos a partir de ahora para detectar regiones sin espina será la de desplazarnos una ventana en las cuatro direcciones espaciales básicas: arriba, abajo, derecha e izquierda. De esta manera, al estar relativamente próximos, únicamente obtendremos información cercana a la dendrita sin que corresponda a la espina de ésta.

Para realizar correctamente esta acción, deberemos tener en cuenta dos puntos fundamentales. En primer lugar, al desplazar la ventana en las respectivas direcciones no se debe, en ningún caso, desplazar la ventana más de lo que permite el límite de la imagen. Por eso, en caso de tener una espina en el límite superior e intentar coger un patch de más arriba, en lugar de cogerlo incompleto o rellenado con los valores de pad, se descartará. Por otro lado, en caso de que dos espinas estén próximas, debemos evitar que al desplazar la ventana de la espina 1 a la espina 2, ésta última quede englobada como patch de la primera y considerada entonces como no espina cuando en realidad sí lo es.

Para solventar ese problema se creará una copia de la imagen que contendrá una subventana roja (valor 1 en la primera capa de RGB) en todas las posiciones donde hayamos detectado espina. De esta manera, podemos consultar si hay o no hay espina detectada en una región antes de considerar ese patch como no espina.

Antes de mostrar los resultados finales, debemos aclarar que como no espina se consideran tanto las regiones negras (donde no hay espina) como también partes de la dendrita que no contienen espina. Estas subventanas tendrán un color más grisáceo pero no corresponden a las imágenes mostradas anteriormente de espinas dendríticas. El conjunto de patches de no espina se guardará con un valor de ground truth de 0.

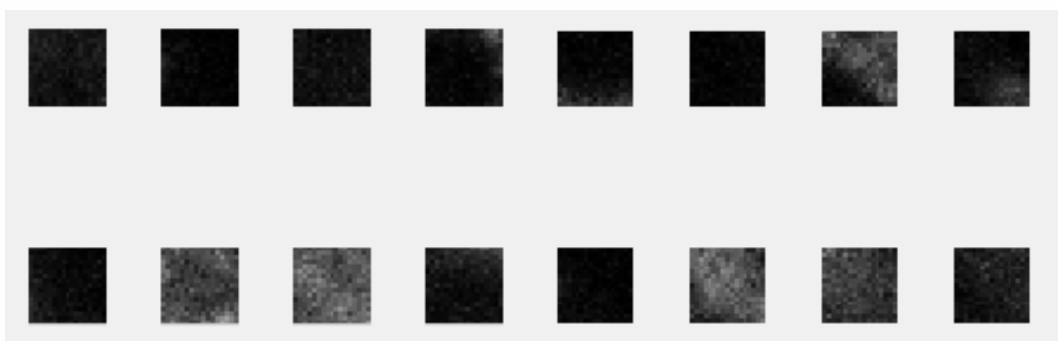


Figura 7.5. 2: Patches de la imagen original que no contienen espina.

Para poder ver el proceso de manera más visual, se propone realizar algo similar para las no espinas poniendo esas regiones detectadas como tal en color azul. De esta manera, se obtendrá una imagen final que muestre los patches elegidos como espina de color rojo y los considerados como no espina de color azul.

Finalmente, el primer resultado obtenido de este proceso en dos dimensiones para el directorio *api.if6.1.8enero*. presenta el siguiente aspecto:

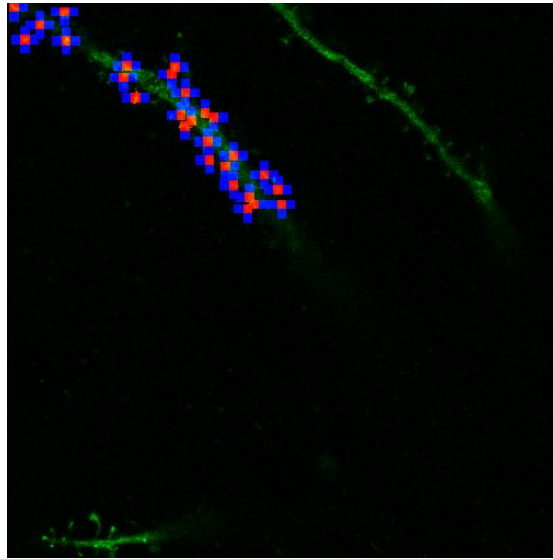


Figura 7.5. 3: Patches 17x17 de espina (rojo) y no espina (azul) de la imagen original.

Como se puede apreciar en la imagen, algunas de las regiones azules donde no hay espina pueden superponerse si ningún problema mientras que nunca encontraremos una región azul sobre una región roja que pueda inducir a confusiones en la máquina.

El mismo proceso que se ha aplicado para esta imagen concreta de buscar patches de espina y no espina deberá repetirse para el resto de imágenes disponibles para esa dendrita. Posteriormente, deberá repetirse el proceso para el conjunto de directorios.

Antes de poder entrenar la red neuronal convolucional, debemos separar adecuadamente los datos en los dos bloques mencionados a lo largo del informe: entrenamiento y testeo, más conocidos como training y testing. De manera arbitraria, se decidió utilizar un 80% de los datos para la primera fase y un 20% para la segunda que permitirían comprobar el buen funcionamiento de la red. Es importante que tanto en la primera como en la segunda fase, los datos se presenten de manera aleatoria a la red y tengamos valores tanto de espina como de no espina en entrenamiento y testeo.

Dentro de cada bloque, tendremos el correspondiente par imagen y resultado esperado. Es decir, en realidad, se dispondrá de cuatro subconjuntos de datos: el 80% de las imágenes para el entrenamiento con el 80% de los valores de ground truth y el 20% de las imágenes para el testeo con el 20% de los valores de ground truth correspondientes. En la primera fase, los valores de ground truth se utilizan para entrenar y corregir los fallos que tenga la red utilizando métodos como backpropagation y en la segunda, se utilizan solo para contrastar las salidas con los resultados reales y evaluar la precisión de la red (success rate).

Tras haber procesado toda la información de las imágenes del primer directorio, obteníamos 3304 inputs con sus respectivos labels. De estas entradas, 982 contendrían una espina y 2322 no la contendrían. Por otra parte, el 80% correspondiente a 2343 inputs se dedicarían al entrenamiento y los 661 restantes al testeo. Algunos de los resultados se muestran a continuación:

```

Iter 15 | Imgs=15000 | time=22.24 |
Finish training. max samples reached
Testing on 661 images...
success rate 87.140696%

```

Figura 7.5. 4: Resultado tras la ejecución del proceso de entrenamiento y testeo.

Una de las primeras cosas que se puede observar es el número de iteraciones. El valor 15 junto con las 15000 imágenes de input implica que realizaremos una actualización de pesos cada 1000 imágenes. La función de entrenamiento analizará todas las muestras de entrada de manera cíclica por lo que no supone ningún problema el tener un número inferior de muestras al que se utilizará para el entrenamiento. Eso sí, a mayor número de muestras diferentes, mayor precisión obtendremos en los resultados.

Por otro lado, tal y como se puede ver en la figura 7.5.4, se obtiene una tasa de éxito en el testeo de las 661 imágenes cercano al 90%. El porcentaje de acierto para cada clase se muestra con más detalle en la siguiente figura:

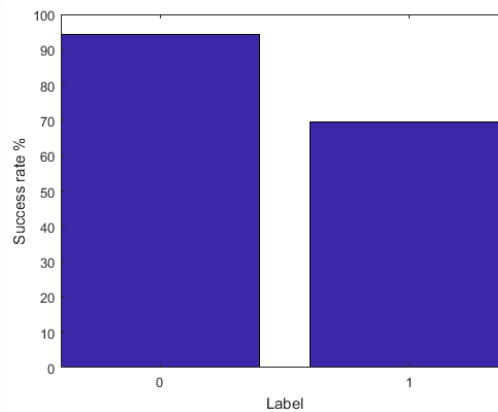


Figura 7.5. 5: Precisión en la clasificación de imágenes en los dos posibles valores de salida.

De los resultados obtenidos hasta ahora, se puede observar que el sistema detecta bastante bien las regiones que no son espinas mientras que presenta algunos fallos en la detección de espinas. Para poder estudiar mejor los casos en los que ha fallado, generaremos una serie de figuras en las que muestre el valor obtenido frente al valor real.

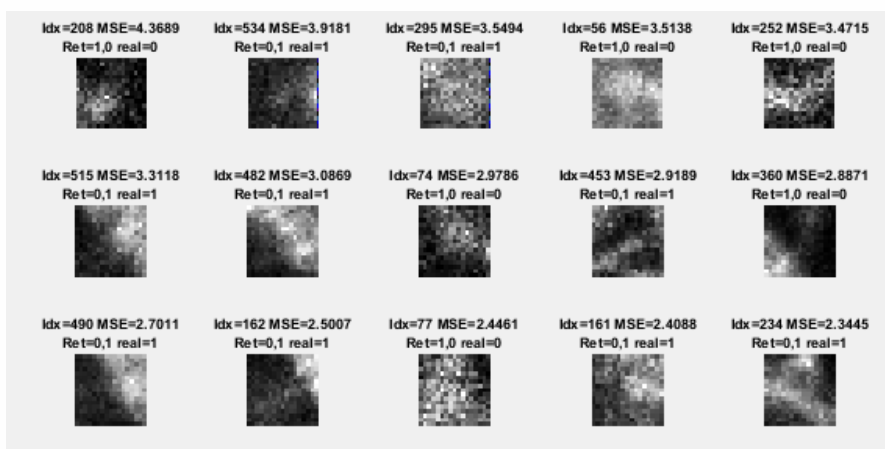


Figura 7.5. 6: Imágenes en las que el valor real no ha coincidido con el predicho.

Estas imágenes muestran los casos en los que el label y la categoría obtenida como salida no han coincidido y ,por tanto, se han detectado erróneamente. Si se analizan más detalladamente estos resultados, podemos observar que ,en algunos casos, la forma de la imagen parece corresponder a una espina real pero se ha detectado de manera errónea. Tras revisar el patch obtenido sobre la imagen original, se descubrió que la información inicial de la nube de puntos generada y proporcionada por el CSIC no correspondía a todas y cada una de las espinas que se podían distinguir en la dendrita. Sin embargo, sí se detectaban la gran mayoría lo que nos permite considerar estos casos aislados como excepciones y mantener una fiabilidad razonable de la red.

Por otro lado, se pueden observar casos en los que realmente resulta difícil distinguir a simple vista si hay o no hay dendrita. Una de las soluciones para este último problema sería obtener información de de las imágenes de las sucesivas capas para una misma posición (o ventana). De esta manera, se podría predecir el resultado con mayor precisión al contar con más información y conocer los valores previos y posteriores a esa imagen situándola así en un contexto más general. El proceso de consulta de información de capas anteriores y posteriores se utilizará en la fase siguiente y corresponde a un enfoque tridimensional del problema.

Otra forma muy útil de obtener los resultados es en forma de matriz de confusión. Esta matriz la conforman dos ejes denominados Output class (ordenadas) y Target Class (abscisas).

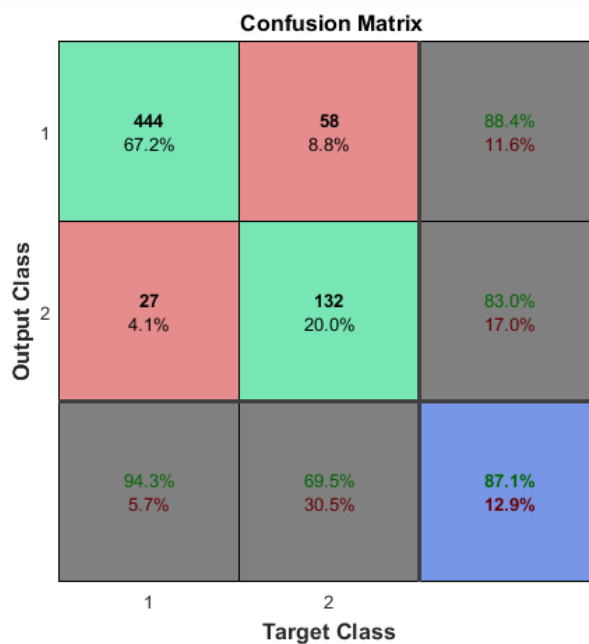


Figura 7.5. 7: Resultados para el primer directorio expresados en forma de matriz de confusión.

