UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"REDES CONVOLUCIONALES PARA CLASIFICACIÓN DE IMÁGENES Y DIBUJOS. INTERPRETACIÓN DEL APRENDIZAJE POR VISUALIZACIÓN DE ACTIVACIONES INTERMEDIAS"

TRABAJO FIN DE GRADO

Septiembre - 2022

AUTOR: Cristina Sánchez Noguera

DIRECTOR/ES: Ramón Pedro Ñeco García

RESUMEN

En este trabajo se realiza una investigación sobre las técnicas de aprendizaje para la clasificación automática de imágenes y dibujos usando redes convolucionales. Además, se exploran técnicas para visualizar e interpretar el aprendizaje realizado por las redes, lo cual es especialmente relevante en aplicaciones en las que la clasificación automática se usa como ayuda o complemento a la clasificación manual como puede ser, por ejemplo, en imágenes para diagnóstico médico o en aplicaciones de control de calidad.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

1.1. OBJETIVOS

Los objetivos del presente trabajo se pueden resumir como sigue:

- Estudio y análisis de la arquitectura y funcionamiento de las redes convolucionales.

- Estudio y análisis del efecto del uso de las técnicas de aumento de datos y los efectos de sobreaprendizaje en las redes convolucionales.

- Estudio de aplicaciones de las redes convolucionales para tareas de clasificación de imágenes.

- Desarrollo e implementación de una aplicación para el reconocimiento automático de dibujos o esbozos rápidos hechos a mano usando redes convolucionales.

- Análisis de los efectos sobre la precisión del reconocimiento de los distintos parámetros de una red convolucional

- Estudio e implementación de métodos de visualización e interpretación del aprendizaje realizado por una red convolucional.

1.2. ESTRUCTURA DEL TRABAJO

Este trabajo se divide en seis apartados, los cuales se describen brevemente a continuación:

- Capítulo 1: Se tratará el tema objeto de este trabajo así como sus objetivos. También se realizará una breve descripción sobre los apartados que lo componen.
- Capítulo 2: Se describirán los fundamentos teóricos sobre la técnica y se hará un repaso sobre su origen.
- Capítulo 3: Se describirán las diferentes herramientas utilizadas para la realización de las simulaciones y obtención de resultados.
- Capítulo 4: Se explicarán detalladamente los programas utilizados así como las variaciones realzadas a dichos programas.
- Capítulo 5: Se describirán las simulaciones realizadas.
- Capítulo 6: Se expondrán las conclusiones acerca del trabajo y se comentarán posibles líneas futuras de trabajo.

CAPÍTULO 2. ESTADO DE LA TÉCNICA. APRENDIZAJE PROFUNDO (DEEP LEARNING)

2.1. ANTECEDENTES

El aprendizaje profundo, en ingles Deep Learning, es una rama de Machine Learning llamada así por el número de capas sucesivas que componen una red, considerándose esto la "profundidad", de ahí el término "deep" en Deep Learning. Este término es relativamente nuevo, ya que empezó a popularizarse a partir de 2006. Esta técnica permite/facilita el aprendizaje de patrones valiéndose del algoritmo de backpropagation para ir modificando los parámetros de la red. El campo del Deep Learning ha supuesto un avance en segmentación de imágenes, detección de objetos, clasificación de imágenes [1], reconocimiento de voz [2] o procesamiento de texto.

Si tenemos en cuenta los altos y bajos que ha sufrido su popularidad a través de los años y la repercusión de diferentes enfoques sobre la técnica, se podría dividir su desarrollo a lo largo de los años en tres etapas: la primera etapa surgiría con la cibernética y se situaría entre los años cuarenta y sesenta, la segunda comenzaría con el conexionismo y se produciría de los años ochenta a mediados de los noventa aproximadamente. Finalmente, la tercera etapa comenzaría a partir de 2006 adquiriendo el nombre por el cual la conocemos hoy en día.

La primera etapa comienza con la aparición de los primeros modelos lineales basados en el aprendizaje biológico, surgiría así en 1943 el primer modelo de neurona creado por McMulloch y Pitts[3]. En 1958, la creación del Perceptrón por Rosenblatt[4], que permitía el entrenamiento de una neurona. En 1969, Minsky y Papert publican un libro llamado *Perceptrons* en el que se demostraban las limitaciones del Perceptrón [5]. Esto provocó la caída de su popularidad así como una reducción de fondos e investigaciones en el campo de la inteligencia artificial que se conoce como el Invierno de la Inteligencia Artificial.

En 1974, Paul Werbos plantea la utilización del algoritmo backpropagation en redes neuronales, pero no será hasta los años ochenta cuando es aplicado de manera satisfactoria para entrenar redes neuronales [6],[7].

En 1998, LeCun propone el primer modelo de red convolucional [8] como conocemos hoy en día, tomando como base el Neocognitrón.

A mediados de los noventa aproximadamente se produjo otro periodo de declive debido al incumplimiento de las altas expectativas vertidas en redes neuronales y otras tecnologías relacionadas con la inteligencia artificial así como a los avances en otras áreas de Machine Learning. No fue hasta 2006, cuando Hinton creó las redes de creencia profunda[9], en inglés Deep Belief Networks, que volvieron a realizarse avances importantes en este campo. En los sucesivos años, varias han sido las redes convolucionales que han conseguido mejores resultados en competiciones que otras tecnologías de Machine Learning [10],[11]. Concretamente, sería en 2012 cuando la primera red convolucional gana el reto propuesto por ImageNet en clasificación de imágenes [1].

2.2. REDES NEURONALES

Una red neuronal es un modelo computacional inspirado en el modelo biológico de redes neuronales [12]. Estas redes están compuestas por una serie de nodos, llamados neuronas, interconectados entre sí, que componen a su vez diferentes capas. Las capas que conforman estas redes son la capa de entrada, una o varias capas ocultas y la capa de salida, como podemos apreciar en la figura 2-1. Cada neurona está conectada entre sí por un peso, este es el grado de importancia que se le da a una entrada a una neurona.
Este tipo de redes suelen ser conocidas como redes feedforward debido al sentido en el que los datos se propagan, ya que van desde la capa de entrada hasta la capa de salida pasando por las capas ocultas.

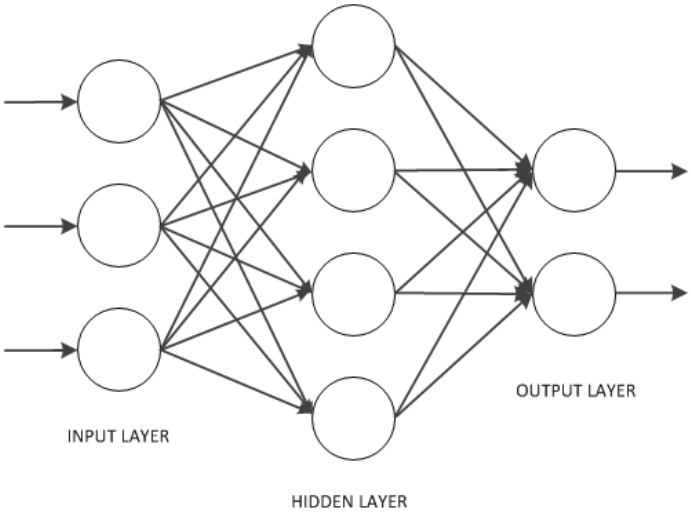


Figura 2-1. Esquema de la arquitectura de una red neuronal artificial.