En esta figura, las dos primeras celdas diagonales (en verde) muestran el número y porcentaje de clasificaciones correctas de la red entrenada. Por ejemplo, 444 regiones han sido correctamente clasificadas como no espina. Esto corresponde al 67.2% del conjunto de regiones estudiadas. De manera similar, 132 regiones son correctamente clasificadas como espina lo que corresponde al 20.0% del conjunto de imágenes estudiadas.

En la casilla central de la parte superior, se ves que 58 de las regiones con espina han sido clasificadas como sin espina y ha sucedido en el 8.8% de las ocasiones. De manera análoga, 27

regiones que no eran espina han sido consideradas como espina. Este hecho se corresponde con el 4.1% de las clasificaciones totales.

Otra forma de interpretar los datos sería considerar que de las 502 predicciones de regiones de no espina, el 88.4% han sido acertadas frente al 11.6% de ocasiones erróneas. Para el caso de las zonas de espinas sucede lo mismo; de las 159 regiones que se han considerado espina, un 83% han correspondido con el valor de ground truth asociado y en el 13% de los casos no.

Si en lugar de considerar las predicciones, se compara con los valores reales que se han introducido a la red, se puede decir que; de las 471 zonas que no tenían espina un 94.3% han sido detectadas correctamente y de las 195 que sí eran espinas un 69.5% han sido detectadas exitosamente.

Todo esto en conjunto, supone una confianza total del 87.1% para clasificar de manera correcta una región dada como input de nuestra red.

Finalmente, también se pueden visualizar algunos de los filtros utilizados en las diferentes capas. Un filtro corresponde a un vector de pesos con el que realizamos la operación de convolución con la imagen. La función principal es detectar el grado de parecido que tiene esa región para la que estamos realizando la convolución con la característica (filtro) concreto. Los filtros son obtenidos directamente por la red. A continuación, se muestran algunos ejemplos:

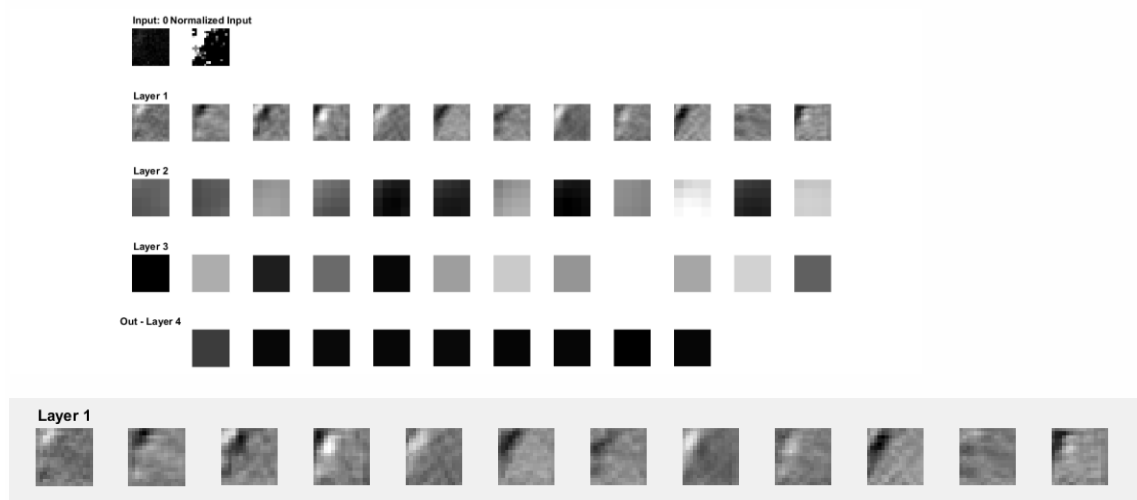


Figura 7.5. 8: Filtros obtenidos para la red entrenada con los datos del directorio *api.if6.1.8enero*.

7.6 FASE 5: ENTRENAMIENTO USANDO CUBOS 3D COMO INPUT

Tras las primeras pruebas realizadas con resultados aparentemente bastante exitosos, es hora de ir un paso más allá y aplicar el aprendizaje profundo con inputs tridimensionales. Estos inputs corresponden a las imágenes de las dendritas que se encuentran en los respectivos directorios.

El proceso es bastante similar al bidimensional ya que primero se necesitará una fase de adaptación de los datos (imágenes) y ,posteriormente, una separación en bloques de entrenamiento y testeo. Esta vez, los datos no serán patches de 17 x 17 sino que serán cubos tridimensionales.

Para poder localizar la posición de cada espina disponemos de unos nuevos archivos que contienen la información de: y_{min} , x_{min} , z_{min} , y_{max} , x_{max} , z_{max} . Existe un archivo por espina, así que si los leemos y hacemos un recorte de las imágenes originales de las dimensiones indicadas (x , y) en el número de capas (z) obtendremos las imágenes tridimensionales de esa espina.

Cada espina tiene un archivo propio asociado acorde a sus propias características. Con el fin de homogeneizar la información se decide utilizar cubos de dimensiones fijas. Para ser más precisos, estos cubos deberían contener la totalidad de la espina en las tres dimensiones. Por ese motivo, antes de realizar los recortes, se deberá leer y guardar toda la información de estos cubos y detectar las dimensiones máximas del conjunto de directorios que tratemos.

Una vez obtenidos estos máximos, deberemos ajustar los nuevos valores de x , y , z para que se ajusten al tamaño de cubo máximo y no solo al de su propia espina. Es decir, si el tamaño máximo que resulta finalmente fuera de 20 en todas las dimensiones y el leído fuera de 10, deberíamos aumentar 5 unidades nuestras dimensiones en todas las direcciones.

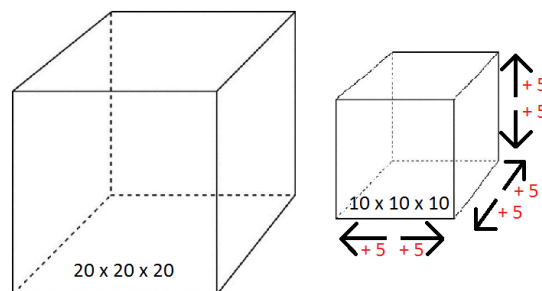


Figura 7.6. 1: Proceso de homogenización y creación del nuevo cubo.

En caso de tener dimensiones diferentes, se ajustará cada dimensión por separado manteniendo siempre la espina en el centro del nuevo cubo. Después de leer y ajustar esta nueva información, se guardará el nuevo cubo de la espina para futuras operaciones donde conocer las posiciones que contengan espina sea necesario.

Realizando un análisis de todas estas dimensiones en los 8 primeros directorios, obtuvimos unas dimensiones de cubo máximo de $48 \times 36 \times 12$. Con estas dimensiones, se obtendría la información de cada espina en su totalidad. Este tamaño de cubo está programado para ajustarse en función de los directorios que se utilicen. La única modificación adicional deberá realizarse en el archivo `config_mnist3d` para indicar a la red el nuevo tamaño de los datos.

De nuevo, estos nuevos cubos deben ajustarse al tamaño de la imagen. Por lo tanto, si aumentamos las dimensiones de un cubo que se encuentra en algún límite de la imagen y no puede adquirir las dimensiones adecuadas, no será considerado en la adquisición de datos.

Para la visualización tridimensional de las imágenes que conforman la espina utilizaremos una función que no integra Matlab y funciona como una GIU (Grafic Interface Unit). A continuación se muestran los resultados tanto para los cubos que contienen espina como para los que no.

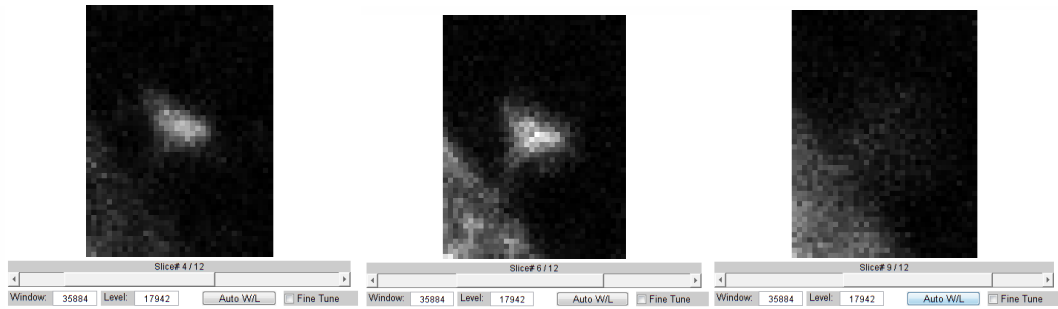


Figura 7.6. 2: Muestra de las diferentes capas del cubo que contienen información tridimensional de la espina.



Figura 7.6. 3: Muestra de las diferentes capas del cubo que no contienen espina.

Los cubos tridimensionales que no tienen espina también se han generado siguiendo la lógica utilizada en dos dimensiones. Por un lado, no deben exceder las dimensiones de la imagen y por otro, no deben superponerse cubos de no espina con cubos de espina.

A diferencia del caso bidimensional, para definir una zona como no espina se debe comprobar que en esa región no aparece ningún fragmento de espina **para ninguna de las capas o imágenes** que conforman el cubo. Es decir, se deben calcular todos los cubos de las espinas para todas las imágenes del directorio y solo entonces repetir el proceso para las zonas de no espina. Nótese que para el caso bidimensional, era posible detectar espinas y no espinas de cada imagen por separado.

Cabe destacar que en este caso tridimensional, es menos probable encontrar cubos en las cuatro direcciones que no contengan ninguna capa de espina. Por ejemplo, para el primer directorio, solo encontramos 10 cubos de no espina lo que supone un número muy reducido de muestras y con el que no obtendremos los mejores resultados posibles. Por ello, se decide ampliar el rango y utilizar las cuatro regiones diagonales disponibles como regiones de no espina siempre y cuando no coincidan con los puntos donde sabemos que sí hay. Con este pequeño cambio, conseguimos sextuplicar el número de muestras de regiones sin espina que combinada con la información del resto de directorios generarán suficientes datos para entrenar la red adecuadamente.

De la misma forma que en la etapa cuatro, se identificarán las capas de los cubos con espina de color rojo y las de no espina de color azul. Como ya se ha realizado en puntos anteriores, para representar la información tridimensional se presentará una muestra de las capas bidimensionales del cubo.

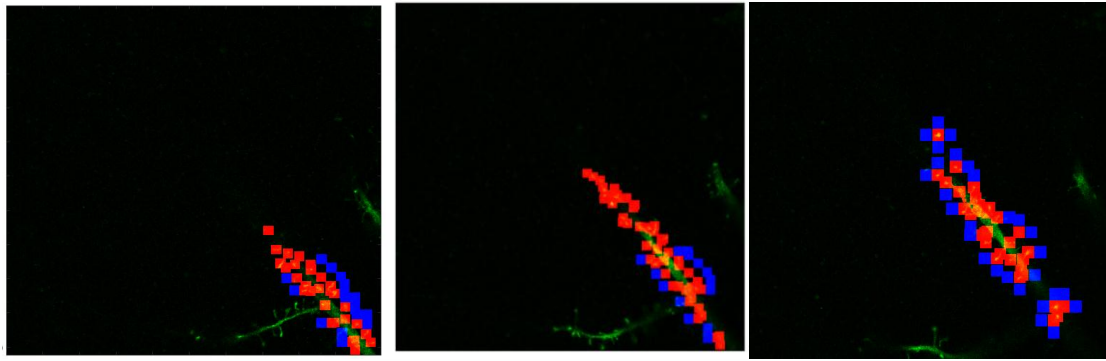


Figura 7.6. 4: Imágenes extraídas durante el proceso de detección de cubos de espinas (rojo) y no espinas (azul).

La información de cada subventana se guardará de la misma forma que anteriormente aunque ahora se tratara con matrices tridimensionales (cubos). El ground truth se mantiene igual; con valores de 1 para espina y 0 para no espina asociados a cada cubo del array de matrices tridimensionales.

Finalmente, los datos se organizarán del mismo modo que en la fase anterior dedicando un 80% de las imágenes y labels a la fase de entrenamiento y el resto para el testeo. El orden de los datos de imágenes y labels se dará de manera aleatoria compartiendo la misma semilla para que cada imagen mantenga asociado el label correcto en todo momento.

Los resultados de la red neuronal una vez entrenada, superan el 90% y se mostrarán más detalladamente en el capítulo de pruebas. Ahora, resultaría más interesante estudiar con qué tipo de imágenes se está alimentando la red para saber si cumplimos con el objetivo principal. Los datos que se presentan a continuación muestran una visión tridimensional del input a la izquierda y una bidimensional del conjunto de capas a partir del que se ha generado:

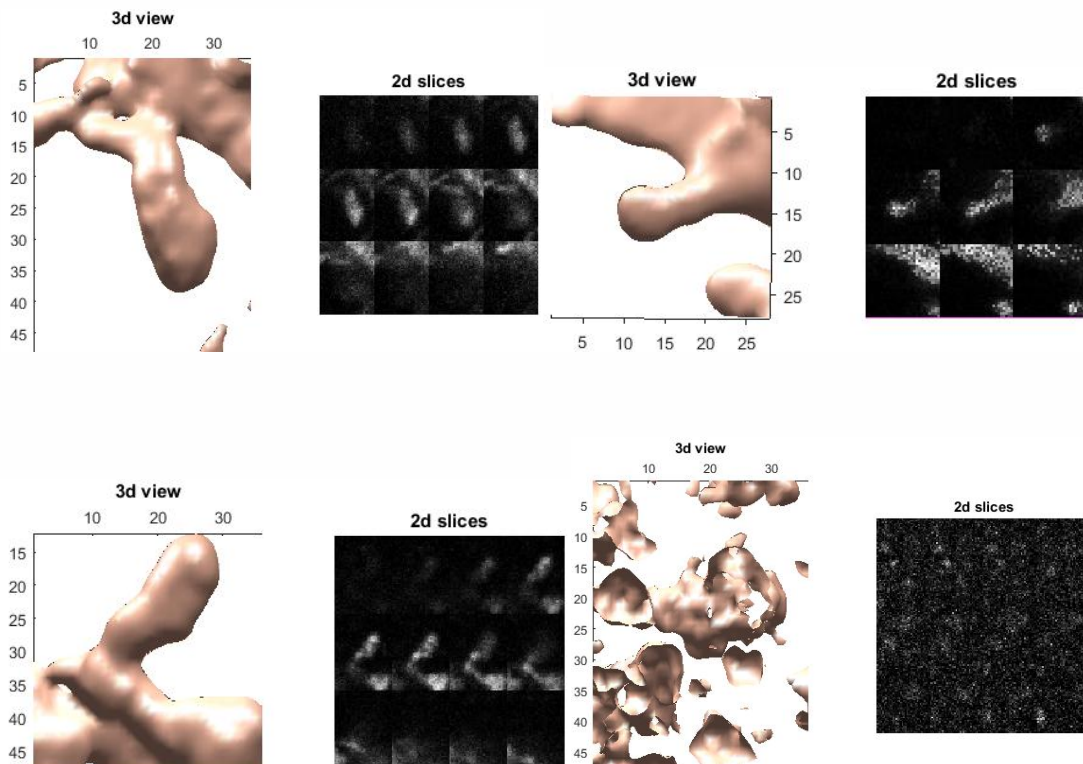


Figura 7.6. 5: Muestra de la vista tridimensional de tres espinas y una región de no espina.

8. CASOS EXPERIMENTALES CONSIDERADOS

En este capítulo se recogerán el conjunto de pruebas y experimentos realizados para las imágenes del CSIC de las dendritas neuronales. El fin último es el de validar el funcionamiento de la red que dependerá en gran medida del proceso de entrenamiento seguido. Por este motivo, se probarán diferentes tamaños de imagen, diversos directorios y diferentes parámetros para ver con qué método se obtienen los mejores resultados. Se debe comentar que la selección de las imágenes, junto con el valor de *ground truth* correspondiente, es posiblemente uno de los puntos clave del proyecto y de los que dependerán en gran medida la correcta detección de espinas dendríticas.

8.1 DATASETS

El conjunto de todas las imágenes de los diferentes datasets presentan las mismas dimensiones siendo éstas de 1024x1024. Las imágenes de cada uno de los directorios describen una misma región de corteza cerebral a diferentes alturas. Sin embargo, las imágenes de un directorio y otro no se corresponden con la misma zona. Para el proceso de pruebas se utilizará desde uno hasta múltiples directorios. Las diferentes agrupaciones de las imágenes disponibles determinarán los distintos datasets:

Dataset1: el conjunto de imágenes que forman el dataset1 corresponde a las imágenes de microscopía del directorio api.if6.1.8enero.

Dataset2: el conjunto de imágenes que forman el dataset2 corresponde a las imágenes de microscopía del directorio api.m16.1.9-1enero.

Dataset3: el conjunto de imágenes que forman el dataset2 corresponde a las imágenes de microscopía del directorio if6.1.1-1enero.

Dataset4: el conjunto de imágenes que forman el dataset2 corresponde a las imágenes de microscopía del directorio if6.1.2-2enero.

Dataset5: el conjunto de imágenes que forman el dataset2 corresponde a las imágenes de microscopía de los directorios api.if6.1.8enero, api.m16.1.5-1enero, api.m16.1.9-1enero, if6.1.1-1enero, if6.1.1-2enero, if6.1.2-1enero, if6.1.2-2enero, if6.1.2-3enero e if6.1.3-1enero.

Dataset6: el conjunto de imágenes que forman el dataset2 corresponde a las imágenes de microscopía del directorio if6.1.3-2enero.

Dataset7: el conjunto de imágenes que forman el dataset2 corresponde a las imágenes de microscopía del directorio if6.2.0-1enero.

8.2 DIMENSIONES

Otro punto con los que se ha experimentado es con el número de dimensiones en la fase de entrenamiento. Para la fase bidimensional, el conjunto de entradas corresponde a subventanas o imágenes mientras que, para la fase tridimensional, los inputs tienen una forma de cubo siguiendo estructura de matriz tridimensional.

Por otro lado, otro de los factores diferenciales en los diferentes entrenamientos es el tamaño de esa ventana o cubo que ha ido variándose a lo largo de las pruebas.

Para el caso de dos dimensiones se han utilizado tamaños de ventana de 17x17. Posteriormente, se han realizado pruebas con un tamaño de 33x33. El criterio de selección de la ventana es arbitrario. El motivo de que la ventana sea 17x17 y no 16x16 es que se han cogido 8 píxeles a derecha, izquierda, arriba y abajo desde el píxel central. De esta manera, la ventana será simétrica y tendrá una anchura y altura total de 8+píxel central+8, es decir, 17 píxeles. El mismo razonamiento se aplicará para la ventana de 33x33 cogiendo esta vez 16 píxeles en las diferentes direcciones.

Para el caso de tres dimensiones, también se han elegido dos tamaños de cubo diferentes. La primera prueba se realiza con un tamaño de 28x28xvariable. Los primeros dos números corresponden a la altura y anchura del cubo. El último denominado como “variable” corresponderá a la profundidad máxima de la mayor de todas las espinas de los directorios que analicemos. De este modo, se puede asegurar que siempre se toman todas las capas de la espina en cuestión y, en los casos en los que la profundidad de la espina no sea la máxima, también se obtendrá información de las capas previas y posteriores. Siguiendo el razonamiento de esta última dimensión de profundidad variable, se generará también una prueba con todas las dimensiones variables, “variable x variable x variable”. Por lo tanto, el tamaño de esta ventana siempre se ajustará a todas y cada una de las espinas ya que tendrá las mismas dimensiones máximas que la espina más alta, la espina más ancha y la espina más profunda. Aunque ya se comentará en la prueba correspondiente, para los nueve primeros directorios este tamaño de ventana será finalmente de 48x46x29.

8.3 DESCRIPCIÓN DE LAS PRUEBAS REALIZADAS

En este apartado, se mostrarán las pruebas realizadas para las que se han utilizado el conjunto de datasets explicados anteriormente.

Se entrenarán 6 redes diferentes mostrando los outputs obtenidos con información sobre su precisión, filtros utilizados, errores y matriz de confusión. Posteriormente, se realizarán una serie de pruebas comparando el valor que obtenemos de la red al introducirle un input concreto frente al valor real conocido. De esta manera, al utilizar nuevos directorios con los que la red no ha sido entrenada podremos ver la precisión real de la red y recoger estos datos

en forma de tablas y matrices de confusión donde se muestren no solo los resultados acertados sino también información sobre los errores como falsos positivos o falsos negativos.

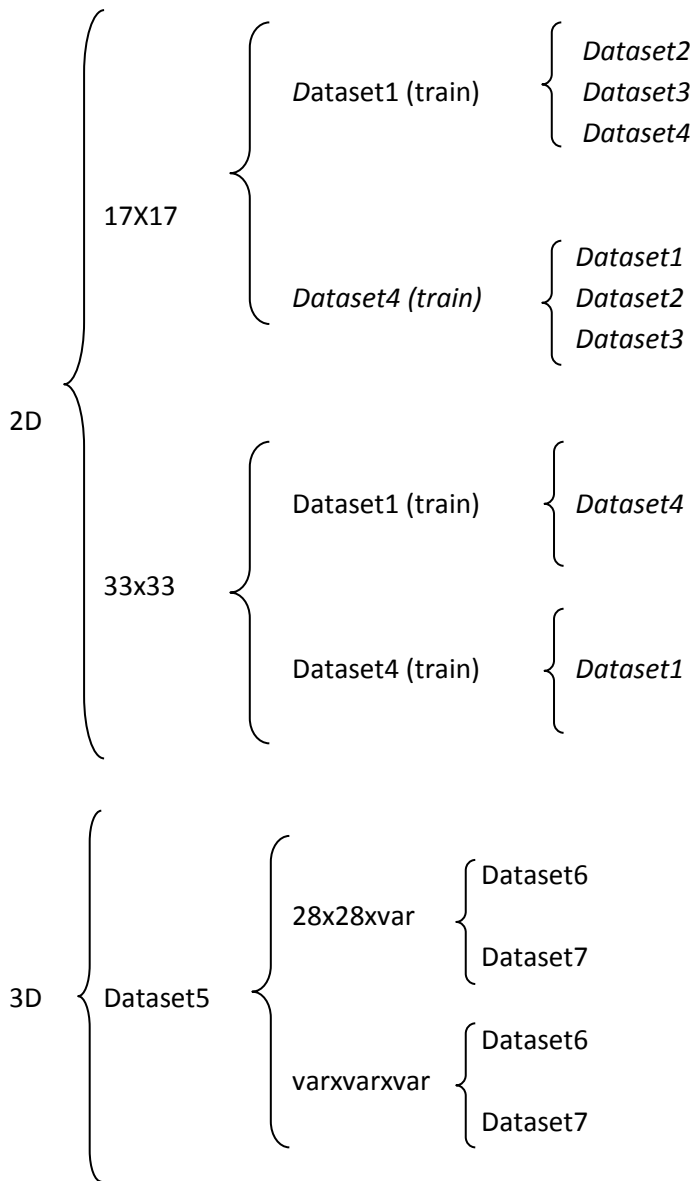


Figura 8.2: Esquema de los diferentes escenarios experimentales.

Para el caso bidimensional, además del método de segmentación de espinas basado en los puntos de intersección calculados se probó a realizar una selección de ventanas 16×16 con un proceso similar al utilizado en el caso tridimensional. Consistía en leer los ficheros que contienen la información tridimensional de cada espina, adaptar las dimensiones x e y para ajustarse a un tamaño fijo de ventana e ir seleccionando los conjuntos correspondientes para cada plano. No obstante, en algunas capas para las que se seleccionaba la espina (especialmente las primeras y últimas) no obtenía información relevante de la espina y no eran adecuadas como imágenes válidas para entrenar la red. Además, utilizando el primer método nos aseguramos que la espina esté centrada en todo momento contribuyendo a una mejor calidad de los datos de entrada para los que la red será entrenada.

9. RESULTADOS EXPERIMENTALES

Una vez definido el conjunto de datos y pruebas con los que se realizarán los diferentes escenarios experimentales, se describirá muy brevemente el procedimiento seguido en cada uno de ellos. En primer lugar, se describirá el número de inputs con el que se ha entrenado la red y los resultados que se obtienen tras el proceso de entrenamiento. Los resultados obtenidos tras el proceso de entrenamiento mostrarán información de la red sobre su porcentaje de acierto, el número de fallos obtenido para cada clase en forma de matriz de confusión, alguno de los filtros utilizados y algunos inputs clasificados erróneamente. En segundo lugar, se testeará la red con una selección de imágenes de otros directorios mostrando la precisión y matrices de confusión obtenidas.