Cada neurona realiza una suma ponderada de todas sus entradas mediante la fórmula 2.1, siendo *w* el peso de la entrada correspondiente, *b* el sesgo o bias, *x* las diferentes entradas a la neurona y *n* el número de entradas a la neurona. Obtenemos una activación *a,* a la cual se le aplica una función de activación, diferenciable y no lineal, para obtener la salida final de la neurona *y*.

$$a_j = \sum_{i=1}^{n} x_i * w_{ij} + b_j \tag{2.1}$$

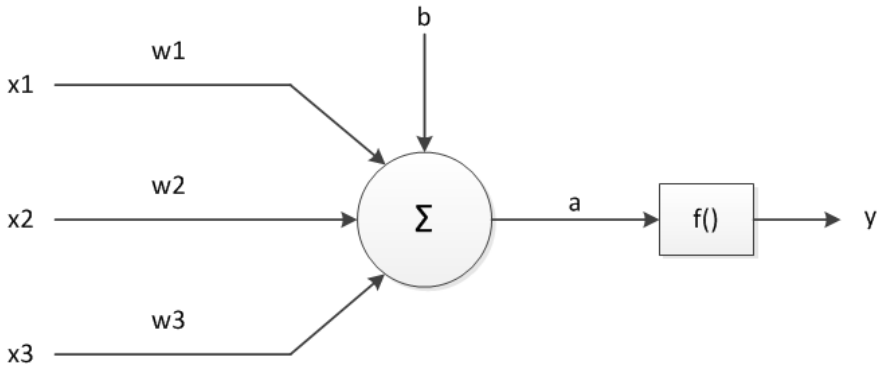


Figura 2-2. Esquema de una neurona artificial.

2.2.1. ESTRUCTURA DE LAS REDES NEURONALES

2.2.1.1 NEURONA DE MCCULLOCH Y PITTS

Esta primera neurona fue creada en 1943 y solo era capaz de realizar operaciones matemáticas básicas, debido a esto su aplicación se ve limitada. Su funcionamiento se basa en que cada entrada recibe un valor booleano, cero o uno, y si la suma de estos valores sobrepasa el valor del umbral, entonces la neurona se activa.

2.2.1.2 LINEAR THRESHLD UNIT

Para subsanar el hecho de que la neurona de McCulloch y Pitts le daba la misma importancia a todas sus entradas se crea la Unidad Lógica de Umbral, en inglés Linear Threshold Unit (LTU). Para ello esta unidad asigna un valor a cada entrada llamado peso, estos podrían ser positivos o negativos. Las entradas siguen teniendo un valor booleano pero este es multiplicado por el valor de su peso asignado. Si la suma de todas estas operaciones es positiva, entonces la LTU se activa.

2.2.1.3 PERCEPTRÓN

El Perceptrón es un algoritmo de clasificación binaria que está compuesto por un conjunto de Unidades Lógicas de Umbral agrupadas en una capa al que también se le puede añadir un sesgo para así aumentar el rendimiento de la red. Este es capaz de ajustar sus pesos con el fin de corregir el comportamiento de la red entrenada.

Si a un Perceptrón se le añaden varias capaz ocultas se le conoce como Perceptrón multicapa y es considerado como un tipo de red neuronal profunda. De hecho, muchos de los modelos creados son ejemplos de este.

2.2.1.4 RED NEURONAL PROFUNDA MODERNA

Este tipo de redes son una versión mejorada del Perceptrón multicapa. Utilizan funciones de activación más complejas que la función escalón, utilizada por el Perceptrón multicapa, como pueden ser ReLU o Sigmoid. Además, se valen de uno de los métodos de descenso del gradiente para su optimización.

2.2.1.5 FUNCIONES DE ACTIVACIÓN

Es una función que permite a la red neuronal aprender patrones complejos, ya que aumentan su capacidad de utilizar información relevante y omitir aquello que es irrelevante. Si bien no utilizarla hace que la red neuronal sea más simple también la hace menos potente e incapaz de aprender dichos patrones complejos.

Según la naturaleza del problema se utilizan diferente funciones de activación, por ejemplo para tareas de clasificación suele utilizarse Sigmoid debido a que es la mejor funciona.

2.2.1.6 FUNCIONES DE ERROR O COSTE

Son funciones cuyo objetivo es evaluar el grado de rendimiento de la red dados unos datos de entrada y se basa en el cálculo de la diferencia entre el valor deseado y el valor obtenido. Así, tenemos un error para cada predicción que realiza nuestra red, los cuales se encarga de agrupar en un solo valor de error final.

Al igual que las funciones de activación, deberemos elegir según la naturaleza de nuestro modelo. Por ejemplo, si estamos trabajando con clasificaciones binarias o clasificaciones multiclase podemos utilizar variaciones de la función crossentropy.

2.3. REDES CONVOLUCIONALES

Una red neuronal convolucional es un tipo de modelo de Deep Learning capaz de distinguir distintas características o patrones en una imagen, atribuyéndoles diferente importancia a través de sesgos y pesos. Todo ello sin necesidad de ser estas características extraídas manualmente, ya que con el suficiente entrenamiento la red es capaz de aprenderlas automáticamente. Este tipo de redes suelen ser utilizadas en tareas de segmentación o clasificación de imágenes.

Estas redes están compuestas por lo general por varias etapas. Las primeras etapas están formadas por capas de convolución y pooling o de agrupación, encargadas de la extracción de características. Por último, se componen de una serie de capas completamente conectadas cuya función es asignar las características extraídas a las salidas correspondientes, como por ejemplo en tareas de clasificación.

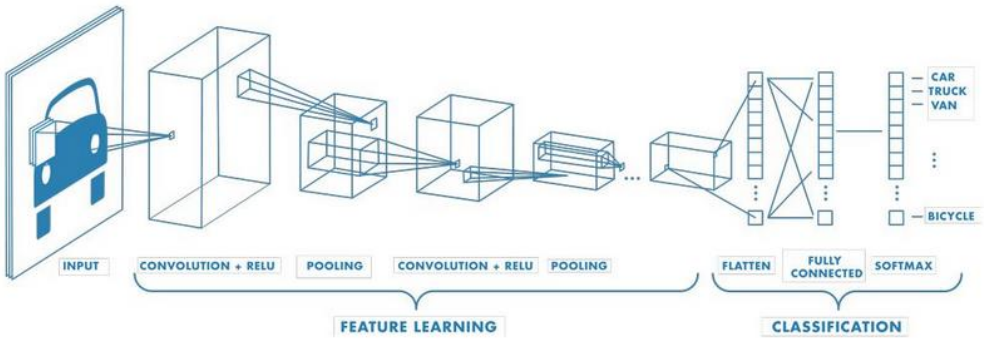


Figura 2-3. Esquema de la arquitectura de una red neuronal convolucional.

2.3.1. CAPA DE CONVOLUCIÓN

La capa de convolución es donde se aplican las operaciones convolucionales. Estas operaciones consisten en una matriz de tamaño n *x* n llamada filtro que va recorriendo de derecha a izquierda la imagen de entrada con un valor de paso, el paso es la distancia entre la posición de dos filtros consecutivos, realizando el producto elemento a elemento entre el filtro y el trozo de imagen sobre el que se sitúa obteniendo así un valor que formará parte del mapa de características resultante. Esta operación se va repitiendo utilizando diferentes filtros obteniendo así diferentes mapas de características que representan diferentes patrones, como por ejemplo podrían ser bordes verticales o texturas.

Las imágenes de entrada tienen un tamaño determinado por las dimensiones altura, ancho y profundidad. Esta profundidad define la de la capa de entrada, de igual manera la profundidad del filtro tiene que ser igual a la profundidad de la imagen de entrada. Además, la profundidad de la salida de una capa depende del número de filtros, si tenemos un número de filtros k y una imagen de entrada a los que se les aplica la operación de convolución entonces nuestro tensor de salida resultante tendrá un tamaño n *x* n *x* k.

Finalmente, al resultado obtenido tras cada capa de convolución se le aplica una función de activación no lineal antes de ser pasado a la siguiente capa, una de las más utilizadas es ReLU.

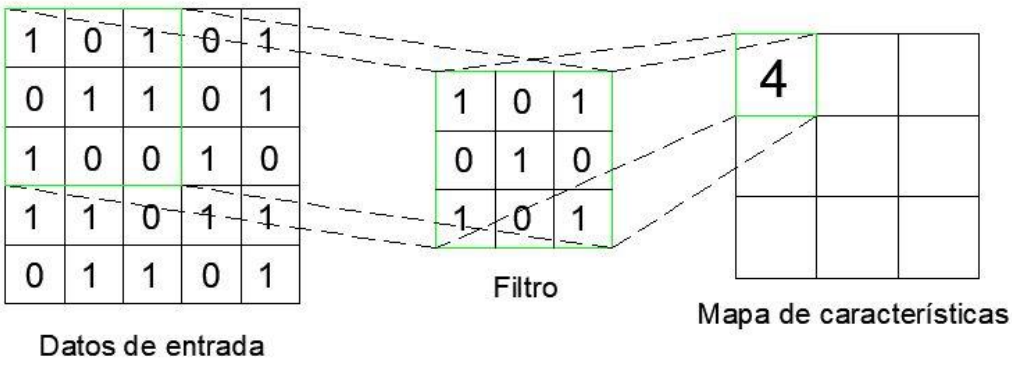


Figura 2-4. Ejemplo de una operación de convolución.

2.3.2. CAPA DE AGRUPACIÓN O POOLING

La capa pooling es la encargada mediante submuestreo de reducir el tamaño de la imagen obtenida tras la operación de convolución a la vez que conserva las características más importantes. La operación más utilizada para ello suele ser Max Pooling, esta operación consiste en obtener el máximo valor del segmento cubierto por el filtro sobre el mapa de características de entrada.

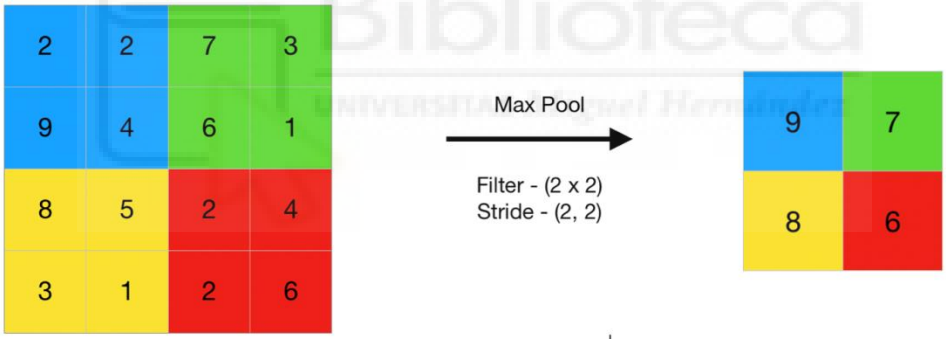


Figura 2-5. Ejemplo de una operación de Max Pooling.

2.3.3. CAPA COMPLETAMENTE CONECTADA

Por último, los mapas de características obtenidos se aplanan en un array de una dimensión antes de pasarlos a una o varias capas completamente conectadas. Estas capas se caracterizan porque todas las neuronas de una capa y la siguiente a ella están conectadas entre sí. La última capa completamente conectada suele tener el mismo número de salidas que de categorías. Su finalidad es mediante la función de activación Softmax aplicada a la última capa totalmente conectada clasificar las imágenes a partir de las características extraídas. Lo que hace esta función es asignar probabilidades a cada clase, siendo estas un valor entre 0 y 1, y debiendo sumar en total 1.

2.4. ALGORITMOS DE ENTRENAMIENTO

2.4.1. OPTIMIZACIÓN EN DEEP LEARNING

2.4.1.1 BACKPROPAGATION

Backpropagation es un algoritmo para calcular el gradiente que es utilizado para ajustar los pesos de una red repetidamente con el objetivo de minimizar el error final para que nuestra salida se acerque lo suficiente a la salida deseada, este se realiza desde la señal de error final hacia las primeras capas.

2.4.1.2 ALGORITMOS DE OPTIMIZACIÓN

Estos algoritmos trabajan junto con otros algoritmos para maximizar el rendimiento de estos últimos sin obtener retardo. Algunos de los más comunes son Adam, Descenso del Gradiente Estocástico o Rmsprop. Entre ellos los algoritmos Adam y Descenso del Gradiente Estocástico suelen ser los más utilizados.

Para entender mejor cómo funcionan estos algoritmos vamos a explicar brevemente el Descenso del Gradiente. Este algoritmo es utilizado para minimizar el error total o coste de una red neuronal. Hay diferentes implementaciones de este algoritmo:

- El original, el cual utiliza todos los datos de entrenamiento en cada iteración.
- Descenso del Gradiente Estocástico, el cual escoge una medida aleatoria para comprobar cómo cambia el error total según los cambios que se realizan en los pesos y sesgos.
- Mini-batch gradient descend, el cual utiliza lotes pequeño de datos para influir lo menos posible en la rapidez y fiabilidad del entrenamiento.

2.4.1.3 DESAFIOS EN LA OPTIMIZACIÓN

Podemos encontrarnos con tres desafíos a la hora de optimizar:

- Mínimo local, ocurre cuando nuestro algoritmo de optimización converge en un mínimo local en vez de un mínimo global.
- Punto de silla, ocurre cuando nuestro algoritmo queda atascado en un punto de silla incapaz de diferenciar si es un mínimo o máximo local.

- Desaparición del gradiente, ocurre cuando el algoritmo de optimización no puede reducir la salida de la función de error ya que el gradiente de esta función se aproxima a cero.

Para resolver estos problemas a la hora de optimizar es importante encontrar la mejor combinación entre la función de activación y la función de optimización para que nuestro modelo converja correctamente y encuentre un punto mínimo ideal.

2.4.2. SOBREENTRENAMIENTO Y REGULARIZACIÓN

2.4.2.1 OVERFITTING

El overfitting, también llamado sobreajuste, es un problema que surge cuando nuestro modelo se ajusta muy bien a todos nuestros datos sin ningún error, esto ocurre porque nuestro modelo ha memorizado los datos utilizados durante el entrenamiento y no es capaz de generalizar. Este fenómeno no es complicado de comprobar, ya que tiende a funcionar bien con los datos de entrenamiento pero no tanto con los datos de test.