Por lo tanto, en este apartado se incluyen los resultados de manera objetiva de las diferentes pruebas previamente descritas. El conjunto de resultados será interpretado en el apartado de conclusiones por lo que únicamente mostrarán los valores obtenidos de las ejecuciones realizadas y los datos para los que se han generado.

9.1 RED BIDIMENSIONAL PARA EL DATASET1 CON VENTANAS 17X17

Si se ejecuta el código de generación de los puntos de intersección y posteriormente se segmentan las espinas en sus respectivas ventanas de dimensiones 17x17 se obtiene un conjunto de 3304 imágenes totales de zonas de espina y no espina. El número de ventanas que sí tienen espina corresponde a 982 de las 3304 imágenes y se identificarán con el valor de etiqueta (*label*) igual a la unidad. Del conjunto total de imágenes, un 80% se dedicará al entrenamiento (2643) y el 20% restante a la fase de testeo (661). Los resultados obtenidos son:

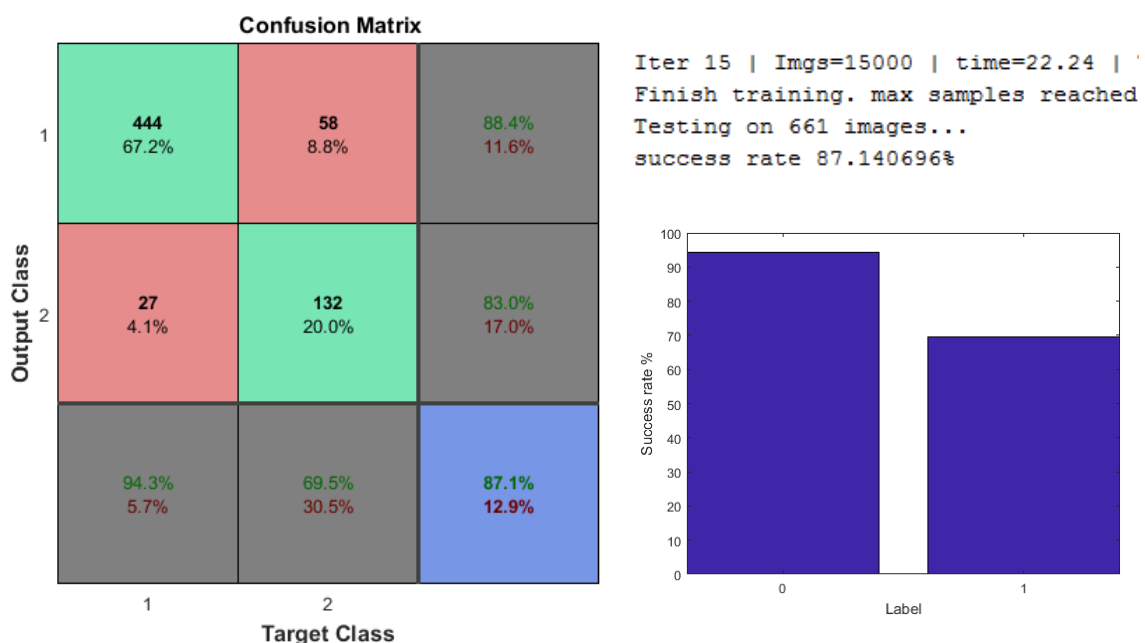


Figura 9.1. 1: Matriz de confusión, ratio de acierto y confianza de la primera red.

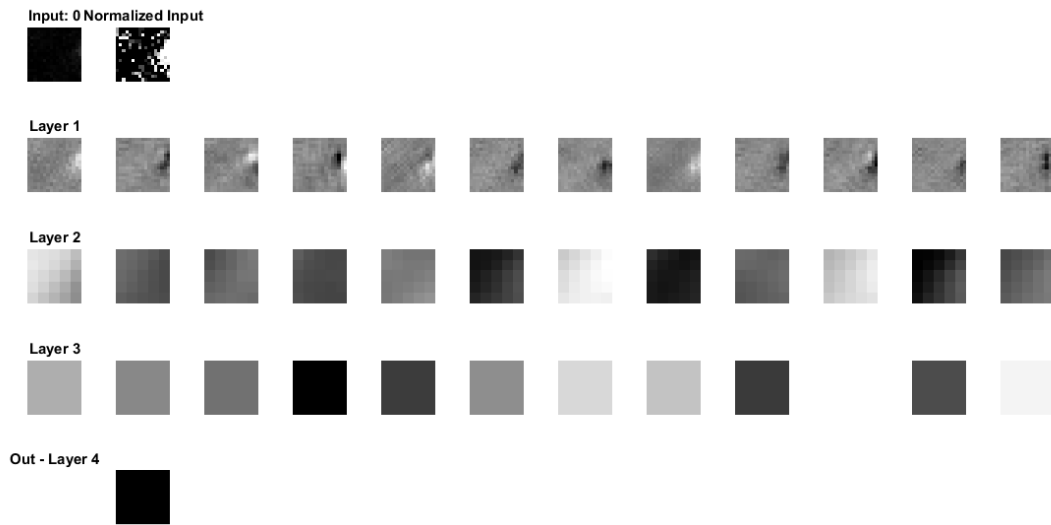


Figura 9.1.2: Filtros de la primera red una vez entrenada.

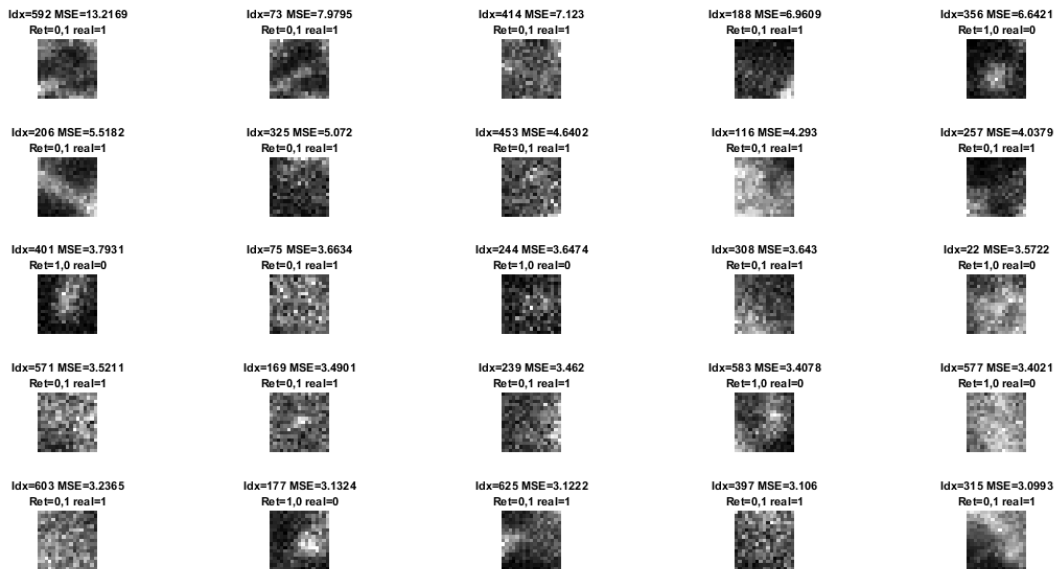


Figura 9.1.3: Errores durante el entrenamiento para la primera red

A continuación, se presentan las matrices de confusión correspondientes a las pruebas realizadas para la primera red entrenada con ventanas de dimensiones 17x17 de las imágenes del dataset 1.



Figura 9.1.4: Matrices de confusión de la primera red testeada con dataset2 (izquierda), dataset3 (derecha) y dataset4 (abajo).

9.2 RED BIDIMENSIONAL PARA EL DATASET4 CON VENTANAS 17X17

Si se ejecuta el código de generación de los puntos de intersección y posteriormente se segmentan las espinas en sus respectivas ventanas de dimensiones 17x17 se obtiene un conjunto de 1811 imágenes totales de zonas de espina y no espina. El número de ventanas que sí tienen espina corresponde a 556 de las 1811 imágenes. Del conjunto total de imágenes, un 80% se dedicará al entrenamiento (1448) y el 20% restante a la fase de testeo (363). Los resultados obtenidos son:

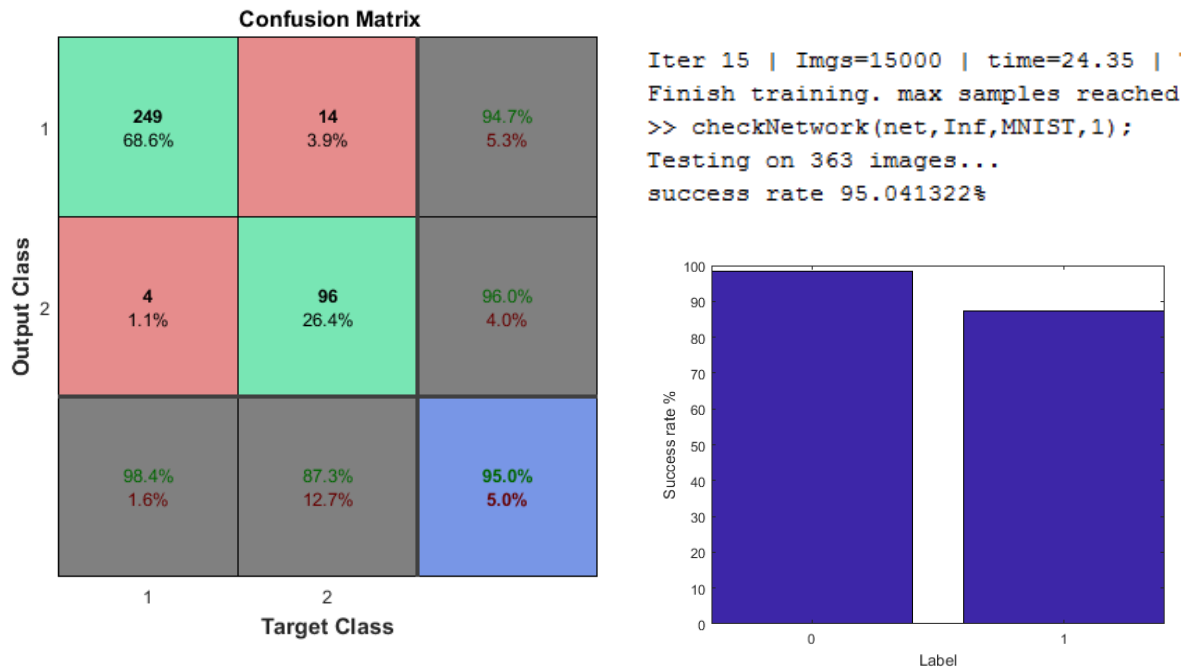


Figura 9.2. 1: Matriz de confusión, ratio de acierto y confianza de la segunda red.

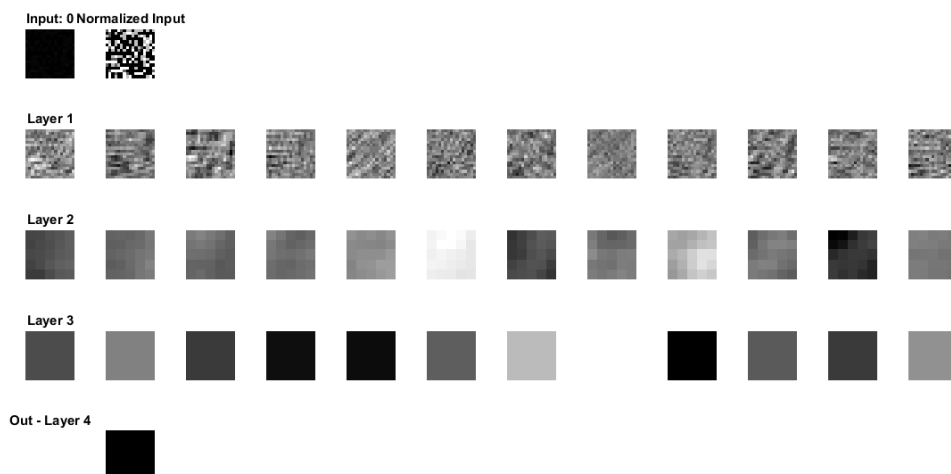


Figura 9.2. 2: Filtros de la segunda red una vez entrenada.

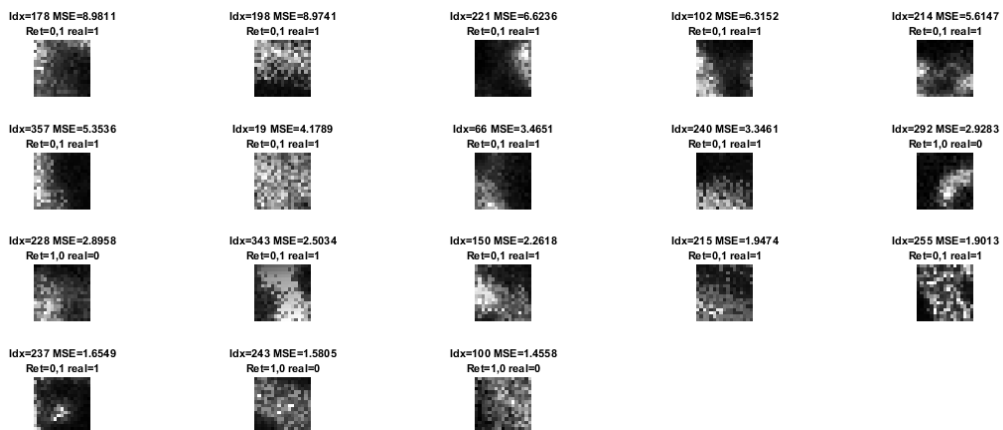


Figura 9.2. 3: Errores durante el entrenamiento para la segunda red

A continuación, se presentan las matrices de confusión correspondientes a las pruebas realizadas para la segunda red entrenada con ventanas de dimensiones 17x17 de las imágenes del dataset4.



Figura 9.2. 4: Matrices de confusión de la segunda red testeada con dataset1 (izquierda), dataset2 (derecha) y dataset3 (abajo).

9.3 RED BIDIMENSIONAL PARA EL DATASET1 CON VENTANAS 33X33

En este apartado se obtendrá un conjunto de 2043 imágenes totales de zona de espina y no espina de dimensiones 33x33. De las 2043 imágenes totales, 982 corresponden a espinas identificadas. Es normal que el número de espinas coincida con el primero de los experimentos, pues nos encontramos en el mismo directorio. Sin embargo, al modificar el tamaño de ventana encontramos un número diferente de imágenes totales debido a menor cantidad de ventanas de no espina. Como en los casos anteriores, del conjunto total de imágenes, un 80% se dedicará al entrenamiento (1634) y el 20% restante a la fase de testeo (409). Los resultados obtenidos se muestran a continuación:

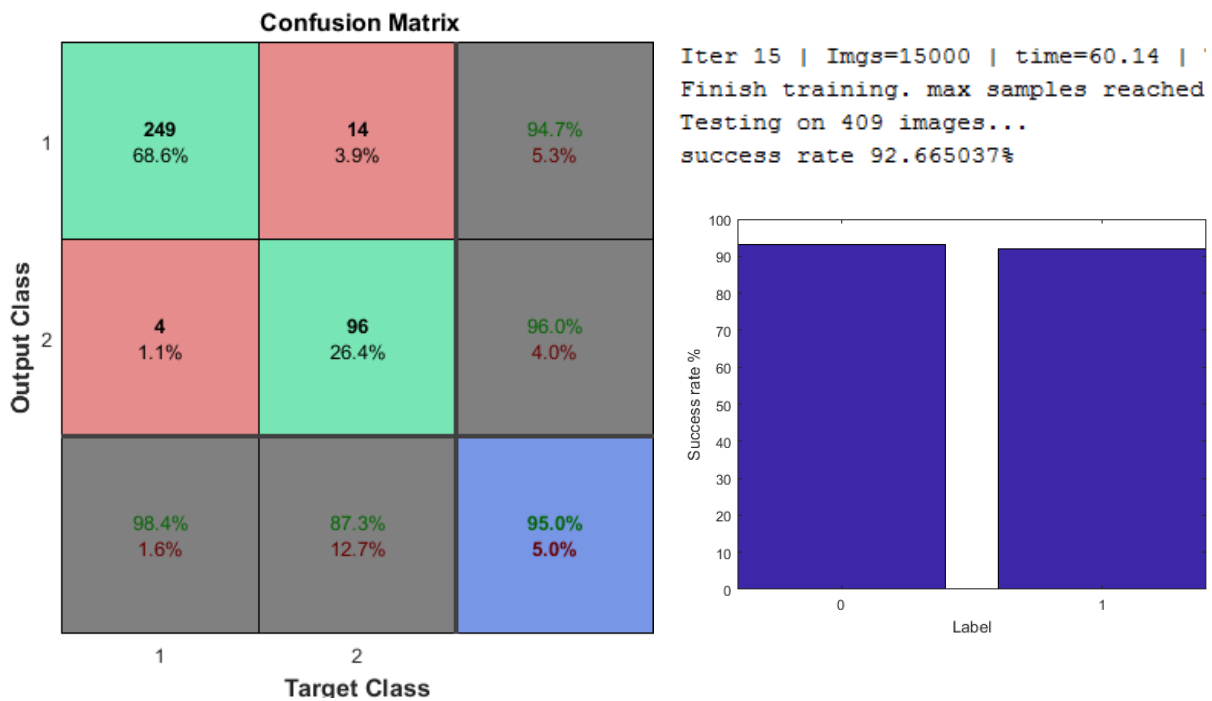


Figura 9.3. 1: Matriz de confusión, ratio de acierto y confianza de la tercera red.

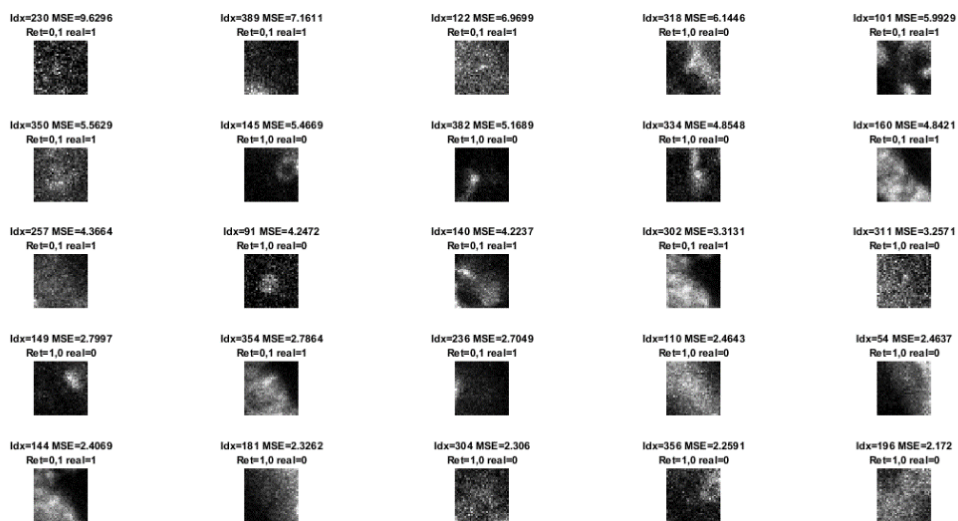


Figura 9.3. 2: Errores durante el entrenamiento para la tercera red.

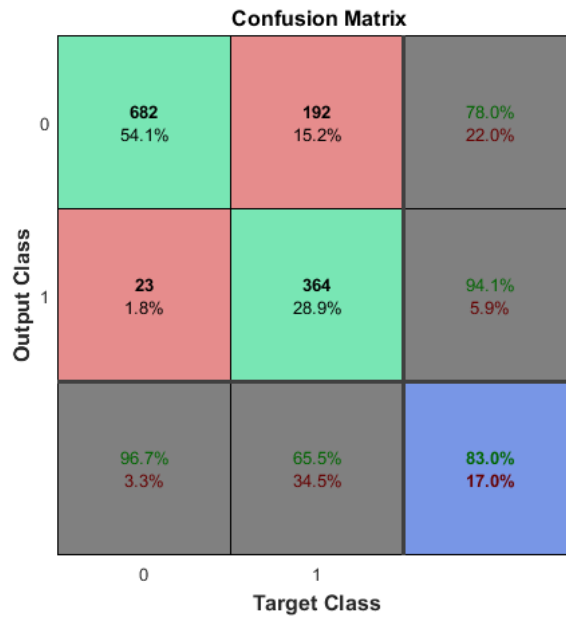


Figura 9.3. 3: Matriz de confusión de la tercera red testeada con dataset4.

9.4 RED BIDIMENSIONAL PARA EL DATASET4 CON VENTANAS 33X33

En este apartado se obtendrá un conjunto de 1261 imágenes totales de zona de espina y no espina de dimensiones 33x33. Del conjunto anterior, 556 corresponden a espinas identificadas y el resto a regiones de dendrita u otras áreas sin espina. Como en los casos anteriores, un 80% se dedicará al entrenamiento (1008) y el 20% restante a la fase de testeo (253). Los resultados obtenidos son:

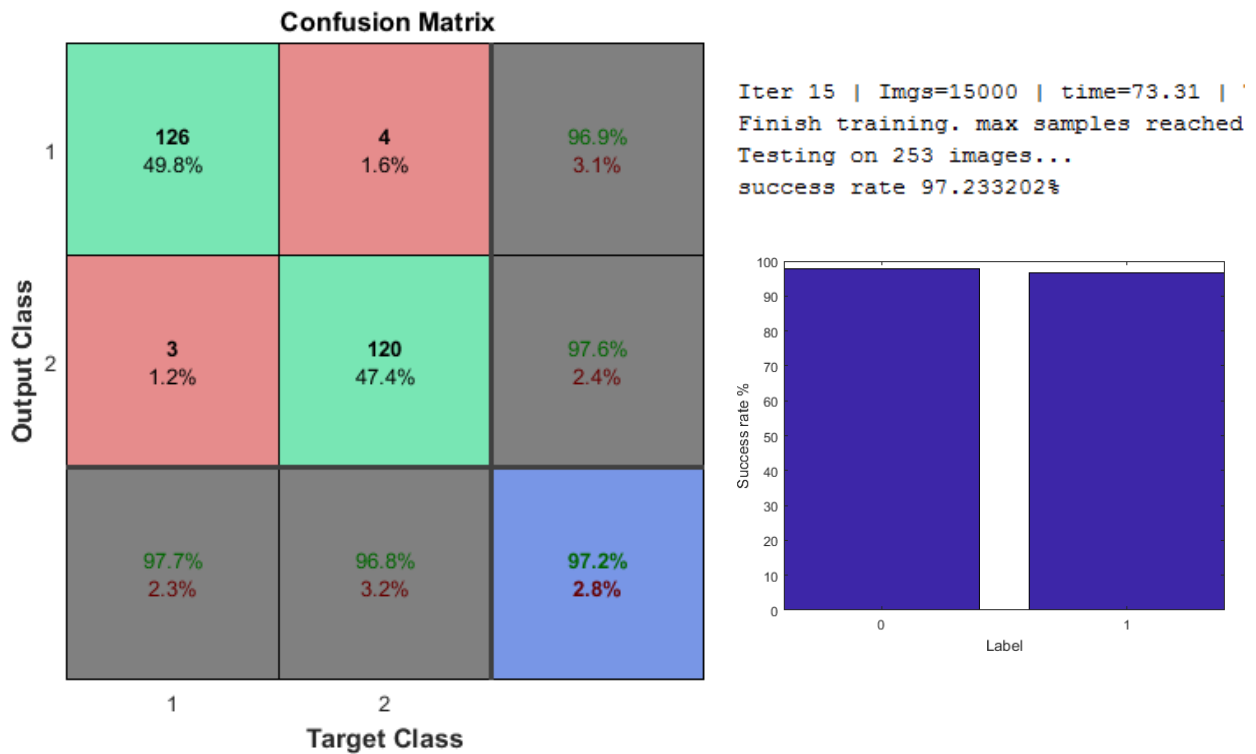


Figura 9.4. 1: Matriz de confusión, ratio de acierto y confianza de la cuarta red.