2.4.2.2 REGULARIZACIÓN

Es una técnica utilizada para deshacerse del overfitting que consta de varios métodos posibles a utilizar:

- Early stopping. Consiste en parar el entrenamiento del modelo si este no muestra una mejora significativa en un número determinado de iteraciones.
- Dropout. El modelo elimina temporalmente algunas de las neuronas o capas que forman parte de la red, esto añade ruido adicional a la red lo que previene que el modelo se ajuste en exceso a los datos de entrenamiento y hace que sea más flexible.
- Regularización L1 y L2. Estos métodos añaden un término de corrección a la función de error, lo que hace que los errores sean aún más penalizados.
- Aumento de datos. Es un método utilizado para aumentar la cantidad de datos de entrenamiento de un conjunto de datos, esto se consigue realizando ligeras modificaciones en los datos existentes.

2.4.2.3 FEATURE SCALING

Es un método utilizado para normalizar el rango de las características con el objetivo de que las redes neuronales se ejecuten con mayor precisión. Hay varios métodos para llevar esto a cabo:

- Estandarización. Ajusta los valores de cada característica para que su media sea cero y su varianza unitaria.
- Rescaling. Escala los valores de cada característica para que estén en un rango entre 0 y 1 o 1 y -1.
- Mean normalization. Calcula la media de cada data point y divide el resultado entre el diferencial máximo y mínimo.
- Scaling to unit length. Divide cada componente de una característica por la distancia euclidiana de su vector.

Este método asegura que cada característica contribuya de manera proporcional al algoritmo de predicción y reduce el tiempo de entrenamiento del modelo acelerando la convergencia del algoritmo de Descenso del Gradiente.

2.5. INTERPRETACIÓN DEL APRENDIZAJE

Una de las cuestiones recurrentes en torno a las redes neuronales es como son capaces de aprender e interpretar, por ejemplo, imágenes. En concreto, saber cómo las redes convolucionales son capaces de distinguir diferentes imágenes y como aprenden a diferenciarlas puede ser interesante para ciertas aplicaciones como en medicina o en detección de errores. Para ello, hay tres técnicas que podemos aplicar que nos ayudan a visualizar e interpretar las representaciones aprendidas por estas redes:

- Visualización de activaciones intermedias. Consiste en mostrar la salida de ciertas capas a una entrada dada, esto nos da una visión de que características codifican los distintos canales que componen una capa.
- Visualización de filtros. Consiste en mostrar el patrón que se activa más en un determinado filtro.
- Visualización de mapas de calor. Consiste en mostrar mapas de calor de la activación de una clase sobre la imagen de entrada. Esto es interesante a la hora

de depurar nuestro modelo, ya que nos ayuda a hacernos una idea de en qué partes de una imagen se basa para clasificarla y a localizar objetos específicos en las imágenes.

CAPÍTULO 3. HERRAMIENTAS UTILIZADAS

En este apartado describiremos las herramientas Tensorflow y Keras utilizadas para la realización de este trabajo así como una pequeña descripción de la plataforma Google Colab y como comenzar a trabajar con ella.

3.1. TENSORFLOW Y KERAS

Tensorflow es una plataforma de código abierto destinada a Machine Learning que ofrece diferentes herramientas así como distintas API para el desarrollo e implementación de modelos en Machine Learning. Una de las más utilizadas es Keras, una API de alto nivel usada para la creación, entrenamiento y evaluación de dichos modelos en Tensorflow.

Entre las funcionalidades de Tensorflow cabe destacar que funciona tanto en CPU, como en GPU o TPU, puede calcular el gradiente de expresiones diferenciables y puede ser exportado a otros sistemas de tiempo de ejecución, como JavaScript o C++.

Antes de ver las arquitecturas y expresiones utilizadas en este trabajo vamos a comentar primero algunos conceptos a destacar sobre estas herramientas:

Tensorflow:

- Tensores, arrays multidimensionales no asignables.
- Variables, tensores asignables cuya función es almacenar el estado de la red.
- Operaciones de tensores, como ReLU. O el producto de tensores.
- Diferenciación automática, es la herramienta con la que Tensorflow implementa algoritmos como backpropagation.

Keras API:

- Layer o capa, es un módulo encargado de procesar datos, los cuales recibe como uno o varios tensores y devuelve como salida también uno o varios tensores. Una capa comprende tanto un estado, el cual seria los pesos de la red, como una transformación de las entradas a las salidas, esto último es llamado forward pass.

- Función de error, cuya función es minimizar la diferencia entre el valor deseado y el valor obtenido del parámetro que queremos evaluar durante el entrenamiento.
- Algoritmo de optimización, determina como aprende el modelo.
- Metrics, función encargada de evaluar el rendimiento del modelo.
- Bucle de entrenamiento, se encarga de aplicar el descenso del gradiente estocástico.

Los pasos a seguir para crear un modelo de clasificación de imágenes son los siguientes:

1. Preprocesar los datos.
2. Crear el modelo.
3. Entrenar el modelo.
4. Evaluar el modelo.

Previamente a estos pasos deberemos cargar el conjunto de datos con el que vayamos a trabajar. También es necesario importar las librerías con las que vayamos a trabajar, esto podrá hacerse en cualquier parte del código no necesariamente al principio.

3.1.1. 1. PREPROCESAMIENTO DE DATOS

Los datos necesitan estar en un rango entre 0 y 1 para poder ser "alimentados" a nuestra red neuronal, como las imágenes suelen estar en un rango entre 0 y 255 tendremos que proceder para que estos valores queden en el rango deseado. Hay diferentes formas de proceder si bien podemos dividir simplemente entre 255 también podemos utilizar la capa de preprocesamiento *rescaling*, en cuyo argumento scale deberemos asignarle 1./255.

Además, todas las imágenes deberán tener un determinado tamaño, para ello podemos utilizar la capa de preprocesamiento *reshape*. También podremos usar la función *image_dataset_from_directory()*, que aplica una serie de transformaciones para preprocesar los datos como cambiar el tamaño de las imágenes o indicar el tamaño de los lotes de datos.

Las dos capas de preprocesamiento nombradas anteriormente se añadirán a nuestro modelo como cualquier otro tipo de capa.

3.1.2. 2. CREACIÓN DEL MODELO

Hay tres API en Keras que nos permiten crear un modelo:

- El modelo secuencial, el cual se limita a apilar capas. Para ello lo primero que tendremos que hacer es definir un modelo secuencial al que le pasaremos como argumentos las capas que formaran parte de él. Vamos a ver un ejemplo para entenderlo mejor:

  model=keras.Sequential([
  layers.Dense(32, activation='relu'),
  layers.Dense(64, activation='relu'),
  layers.Dense(10, activation='softmax')
  ])

  También podemos utilizar el método *add():*

  model=keras.Sequential()
  model.add(layers.Dense(32, activation='relu')
  model.add(10, activation='softmax')

- La API funcional. Solo puede aplicarse a modelos con una entrada y una salida, apilando una capa sobre otra como el modelo secuencial. Primero, se declara un Input, a continuación se encadenan las llamadas a las capas creadas y finalmente se crea el modelo a partir de las entradas y salidas.

  inputs=keras.Input()
  x=layers.Dense() (inputs)
  outputs=layers.Dense() (x)
  model=keras.Model(inputs=inputs, outputs=outputs)

- Subclasificación de modelos, en la cual tú creas el modelo desde cero. Utilizando el método *__init__()* para definir las capas de nuestro modelo y el método *call()* para definir el forward pass.

Una vez definida la arquitectura de nuestro modelo podremos visualizarla mediante el método *model.summary()*.

A continuación, configuramos el proceso de entrenamiento. Para ello utilizamos el método *compile()*, donde configuramos la función de error, el algoritmo de aprendizaje y el parámetro a evaluar durante el entrenamiento con los argumentos *loss*, *optimizer* y *metrics* respectivamente.

model.compile(optimizer, loss, metrics)

3.1.3. 3. ENTRENAMIENTO DEL MODELO

Una vez tenemos todo listo pasamos al entrenamiento. En esta etapa implementamos el bucle de entrenamiento mediante el método *fit()*, donde tendremos que configurar parámetros como el número de épocas, los datos de entrada u otros como podrían ser los callbacks.

model.fit(x, y, epochs, validation_data, validation_split, batch_size, callbacks)

A continuación, comentamos los parámetros utilizados en este trabajo. El parámetro *x*, que especifica los datos de entrada; el parámetro *epochs*, especifica el número de iteraciones que se realizarán; el parámetro *validation_data*, que especifica los datos sobre los que se evalúa el error al final de cada época; el parámetro *validation_split*, que especifica la fracción de los datos que serán de validación; el parámetro *batch_size*, que especifica el número de muestras por actualización del gradiente y el parámetro *callbacks* con el cual especificamos los callbacks utilizados. Un *callback* es un objeto que realizada ciertas acciones en diferentes puntos del entrenamiento, por ejemplo, guardar el modelo cada cierto tiempo.

3.1.4. 4. EVALUACIÓN DEL MODELO

Por último, comprobamos cómo se comporta el modelo con los datos de evaluación o test mediante el método *evaluate()*, evaluando el parámetro deseado como podría ser la precisión con la que las imágenes son clasificadas.

model.evaluate(x)

Con el argumento *x* indicamos los datos de entrada que queremos evaluar.

3.2. GOOGLE COLAB

Todo ello lo realizaremos en cuadernos Jupyter alojados en Google Colaboratory, también llamado Colab, que nos permite tanto la programación como ejecución de código en Python en el navegador sin necesidad de descarga de ningún programa adicional. Permite el acceso a GPU o TPU y combinar texto con imágenes, LaTeX o HTML. Colab viene por defecto con Tensorflow y Keras instalado, por lo que no será necesario ninguna descarga para comenzar a trabajar con ellos.

Para comenzar a trabajar deberemos crear un cuaderno, podemos bien trabajar desde google drive o desde la página de Google Colaboratory (https://colab.research.google.com), donde clicaremos en Archivo/Nuevo cuaderno.

Una vez creado el documento podremos generar celdas tanto de código como de texto simplemente clicando en los botones + Código y + Texto situados en la barra de herramientas. Las celdas de código pueden ser ejecutadas pulsando el botón de reproducción situado en el lado izquierda de la celda o pulsando Ctrl+Enter. Las celdas pueden ser desplazadas a lo largo del documento desde la barra de herramientas superior que aparece al seleccionar una celda.

Antes de ejecutar cualquier fragmento de código deberemos conectarnos a la máquina virtual que nos proporciona Google Colab clicando en el botón conectar que aparece arriba a la derecha en la barra de herramientas.

Si queremos cambiar la configuración del entorno de ejecución deberemos clicar en Entorno de ejecución/Cambiar tipo de entorno de ejecución donde nos aparecerá una ventana donde podremos elegir el acelerador de hardware que queramos utilizar o incluso no utilizar ninguno. Para ver los archivos cargados en nuestro cuaderno deberemos clicar en el icono de carpeta situado en la barra lateral.

CAPÍTULO 4. SISTEMA DE CLASIFICACIÓN UTILIZANDO REDES CONVOLUCIONALES

En este apartado veremos cómo aplicar las redes convolucionales para tareas de clasificación de imágenes partiendo de un ejemplo de clasificación de imágenes de perros y gatos para luego centrarnos en un modelo para clasificación de dibujos rápidos a mano.

4.1. CLASIFICACIÓN DE IMÁGENES DE PERROS Y GATOS

Para nuestro primer sistema de clasificación vamos a partir del ejemplo del capítulo 8 apartado 8.2 del libro *Deep Learning with Python* de François Chollet sobre clasificación de imágenes. Utilizaremos el conjunto de datos sacado de una competición propuesta en 2013 por Kaggle que consistía en clasificar imágenes de perros y gatos. Este conjunto de datos está compuesto por 25000 imágenes entre las dos clases. Sin embargo, nosotros vamos a utilizar una versión reducida de este utilizando solo 5000 imágenes; 2000 imágenes para entrenamiento, 1000 para validacion y 2000 para test. A continuación, veremos una explicación acerca del código utilizado.

Para comenzar debemos obtener el dataset con el cual vamos a trabajar. Se trata de un conjunto de datos alojado en kaggle, por lo que utilizaremos una API proporcionada por ellos para poder descargarlo en nuestro cuaderno en Colab. Esta API solo está disponible para usuarios de kaggle por lo que tendremos que identificarnos, para ello tendremos que cargar un archivo con extensión .json que nos proporcionan desde su pagina web, donde se encuentran nuestras credenciales, a través del comando *files.upload()*. Con el comando *mkdir* creamos un directorio llamado kaggle, donde copiaremos el archivo .json con el comando *cp*. Ahora ya podemos descargarnos el conjunto de datos dogs-vs-cats con la siguiente linea de código: *!kaggle competitions download -c dogs-vs-cats* y descomprimirlo con *unzip*.

```
from google.colab import files
files.upload()
```

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle competitions download -c dogs-vs-cats
!unzip -qq dogs-vs-cats.zip
!unzip -qq train.zip
```

Posteriormente, crearemos un dataset más pequeño a partir del dataset original para trabajar con él. Utilizaremos las imágenes alojadas en la carpeta *train* para crear dicho dataset al que llamaremos *cats_vs_dogs_small*, dividiéndolo en tres subconjuntos: train, compuesto por mil imágenes para cada clase, validation, compuesto por quinientas imágenes para cada clase, y test, compuesto por mil imágenes para cada clase. Para ello crearemos una función llamada *make_subset*, que se encarga de copiar imágenes del dataset original al subconjunto que queremos crear a partir del comando *shutil.copyfile*. Con los argumentos *start_index* y *end_index* indicaremos el rango de imágenes que queremos para nuestro dataset.

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
      for category in ("cat", "dog"):
            dir = new_base_dir / subset_name / category
            os.makedirs(dir)
            fnames = [f"{category}.{i}.jpg" for i in range(
            start_index, end_index)]
            for fname in fnames:
                  shutil.copyfile(src=original_dir / fname,
                              dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2500)
```

A continuación, preprocesaremos los datos. La función *image_dataset_from_directory(directory, image_size, batch_size)* aplica una serie de transformaciones para preprocesar los datos. Esta función produce lotes de imágenes a partir del directorio especificado junto con las etiquetas 0 y 1, redimensionando las

imágenes para que todas sean del mismo tamaño, definiendo este tamaño a través del parámetro *image_size*. Con el parámetro *directory* le indicamos donde están almacenados los datos que va a necesitar y con *batch_size* el tamaño de los lotes de datos.