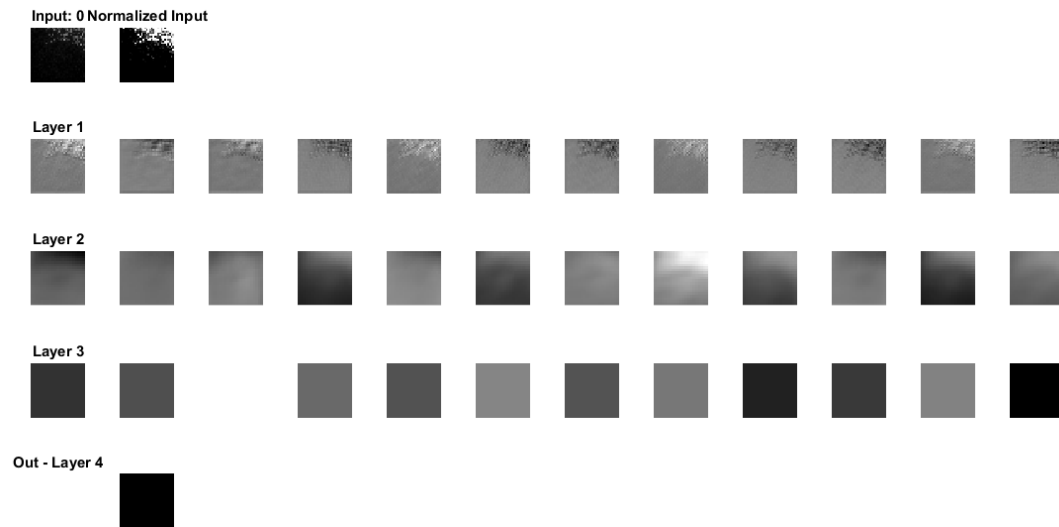


Figura 9.4. 2: FilTROS de la cuarta red una vez entrenada.

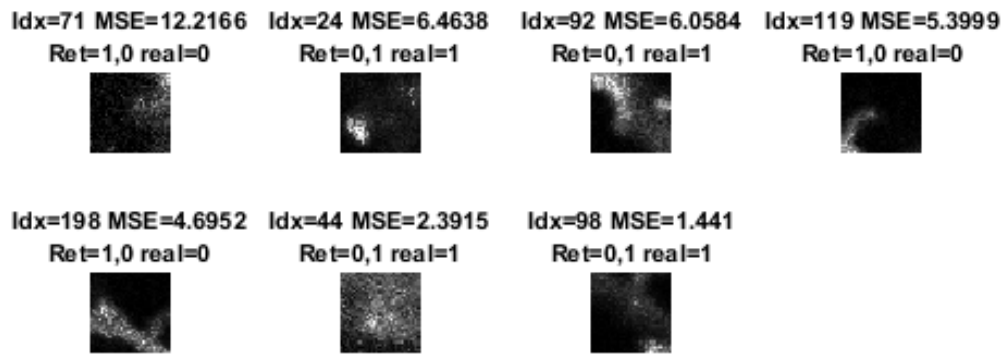


Figura 9.4. 3: Errores durante el entrenamiento para la cuarta red.

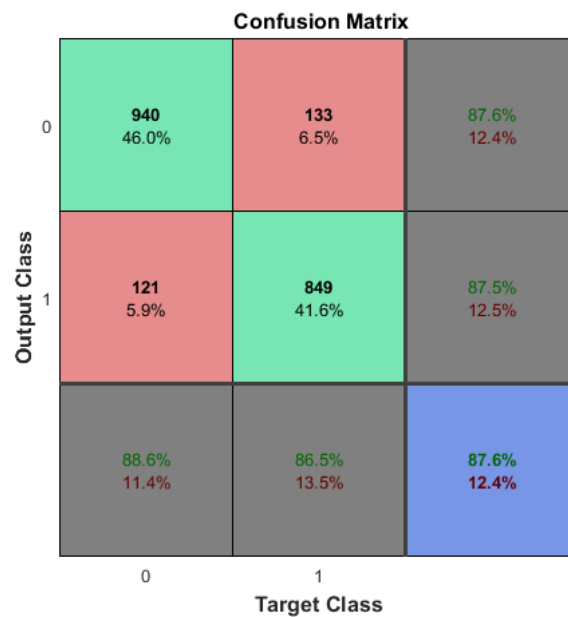
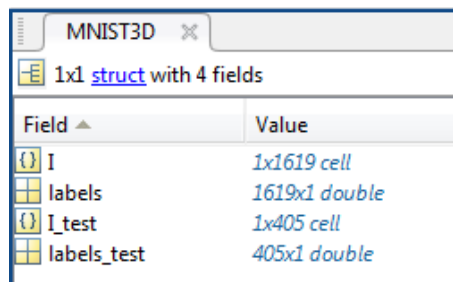


Figura 9.4. 4: Matriz de confusión de la cuarta red testeada con dataset1.

9.5 RED TRIDIMENSIONAL PARA EL DATASET5 CON VENTANAS 28x28xvar

Para entrenar la red tridimensional, se utilizarán cubos de dimensiones 28x28xvar obtenidos directamente de las imágenes de microscopía. Como se ha comentado en el capítulo 7 de desarrollo del proyecto, los cubos generados de no espina no contendrán ninguna región de espina en él para no confundir a la máquina. Además, los cubos generados en las diferentes direcciones que rodean a la espina no deben exceder del tamaño de la imagen. Por esos dos motivos, el número de muestras para entrenar la red se reduce considerablemente y no dispondremos de muestras suficientes si continuamos con el método utilizado en dos dimensiones. Para solventar esa falta de pruebas se realizarán dos acciones; por un lado, se aumentará el número de direcciones en las que se toman los cubos de no espina (añadiendo también diagonales) y, por otro lado, se tomarán muestras de hasta un total de 9 directorios.

El conjunto total de imágenes obtenidos es de 2024 de dimensiones 28x28x29, de las cuales 936 contendrán información de espinas dendríticas. Como en los casos anteriores, un 80% se dedicará al entrenamiento (1619) y el 20% restante a la fase de testeo (405):



The screenshot shows a MATLAB workspace window titled 'MNIST3D'. It displays a 1x1 struct with 4 fields:

Field	Value
I	1x1619 cell
labels	1619x1 double
I_test	1x405 cell
labels_test	405x1 double

9.5. 1: Datos entrenamiento y testeo de la red.

Los resultados de la ratio de acierto, matriz de confusión y errores se presentan a continuación:

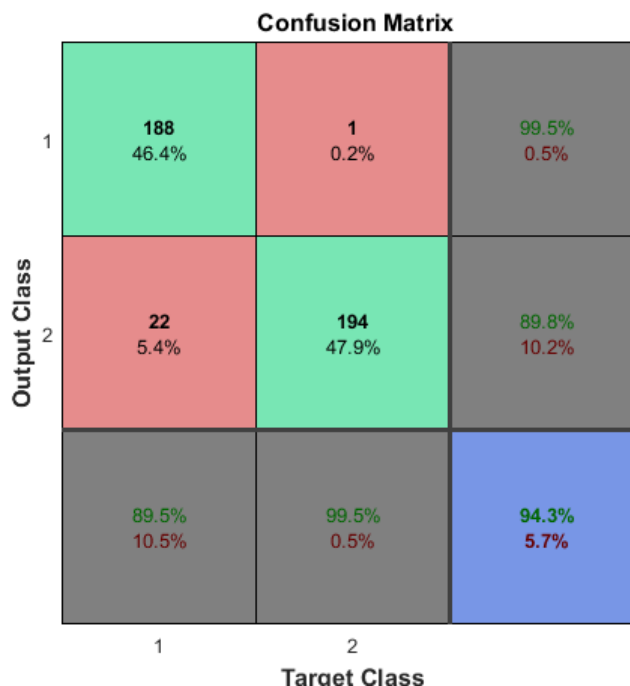


Figura 9.5. 2: Matriz de confusión de la quinta red.

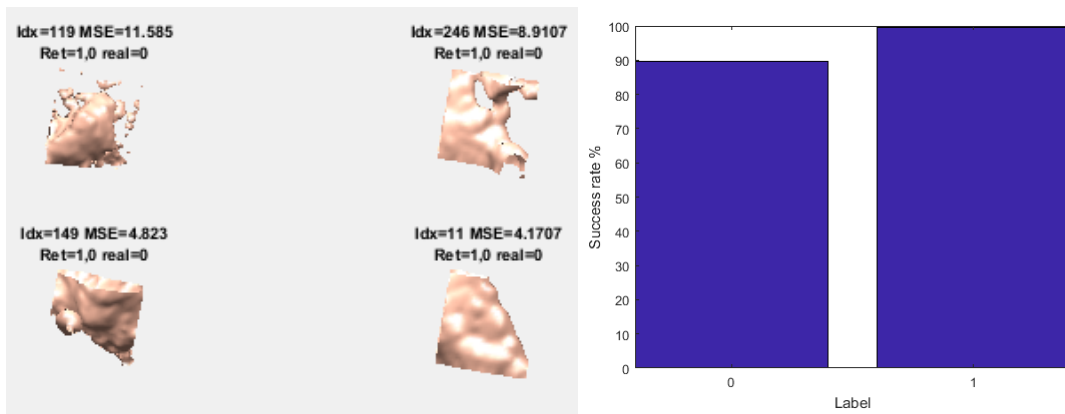
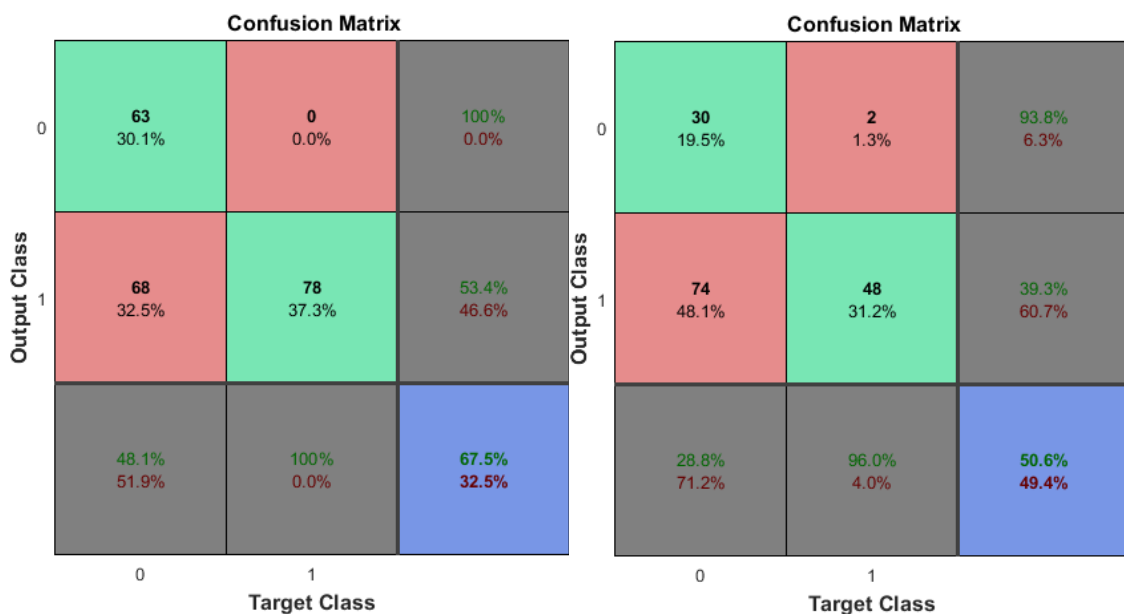


Figura 9.5.3: Errores y ratio de acierto de la quinta red.



9.5. 3: Matrices de confusión de la quinta red con dataset6 (izquierda) y dataset7 (derecha).

9.6 RED TRIDIMENSIONAL PARA EL DATASET5 CON VENTANAS MÁXIMAS.

Para entrenar la red tridimensional, se utilizarán cubos de dimensiones var x var x var. Para el conjunto de datos sobre los que se entrena la red, correspondiente a los 9 directorios que componen el dataset5, estas dimensiones máximas alcanzan los valores 48 x 46 x 29 generando así un cubo que contendrá cada espina en su totalidad.