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
      new_base_dir / "train",
      image_size=(180, 180),
      batch_size=32)
validation_dataset = image_dataset_from_directory(
      new_base_dir / "validation",
      image_size=(180, 180),
      batch_size=32)
test_dataset = image_dataset_from_directory(
      new_base_dir / "test",
      image_size=(180, 180),
      batch_size=32)
```

Seguidamente, aplicaremos la técnica de aumento de datos. Con este fin crearemos una serie de capas de preprocesamiento de imágenes con *keras.Sequential*, que más tarde incluiremos en nuestra red. Las capas utilizadas serán las siguientes:

- *Layers.RandomFlip(mode)* le dará la vuelta a las imágenes según el argumento *mode*, pudiendo ser "horizontal", "vertical" u "horizontal_and_vertical", de manera aleatoria.
- *Layers.RandomRotation(factor)* rotará las imágenes de manera aleatoria. El argumento *factor* consiste en un número en coma flotante que representa el rango de la rotación en sentido horario y antihorario.
- *Layers.Zoom(heigh_factor)* hará zoom, acercándose o alejándose, a las imágenes aleatoriamente según el argumento *heigh_factor*.

```
data_augmentation = keras.Sequential(
      [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
       ]
)
```

El siguiente paso será crear la red neuronal convolucional. Empezaremos instanciando las entradas con el comando *keras.Input(shape)*. El argumento *shape* define el tamaño y número de canales de la imagen. Utilizamos *layers.Rescaling(scale)* para modificar el rango de valores de los pixeles de las imágenes de entrada de [0,255] a [0,1], indicándolo como 1./255 con el argumento *scale*.

Ahora definimos las capas que forman parte de la arquitectura de nuestra red, añadiendo las ya creadas anteriormente para el aumento de datos.

Con *layers.Conv2D(filters, kernel_size, activation)* crearemos las capas de convolución. El parámetro *filters* define el número de filtros utilizados, el parámetro *kernel_size* el tamaño de los filtros y el parámetro *activation* la función de activación utilizada.

Con *layers.MaxPooling2D(pool_size)* crearemos las capas de agrupación, el parámetro *pool_size* indica el tamaño de la "ventana" utilizada para las operaciones de submuestreo.

Con *layers.Flatten* aplanaremos la salida de la capa anterior a esta para poder pasarla como entrada a una capa densa.

Layers.Dropout(rate) aplicará la técnica de dropout, con el parámetro *rate* indicaremos la probabilidad con la que las neuronas se desactivarán de las capas ocultas.

Layers.Dense(units, activation) creará una capa densa, con el parámetro *units* indicaremos el número de neuronas que constituyen la capa y con el parámetro *activation* la función de activación que vamos a utilizar.

Crearemos la red con *Keras.Model(inputs, outputs)*, con el parámetro *inputs* indicamos las entradas a la red y con el parámetro *outputs* las salidas de la red.

Con *model.compile(loss, optimizer, metrics)* configuraremos la red para entrenarla, definiendo la función de error a través del parámetro *loss*, en este caso utilizaremos *binary_crossentropy*, con el parámetro *optimizer* indicamos el algoritmo de aprendizaje, utilizaremos el algoritmo *rmsprop*, y con el parámetro *metrics* la medida que queremos que el modelo evalúe durante el entrenamiento y test, en este caso se tratará de la precisión con la que la red identifica las imágenes.

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Entrenaremos la red con *model.fit(x, epochs, validation_data, callbacks)*. Con el argumento *x* especificaremos los datos de entrada, con *epochs* el número de iteraciones que realizará la red y con *validation_data* indicaremos los datos utilizados para evaluar cualquier parámetro especificado anteriormente al final de cada iteración.

Utilizaremos como callback *ModelCheckpoint(filepath, sabe_best_only, monitor)*, cuya función es guardar el modelo después de cada época. Si *save_best_only* es true guardará la red cuando el valor del error actual sea más bajo que cualquiera de los anteriores. Con el parámetro *filepath* indicaremos la ruta donde queremos guardar el archivo de la red y con *monitor* el parámetro que queramos monitorizar.

```
callbacks = [
      keras.callbacks.ModelCheckpoint(
            filepath="convnet_from_scratch_with_augmentation.keras",
            save_best_only=True,
            monitor="val_loss")
]
history = model.fit(
      train_dataset,
      epochs=100,
      validation_data=validation_dataset,
      callbacks=callbacks)
```

Posteriormente, representamos gráficamente la evolución del entrenamiento en dos gráficas distintas. En una comparamos los valores de error obtenidos tanto en los datos de entrenamiento como en los de validación. Lo mismo con los valores de precisión.

```
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14,5))

ax[0].plot(epochs, accuracy, "bo", label="Training accuracy")
ax[0].plot(epochs, val_accuracy, "b", label="Validation accuracy")
ax[0].set_title("Training and validation accuracy")
ax[0].legend()

ax[1].plot(epochs, loss, "bo", label="Training loss")
ax[1].plot(epochs, val_loss, "b", label="Validation loss")
ax[1].set_title("Training and validation loss")
ax[1].legend()