El conjunto total de imágenes obtenidos es de 1667 de dimensiones var x var x var, de las cuales 924 contendrán información de espinas dendríticas. Como en los casos anteriores, un 80% se dedicará al entrenamiento (1341) y el 20% restante a la fase de testeo (336). El conjunto de pruebas realizada para esta última red se muestra a continuación:

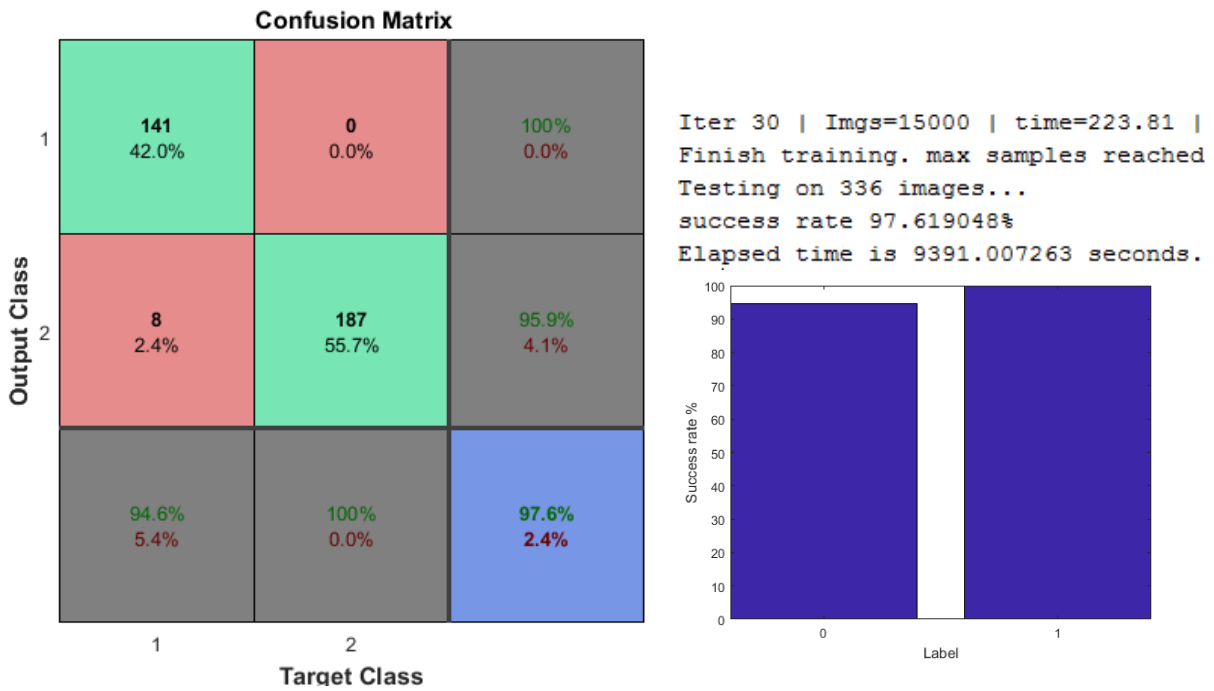


Figura 9. 6. 1: Matriz de confusión, ratio de acierto y confianza de la sexta red.

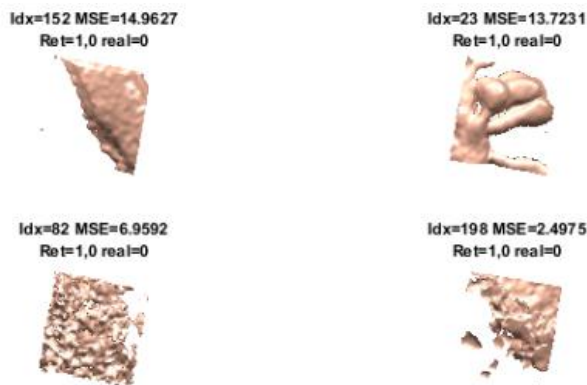
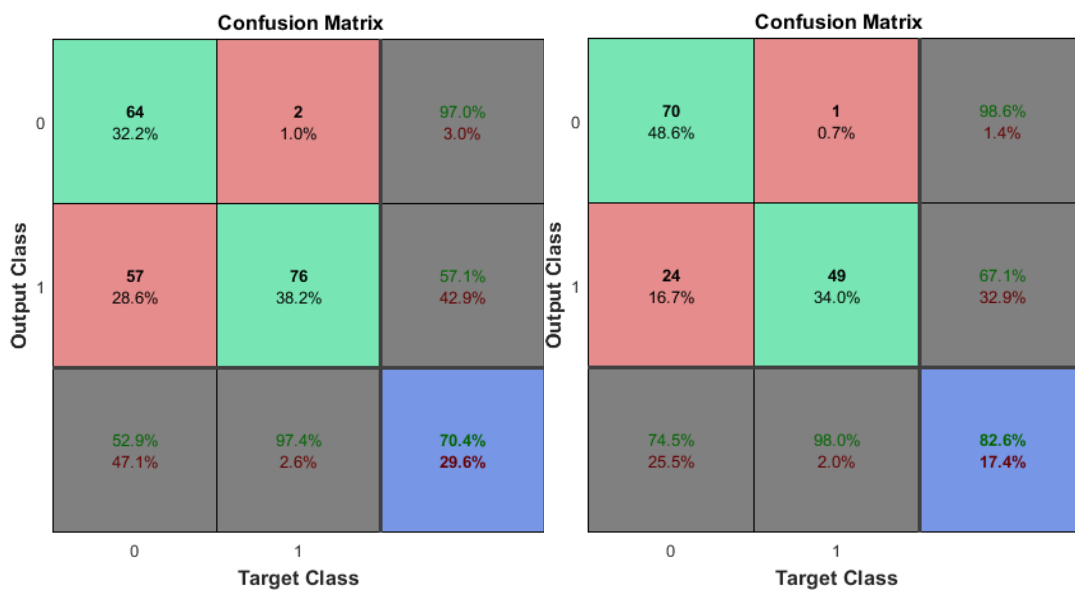


Figura 9.6. 2: Errores durante el entrenamiento para la sexta red.



9.6. 3: Matrices de confusión de la sexta red con dataset6 (izquierda) y dataset7 (derecha).

9.7 RESULTADOS DE LAS PRUEBAS

En este apartado, se recogerán los datos más relevantes obtenidos en la fase de pruebas y se agruparán en forma de tabla para una visualización más clara y ordenada.

			Acierto	Dataset1	Dataset2	Dataset3	Dataset4
2D	17x17	Dataset1	87%	-	86'2%	86,5%	91,9%
		Dataset4	95%	88%	86,50%	87,2%	-
	33x33	Dataset1	92%	-	-	-	83%
		Dataset4	97%	87,6%	-	-	-

Tabla 1: Resultados de las pruebas bidimensionales.

			Acierto	Train	Test	Train + test	Dataset6	Dataset7
3D	Dataset5	28x28xvar	94%	804/741	216/195	1020/936	67,5%	50,6%
		varxvarxvar	97%	737/737	195/187	932/924	70.4%	82,6%

Tabla 2: Resultados de las pruebas de la red tridimensional.

10. ANÁLISIS DE LOS RESULTADOS

En este capítulo se analizan los resultados más relevantes obtenidos durante la fase de pruebas. Es importante remarcar que en este apartado no se definirán tan detalladamente los conceptos de matriz de confusión, filtros o errores que ya se explican en profundidad en el apartado 7.5 Fase 4 de esta memoria. Simplemente, se recordará de manera breve su función:

Matriz de confusión: muestra en forma de matriz los resultados obtenidos para nuestro algoritmo basado en el aprendizaje supervisado. El eje denominado *Output Class* representa las predicciones o valores obtenidos de nuestra red, mientras que el eje *Target Class* representa los valores reales obtenidos correctamente. Es importante tener un número equilibrado y no demasiado bajo de ambas muestras para que el valor de precisión sea acertado.

Errores: Por errores nos referimos a las figuras que contienen información de ventanas cuya predicción ha sido errónea.

Filtros: es el resultado de aplicar las diferentes operaciones como por ejemplo convolución y que se centran en detectar características dentro de la imagen. Para este tipo de entradas de patches relativamente pequeños y para espigas que no son imágenes habituales o con elementos muy distintivos no se diferenciarán demasiado bien. Sin embargo, el input de nuestra red fueran fotografías de personas, los diferentes filtros mostrarían caras o incluso ojos o bocas que utilizarán para clasificar esa imagen como la imagen de un ser humano.

A continuación, se realizan diversas comparaciones tomando en cuenta el dataset utilizado, las dimensiones o si es bidimensional o tridimensional.

10.1 DATASETS UTILIZADOS

El número de aciertos o precisión de nuestra red se debe en gran medida al conjunto de datos utilizados para el proceso de entrenamiento. En los dos primeros casos, se han testeado las dos redes neuronales obtenidas con los mismos directorios y mismo tamaño de ventana, pero se han entrenado con datasets diferentes. Como se puede observar, aun contando con un menor número de muestras, vemos que la red entrenada con el dataset4 presenta mejores resultados pasando de un 87% a un 95% en la fase de testeo. En el resto de pruebas realizadas, la red entrenada con este último dataset también presenta un ligero incremento en la precisión al comparar ambas redes y testearlas con imágenes de nuevos directorios. Esto se debe en gran parte a que en las imágenes del dataset4 presentan una única espina y es mucho menos probable que se introduzcan erróneamente algunos elementos como no espigas pertenecientes a las dendritas cercanas. Es decir, que en general, observamos que la precisión del dataset4 también es mayor debido a la calidad de la información con la que la red ha sido entrenada.

Esta conclusión se verifica de nuevo en el caso bidimensional con ventanas de 33x33 donde se obtiene un 92% de acierto general para la red entrenada con el dataset1 frente a un 97% para la red entrenada con el dataset4. No obstante, aunque sí es cierto que se siguen obteniendo mejores resultados al entrenar con el dataset4, la diferencia se ha reducido en 3 puntos porcentuales. Esto se debe a que, al aumentar el tamaño de las ventanas de input, los criterios de no solapamiento son más estrictos y se reduce el número de regiones de no espina seleccionadas.

Por último y, como es lógico, siempre se obtienen mejores resultados al testear la red sobre las imágenes pertenecientes al dataset con el que ha sido entrenada que cuando se prueba con datasets diferentes, sin embargo, este tipo de pruebas no aporta demasiada información útil y han sido descartadas.

10.2 DIMENSIONES

Tal y como se acaba de introducir en el punto anterior, variar el tamaño de la ventana o cubo de las imágenes que formarán el input de la red influye en los resultados que se obtengan con ésta. Se ha observado que, en todos los casos, al aumentar este tamaño, el porcentaje y reconocimiento exitoso de espinas introducidas en la red mejora considerablemente. Por ejemplo, para el caso del dataset1 se aumenta entorno al 5% su precisión y también en el caso del dataset4 se produce un aumento de dos puntos porcentuales al aumentar la ventana de 17x17 a 33x33.

En el caso tridimensional, se produce de nuevo un aumento del 3% en la precisión global de la red al pasar de dimensiones 28x28x29 a 48x46x29. El motivo por el que se produce este aumento es que, al incrementar el tamaño de la ventana, se introduce más información de la espina en la red. De este modo, en lugar de disponer de segmentos de espina difícilmente diferenciables de la dendrita u otros elementos, disponemos de una información más global y completa que producen una mejor calidad del entrenamiento y desempeño de la red.

Se debe comentar, que incrementar excesivamente el tamaño de los cubos con los que se alimenta la red, supone riesgo de abarcar una región demasiado grande en la que se incluyan varias espinas u otros elementos no deseables. Por este motivo, para el caso tridimensional las dimensiones máximas correspondían a 48x46x29 que es un tamaño suficiente para englobar a cualquiera de las espinas reduciendo el riesgo de incluir partes de otras dendritas de las que no disponemos información e inducirían a una incorrecta clasificación.

Finalmente, al incrementar las dimensiones de los cubos o imágenes (input) con los que se alimenta la red en la fase de entrenamiento, se observa un aumento considerable del tiempo de ejecución pasando de unos 15 minutos a 156 minutos para el caso tridimensional de cubo máximo. Para la fase de segmentación de las espinas, también se observa un incremento del tiempo ya que, en este último caso, el número de directorios y datos de los que obtener información es mayor.

10.3 2D / 3D

Teóricamente, al introducir inputs tridimensionales, la información con la que se entrena la red es mayor y los resultados mejores. Efectivamente, los resultados obtenidos para el caso tridimensional han supuesto un incremento en la precisión de la red a cambio de un mayor tiempo de ejecución. Sin embargo, el incremento no es tan significativo como se esperaba. Por otro lado, las imágenes originales disponen de ruido o elementos no deseados que no corresponden ni a dendrita ni a espina pero que tampoco son totalmente negros. Probablemente, correspondan a elementos del medio en los que se encuentran las neuronas.

Para el caso tridimensional también se ha observado que, para ambos tamaños de cubo, el reconocimiento de espinas resulta bastante exitoso. Es decir, la red es capaz de distinguir un input como espina cuando se le presenta. Sin embargo, debido a estos elementos en el medio que hemos comentado antes o irregularidades de las propias dendritas, es común en el caso tridimensional detectar más inputs como espinas de los que en realidad hay. Otra posible causa por la que se observa una tendencia a detectar más espinas de las reales, es el hecho de englobar en las ventanas dendritas cercanas o incluso de la misma dendrita que no se encuentran en el fichero de puntos generados e identificados como tal. Es decir, que es posible que efectivamente estemos clasificando algunas de las espinas correctamente, pero al no disponer de esa información por parte del CSIC, no se correspondan con los valores de *ground truth* calculados. Es precisamente por este motivo y por un número reducido de muestras tridimensionales para los últimos datasets que, en algunas pruebas tridimensionales, los resultados puedan no ser tan satisfactorios como se esperaba. Sin embargo, con una máquina más potente que permita generar los cubos (inputs) más rápidamente y de más directorios diferentes, se podría obtener un número de muestras superior con mejores resultados y menos sensibles a desviaciones o errores esporádicos, lo que supondría una mayor confianza y estabilidad en dichos resultados.

11. CONCLUSIONES Y TRABAJO FUTURO

Los principales objetivos del presente proyecto final de carrera se podrían resumir en:

- Recopilar, interpretar y fusionar la información disponible de las imágenes cerebrales,
- Obtener los inputs necesarios cuyo contenido son regiones de espina y de no espina que permitirán, junto a la creación de un *ground truth*, entrenar la red.
- Encontrar una red (CNN en nuestro caso) que permita realizar este proceso de aprendizaje a través de la modificación de ciertos parámetros tanto en dos como en tres dimensiones.
- Evaluar la clasificación final de la red para nuevos inputs.

Los primeros puntos se describen de manera más extensa en el séptimo capítulo donde se detallan los pasos y metodologías seguidas. Además, no solo se incluye una solución al problema inicial, sino que se plantean y evalúan diferentes opciones hasta finalmente obtener un código o metodología general, lo más optimizado posible, que pueda aplicarse a todos los directorios de una manera cómoda y eficiente.

Tal y como se muestra en el capítulo de casos experimentales y resultados, se obtienen porcentajes muy razonables para la clasificación de espinas, desde el 70% hasta más del 90% en algunos casos. Estos porcentajes varían dependiendo de los directorios utilizados, los tamaños de ventana empleados y las estrategias bidimensionales o tridimensionales seguidas en el proceso de obtención de inputs.

Por todo ello, podemos concluir que se ha cumplido con los objetivos planteados para este proyecto fin de carrera. Además, cabe destacar que muy posiblemente, parte de este trabajo se utilice en un futuro como base para la publicación de un artículo científico una vez traducido y sintetizado.

Asimismo, aunque los resultados son bastante satisfactorios siempre cabe la posibilidad de seguir ajustando parámetros de la red o añadir capas para mejorar su rendimiento (proceso de *fine-tuning*) aún más o incluso buscar una CNN alternativa que también sea capaz de funcionar en dos y tres dimensiones. También, se puede seguir experimentando con diferentes tamaños de ventana, o criterios para la selección de espinas y no espinas a parte de los mencionados en este documento.

Otro posible trabajo futuro sería hacer un estudio más detallado del porcentaje de acierto de las diferentes redes entrenadas, repitiendo el entrenamiento más de una vez y comparando los resultados.

Finalmente, me gustaría destacar que, tal y como se ha comentado en la introducción, este trabajo forma parte de un proyecto internacional mayor que junto con mi plaza de alumno colaborador de la Escuela Politécnica Superior, me ha permitido desarrollar un gran número de habilidades, especialmente, relacionadas con la abstracción de problemas y búsqueda de

alternativas cuando los primeros resultados no son exitosos. Además, debido a la naturaleza de los proyectos de investigación, los resultados o métodos a implementar requieren de cierta imaginación y puesta en acción para verificar si son o no válidos, lo que supone un cierto reto personal. Por otro lado, ha sido fundamental una formación previa básica y la consulta de foros para solucionar todo tipo de problemas inesperados

BIBLIOGRAFÍA

- [1] <http://cajalbbp.cesvima.upm.es/#about>
- [2] <http://cajalbbp.cesvima.upm.es/img/OrganizacionModEnInf.png>
- [3] <https://es.wikipedia.org/wiki/Cerebro>
- [4] <https://es.wikipedia.org/wiki/Neurona>
- [5] <http://respuestas.tips/wp-content/uploads/2013/10/partes-de-una-neurona.jpg>
- [6] <https://es.wikipedia.org/wiki/Dendrita>
- [7] <https://neuwritesd.files.wordpress.com/2017/03/angry-y-u-no.jpg>
- [8] <http://cleverdata.io/que-es-machine-learning-big-data/>
- [9] <http://cleverdata.io/conceptos-basicos-machine-learning/>
- [10] [https://d3c33hcgivew3.cloudfront.net/974fa7509d583eabb592839f9716fe25_Lecture1.pdf?Expires=1498608000&Signature=fBijoDX0n2rDiKpKxHGOwasGLCv9p5sRqyoZbo0CZYGo~RxF5q5qXtTe8e7hlyonaDIVcbEBvSAwqaW96wasZssGnyyF2VwhBtBz0MNqz-JpfSHJdOusG4ciBwX-FOZ2ctE5zmbfpCv3BUkK2qCGTolkSEnSEloQCJIZStDNb5s_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A →pdf1 del curso de aprendizaje automático \(Coursera\).](https://d3c33hcgivew3.cloudfront.net/974fa7509d583eabb592839f9716fe25_Lecture1.pdf?Expires=1498608000&Signature=fBijoDX0n2rDiKpKxHGOwasGLCv9p5sRqyoZbo0CZYGo~RxF5q5qXtTe8e7hlyonaDIVcbEBvSAwqaW96wasZssGnyyF2VwhBtBz0MNqz-JpfSHJdOusG4ciBwX-FOZ2ctE5zmbfpCv3BUkK2qCGTolkSEnSEloQCJIZStDNb5s_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A →pdf1 del curso de aprendizaje automático (Coursera).)
- [11] <https://uploads.toptal.io/blog/image/330/toptal-blog-image-1395721394410.png>
- [12] <http://3qeqr26caki16dnhd19sv6by6v.wpengine.netdna-cdn.com/wp-content/uploads/2016/08/Why-Deep-Learning-1024x742.png>
- [13] <http://blogthinkbig.com/diferencias-entre-machine-learning-y-deep-learning/>
- [14] https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales
- [15] http://relopezbriega.github.io/images/conv_layer.png
- [15] <http://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/>
- [16] <https://s3-eu-west-1.amazonaws.com/com.cambridgespark.content/tutorials/convolutional-neural-networks-with-keras/figures/pool.png>
- [17] <http://cs231n.github.io/assets/nn1/relu.jpeg>
- [18] <http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg>
- [19] <https://www.coursera.org/learn/machine-learning>
- [20] <http://cs231n.github.io/convolutional-networks/>
- [21] <http://jorditorres.org/libro-hello-world-en-tensorflow/>
- [22] <http://www.deeplearningbook.org/>
- [23] <http://yann.lecun.com/exdb/mnist/>

[24] <https://github.com/hagaygarty/mdCNN>

Otros enlaces de interés:

<https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/>

https://es.wikipedia.org/wiki/Aprendizaje_profundo

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>

ANEXO 1. DESCRIPCIÓN DEL CÓDIGO EN MATLAB.

En esta sección, se detalla el conjunto de funciones y scripts utilizados de manera general para este proyecto. Concretar el funcionamiento detallado de cada programa, alargaría excesivamente el documento y no tendría una mayor utilidad, ya que, tras dedicar un cierto tiempo a analizarlo y haber leído el informe se pueden comprender la gran mayoría de las instrucciones o bucles dentro de éste.

La función principal de los scripts que se muestran a continuación será la de segmentar las espinas, por un lado, adaptar los inputs a nuestra red y los de la propia red. Además, se incluirá el formato de los archivos necesarios que se utilizan para la extracción de datos.

Api.if6.1.8enero.sp.000: ejemplo de archivo para el primer directorio que contiene información de todos los puntos que forman la espina 000. Cada directorio presentará múltiples archivos en función del número de espinas que tengan y cada archivo presentará una longitud variable en función del número de puntos que definan esa espina concreta.

Api.if6.1.8enero_Z000: corresponde a la primera imagen de microscopía del primer directorio. El número 0 se corresponde también con el plano 0. Como en el caso anterior, el número de imágenes tomadas para cada directorio será diferente con valores típicos de entre 30 y 70.

Api.m16.1.9-1enero.sp.0000.pxl: archivo que contiene los valores de x, y, z mínimos y máximos de cada espina para el primer directorio. Habrá tantos archivos como espinas haya.

Built_window: función utilizada para las pruebas bidimensionales que se encarga de localizar los centroides de los puntos y llamar la función `pintar` para obtener las ventanas en las orientaciones que se le indiquen por parámetro. Devolverá como resultado el conjunto de inputs preparados y el conjunto de *labels* o valores de *ground truth* de estos inputs.

Pintar: función que llama `built_window` y se encarga de seleccionar una subventana de la imagen original y reemplazar en una copia el valor de la matriz por un conjunto de "1" para que podamos consultar posteriormente si esa región ya ha sido seleccionada.

Pintar33x33: realiza la misma función que "pintar" pero para un tamaño de ventana mayor de 33x33 en lugar de 17x17.

Espina_xplanos: con este sencillo código se pretende ver en cuántos planos aparece una espina. Este código fue utilizado para realizar pruebas sobre el tamaño que debía tener el cubo en tres dimensiones.

Espinas_segmentadas: rellena los contornos de los puntos de espina obtenidos y llama a la función `built_window` para obtener el conjunto de datos necesarios para entrenar la red.

Espinas3Dgeneral: realiza una operación similar a `Espinas_segmentadas` pero para el caso tridimensional, obteniendo matrices tridimensionales como inputs en lugar de bidimensionales.

Imshow3D: script necesario para visualizar el cubo o matriz tridimensional a partir de las diferentes capas que la componen. No existe una función propia de Matlab, por este motivo se ha tenido que generar tal código.

Visualizar fichero: script para visualizar la nube de puntos que forman una espina.

Unplano_Unaespina: código para visualizar la intersección entre un plano y una única espina.

Intersec_todas_1plano: código utilizado en las fases iniciales para representar la intersección de un plano determinado con el conjunto de las espinas de un directorio concreto.

Visualizar_Espinas: fichero que permite visualizar todas las espinas convirtiendo los puntos de intersección a su valor correspondiente en píxeles para visualizarlas sobre la propia imagen.

Read_off: código necesario para poder leer el valor de los puntos de las espinas que se encuentran en los respectivos archivos. off.

Intersección: script capaz de generar el conjunto de puntos de intersección para cualquier directorio siempre que se disponga de los archivos necesarios.

Adaptar código: código necesario para subdividir y mezclar aleatoriamente el conjunto de datos obtenidos de las espinas segmentadas. En este script, se dividirán siguiendo las proporciones mencionadas a lo largo del documento dedicando un 80% al entrenamiento y un 20% al testeo. Organizará los valores de *inputs* y *ground truth* en una estructura llamada MNIST que utilizará la red.

Crearcubo: función similar a “pintar” pero en lugar de obtener subventanas de la imagen original como inputs generará cubos tridimensionales. De la misma forma, también colocará valores de “1” cuando esta región sea espina en dichas posiciones para poder distinguirla y no errar en la selección de regiones de no espina.

Crearventana: realiza la misma función que “crearcubo” pero a diferencia de “pintar” no utilizará los valores de puntos sino los valores de los ficheros que indiquen las posiciones de las espinas.

Visualizar ambos: código para visualizar la intersección entre un plano y una única espina.

Visualizar_z55: código inicial para visualizar un solo plano en el espacio.

Prueba: código que prueba nuestra red directamente sobre un conjunto de inputs. A continuación, obtendremos un mensaje que nos mostrará si el resultado de la clasificación es de 0 (no espina) o de 1 (espina). Si el input es un conjunto de datos, también nos devolverá una variable con el número total de espinas detectadas.

Intersección 2D: código para obtener la intersección de la nube de puntos con los planos correspondientes.

DemoMNIST/DemoMNIST3D: código de generación, entrenamiento y testeo de la red.