fig.savefig("name.png", bbox_inches='tight')
plt.show()
```

Por último, cargaremos el modelo que mejor resultado nos ha dado con *keras.models.load_model(filepath). Test_model.evaluate(x)* devuelve el valor del error y precisión, evaluándolo en el modo test.

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

4.1.1. PRUEBAS REALIZADAS

En este apartado describiremos las modificaciones realizadas en la arquitectura del sistema de clasificación de imágenes de perros y gatos. Estas consistirán en ir modificando distintos aspectos de su arquitectura como el número de capas, el número de neuronas de cada capa y la distribución de las capas. También utilizaremos distintas técnicas como dropout, regularización L2, extracción de características o fine tuning.

- PRUEBA 1

Reducimos a la mitad el número de filtros de todas las capas convolucionales.

- PRUEBA 2

Duplicamos el número de filtros de todas las capas convolucionales.

- PRUEBA 3

Añadimos a la arquitectura una capa densa con cien neuronas y función de activación ReLU, sin modificar el número de filtros inicial de las capas convolucionales.

- PRUEBA 4

A la modificación 3 le aplicamos la regularización L2 a la capa densa añadida, con un valor de regularización de 0.001, manteniendo el número de neuronas a cien.

- PRUEBA 5

Probamos ahora variando en la modificación anterior el número de neuronas de la capa densa a 250 neuronas.

- PRUEBA 6

Modificamos ahora el número de neuronas de la capa densa a 50.

- PRUEBA 7

Reducimos el número de capas de nuestra arquitectura inicial, pasando a tener tres capas de convolución y dos de agrupación alternándose sucesivamente.

- PRUEBA 8

Reducimos el número de filtros a la mitad de la modificación anterior.

- PRUEBA 9

Duplicamos el número de filtros de la modificación 7.

- PRUEBA 10

Añadimos a la modificación 7 una capa densa con 100 neuronas y función de activación ReLU.

- PRUEBA 11

Variamos el número de neuronas de la capa densa a 500.

- PRUEBA 12

Aplicamos la regularización L2 a la modificación 10.

- PRUEBA 13

Probamos con cuatro capas de convolución y tres de agrupación alternándose sucesivamente, siendo el número de filtros de las capas convolucionales 32, 64, 128 y 256 respectivamente.

- PRUEBA 14

Disminuimos el número de filtros a la mitad de la modificación anterior.

- PRUEBA 15

Duplicamos el número de filtros de la modificación 13.

- PRUEBA 16

Vamos a probar otra combinación de la arquitectura anterior, esta vez constará de dos capas de convolución seguidas de una de agrupación y así sucesivamente. El número de filtros será el indicado en la modificación anterior: 32, 64, 128 y 256 respectivamente.

- PRUEBA 17

Probamos con cuatro capas convolucionales y dos de agrupación, alternándose excepto las dos últimas que serán dos capas de convolución seguidas. Añadimos una capa densa con doscientas cincuenta neuronas y función de activación ReLU. El número de filtros de las capas convolucionales será de 32, 64, 128 y 128 respectivamente.

- PRUEBA 18

Probamos con cuatro capas convolucionales y cuatro de agrupación alternándose, siendo el número de filtros 32, 64, 128 y 256 respectivamente.

- PRUEBA 19

Duplicamos el número de filtros de la anterior arquitectura.

- PRUEBA 20

Disminuimos el número de filtros a la mitad de la modificación 18.

- PRUEBA 21

Probamos ahora variando los filtros, modificándolos de manera que la primera capa convolucional tenga 32, la segunda 64 y las dos últimas 128 filtros.

- PRUEBA 22

Añadimos a la modificación 18 una capa densa con cien neuronas y función de activación ReLU.

- PRUEBA 23

Probamos cambiando el número de neuronas de la capa densa a doscientas cincuenta.

- PRUEBA 24

Añadimos la técnica de regularización L2 a la capa densa con un valor de regularización de 0.001 y cambiamos el número de neuronas de dicha capa a 64. También modificamos el parámetro *rate* de la capa dropout a 0.2.

- PRUEBA 25

Añadimos otra capa de convolución y otra de agrupación a la anterior modificación, teniendo cinco capas de convolución y cinco de agrupación. Además, aumentamos el número de neuronas de la capa densa a 512. Cambiamos el número de filtros a 32, 64, 64, 128 y 128 respectivamente.

- PRUEBA 26

Quitamos la capa densa que habíamos añadido a la modificación anterior y por tanto, la regularización L2. Cambiamos el número de filtros a 64, 128, 256, 256 y 256 respectivamente. También modificamos el parámetro *rate* de la capa dropout a 0.5.

- PRUEBA 27

Ahora le añadimos a la modificación anterior una capa densa con 512 neuronas y función de activación ReLU. Además, le añadimos la regularización L2 con un valor de regularización de 0.001.

- PRUEBA 28

Modificamos el número de filtros de la modificación anterior a 32, 64, 128, 256 y 256 filtros respectivamente. Además, cambiamos el valor del parámetro *rate* de la capa dropout a 0.2. También quitamos la capa densa con la regularización L2.

- PRUEBA 29

Aplicamos la técnica de extracción de características con aumento de datos.

- PRUEBA 30

Aplicamos la técnica de fine tuning.

4.2. RECONOCIMIENTO DE DIBUJOS

Para nuestro sistema de reconocimiento de dibujos utilizaremos el conjunto de datos llamado "The Quick, draw! Dataset" desarrollado por Google para entrenar una red convolucional para el reconocimiento de dibujos rápidos hechos a mano. Este conjunto de datos fue obtenido a partir de un juego online llamado Quick, draw! en el que los participantes debían dibujar en menos de veinte segundos la figura que se les indicaba. Vamos a ver a continuación una descripción del código empleado para ello.

Primero cargaremos un fichero con extensión .txt donde aparecerán las clases utilizadas, para ello utilizaremos el método *files.upload()*.

```
from google.colab import files
files.upload()
```

Abriremos el archivo que contiene las clases a utilizar con la función *open()* y con el método *readlines()* asignaremos a *classes* cada línea del archivo como un elemento de una lista. Por último, cerraremos el archivo con el método *close()*.

```
f = open("mini_classes_25.txt","r")
# And for reading use
classes = f.readlines()
f.close()
```

Una vez tenemos todas las líneas, reemplazaremos los saltos de línea por espacios y los espacios por guiones bajos mediante un bucle *for* y el método *replace()*.

```
classes = [c.replace('\n','').replace(' ','_') for c in classes]
```

A continuación, descargaremos el dataset. Lo guardaremos en una carpeta llamada *data* creada a partir del comando *mkdir*. Para ello, crearemos una función a la que llamaremos *download* que se encargará de descargar los datos a partir de la dirección url contenida en *path* y los guardaremos en la carpeta *data* mediante un bucle *for*.

```
!mkdir data
import urllib.request
def download():

  base = 'https://storage.googleapis.com/quickdraw_dataset/full/num
py_bitmap/'
  for c in classes:
    cls_url = c.replace('_', '%20')
    path = base+cls_url+'.npy'
    print(path)
    urllib.request.urlretrieve(path, 'data/'+c+'.npy')


download()
```

Cada clase contiene diferentes muestras de números de matrices almacenadas en formato .npy. Como tenemos algunas limitaciones de memoria, solo cargaremos 4000 imágenes por clase. Para ello, crearemos una función llamada *load_data*, la cual inicializará las variables *x*, *y* y *class_names*, seguidamente cargará los datos en sus ficheros correspondientes y a continuación, generará un orden aleatorio en el conjunto de datos para que no salgan 4000 imágenes seguidas de la misma clase. Por último, separará los datos en datos de entrenamiento y datos de test.

```
def load_data(root, vfold_ratio=0.2, max_items_per_class= 4000 ):
    all_files = glob.glob(os.path.join(root, '*.npy'))

    # inicializamos variables
    x = np.empty([0, 784])
    y = np.empty([0])
    class_names = []
```

```
    # cargamos cada fichero de datos
    for idx, file in enumerate(all_files):
        data = np.load(file)
        data = data[0: max_items_per_class, :]
        labels = np.full(data.shape[0], idx)

        x = np.concatenate((x, data), axis=0)
        y = np.append(y, labels)

        class_name, ext = os.path.splitext(os.path.basename(file))
        class_names.append(class_name)

    data = None
    labels = None

    # Generamos un orden aleatorio en el dataset,
    permutation = np.random.permutation(y.shape[0])
    x = x[permutation, :]
    y = y[permutation]

    # separamos en datso de entrenamiento y de test
    vfold_size = int(x.shape[0]/100*(vfold_ratio*100))

    x_test = x[0:vfold_size, :]
    y_test = y[0:vfold_size]

    x_train = x[vfold_size:x.shape[0], :]
    y_train = y[vfold_size:y.shape[0]]
    return x_train, y_train, x_test, y_test, class_names

x_train, y_train, x_test, y_test, class_names = load_data('data')
```

Ahora procederemos a preprocesar los datos, así primero ajustaremos el tamaño de las imágenes según la variable *image_size* y normalizaremos los valores para que estén en un rango entre 0 y 1 dividiéndolos entre 255. Por último, convertiremos los vectores y_train e y_test en matrices con la función *to_categorical*.

```
num_classes = len(class_names)
image_size = 28

# Hacemos un reshape y normalización
x_train = x_train.reshape(x_train.shape[0], image_size, image_size,
 1).astype('float32')
x_test = x_test.reshape(x_test.shape[0], image_size, image_size, 1)
.astype('float32')
```

```
x_train /= 255.0
x_test /= 255.0

# Convertimos vectores a matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Definiremos el modelo con la API modelo secuencial añadiendo las capas con el método *add()*, configurándolo de manera que tendremos tres capas convolucionales con 16, 32 y 64 respectivamente y función de activación ReLU, tres capas de agrupación con tamaño 2x2 de la ventana y una capa densa con 128 neuronas y función de activación ReLU previa a la capa de salida. Finalmente, configuraremos el proceso de entrenamiento mediante el método *compile()*.

```
model = keras.Sequential()
model.add(layers.Convolution2D(16, (3, 3),
                        padding='same',
                        input_shape=x_train.shape[1:], activation='
relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Convolution2D(32, (3, 3), padding='same', activati
on= 'relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Convolution2D(64, (3, 3), padding='same', activati
on= 'relu'))
model.add(layers.MaxPooling2D(pool_size =(2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))

# Cambiar el número de unidades según el número de clases
model.add(layers.Dense(25, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['top_k_categorical_accuracy'])
print(model.summary())
```

Pasamos ahora al entrenamiento del modelo durante 5 épocas, utilizando como callback el ya explicado en el apartado 4.1.1.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="red_entrenada_25.keras",
        save_best_only=True,
```

```
        monitor="val_loss")
]

model.fit(x = x_train, y = y_train, validation_split=0.1, batch_siz
e = 256, epochs=5, callbacks=callbacks)
```

A continuación, comprobaremos el resultado de la red evaluándola en los datos de test.

```
score = model.evaluate(x_test, y_test)
print('Test accuarcy: {:0.2f}%'.format(score[1] * 100))
```

Mostramos un ejemplo de inferencia, tomando una imagen aleatoria de los datos de test. También generamos 5 predicciones de salida para la imagen de entrada con el método *predict()*.

```
import matplotlib.pyplot as plt
from random import randint
%matplotlib inline
idx = randint(0, len(x_test))
img = x_test[idx]
plt.imshow(img.squeeze())
pred = model.predict(np.expand_dims(img, axis=0))[0]
ind = (-pred).argsort()[:5]
latex = [class_names[x] for x in ind]
print(latex)
```

4.2.1. PRUEBAS REALIZADAS

En este apartado describimos las pruebas realizadas en el sistema de clasificación de dibujos rápidos hechos a manos. Estas pruebas consistirán en realizar una serie de modificaciones en la arquitectura de la red tales como variar el número de capas de la arquitectura así como su distribución o el número de filtros de las capas. También probaremos con diferente número de clases.

- PRUEBA 1

Cambiamos el número de filtros de las capas convolucionales, excepto la primera, y la penúltima capa densa a 16, 64 y 128 filtros respectivamente.

- PRUEBA 2

Cambiamos en la prueba 1 los filtros de la segunda capa convolucional a 64 y la tercera capa convolucional a 128.

- PRUEBA 3

Cambiamos el número de filtros de las capas convolucionales y la penúltima capa densa a 32, 64, 256 y 256 respectivamente.

- PRUEBA 4

Cambiamos en la prueba 3 el número de filtros de la primera capa convolucional a 64.

- PRUEBA 5

Cambiamos el número de filtros de las capas convolucionales y la penúltima capa densa a 32, 32, 128 y 128 respectivamente.

- PRUEBA 6

Variamos los filtros de todas las capas convolucionales y la penúltima capa densa a 16, 32, 64 y 64 respectivamente.

- PRUEBA 7

Modificamos la arquitectura añadiéndole tres capas convolucionales con lo que tendríamos seis capas de convolución, agrupándolas de dos en dos seguidas de una capa de agrupación. Variamos el número de filtros de las capas convolucionales y la penúltima capa densa a 16, 16, 32, 32, 128, 128 y 256 respectivamente.

- PRUEBA 8

Duplicamos el número de filtros de las cuatro primeras capas convolucionales de la prueba anterior.

- PRUEBA 9

Modificamos la arquitectura de manera que tenemos cuatro capas convolucionales y cuatro capas de agrupación alternándose sucesivamente. Definimos el número de filtros de las capas convolucionales y la penúltima capa densa a 32, 64, 128, 128 y 256 respectivamente.

- PRUEBA 10

Variamos el número de filtros de la primera capa convolucional de la prueba anterior a 64.

- PRUEBA 11

Cambiamos el número de filtros de las tres primeras capas convolucionales de la prueba 10 a 16, 32 y 64 respectivamente.

- PRUEBA 12

Modificamos el número de filtros de la penúltima capa densa de la prueba anterior a 128.

- PRUEBA 13

Modificamos el número de capas de la arquitectura a cuatro de convolución y dos de agrupación, de manera que agrupamos las capas convolucionales de dos en dos seguidas de una capa de agrupación. Definimos el número de filtros como 16 el primer grupo de capas convolucionales, 32 el segundo y 128 la penúltima capa densa.

- PRUEBA 14

Cambiamos de la prueba anterior el número de filtros del primer grupo de capas convolucionales a 32 y del segundo grupo a 64.

- PRUEBA 15

Variamos el número de filtros de la prueba 13 a 64 el primer grupo de capas convolucionales, 128 el segundo y 256 la penúltima capa densa

- PRUEBA 16

Modificamos el número de capas de la arquitectura a cinco capas convolucionales y cuatro de agrupación las cuales van alternándose sucesivamente, siendo el número de filtros de 16, 32, 64, 128 y 128 respectivamente. Cambiamos el número de filtros de la penúltima capa densa a 256.

- PRUEBA 17

Cambiamos el número de filtros de la primera capa convolucional de la prueba anterior a 32.

- PRUEBA 18

Variamos el número de filtros de la prueba 16 de la primera y segunda capa convolucional a 64 y los de la quinta capa convolucional a 256.

- PRUEBA 19

Cambiamos de la prueba 16 el número de filtros de las capas convolucionales y la penúltima capa densa a 16, 16, 32, 32, 64 y 64 respectivamente.

- PRUEBA 20

Trabajamos con 50 clases en la arquitectura original.

- PRUEBA 21

Trabajamos con 100 clases en la arquitectura original.

4.3. ACTIVACIONES INTERMEDIAS

En este tipo de redes es relativamente fácil poder identificar qué es lo que aprenden, ya que son representaciones de conceptos visuales. Hay diferentes técnicas para poder visualizar que aprenden estas redes, entre ellas están la visualización de activaciones intermedias, la visualización de filtros y la visualización de mapas de calor.

En este trabajo nos centraremos en la técnica de visualización de activaciones intermedias. Vamos a obtener los mapas de características obtenidos en cada capa que forma parte de la arquitectura de la red mediante la visualización de las activaciones intermedias, llamamos activación a la salida de una capa. Nos centraremos en la activación de la primera capa convolucional.

Una vez descargados los datos correspondientes a las veinticinco clases, cargaremos los datos de manera que tendremos cuatro mil imágenes por clase y procederemos a preprocesar los datos. Estos pasos se realizarán de igual manera que en el modelo de reconocimiento de dibujos. Tras realizar estos pasos cargaremos el modelo previamente entrenado en nuestro sistema de reconocimiento de dibujos.

Para obtener estas activaciones primero crearemos un modelo que devuelve como salida las activaciones de cada capa de nuestro modelo previamente entrenado. Para ello obtendremos las capas convolucionales y de agrupación mediante un bucle *for* y la función *isintance*, esta función devuelve verdadero si el objeto especificado es del tipo especificado, en nuestro caso si las capas son del tipo convolución o agrupación.

También obtendremos los nombres de dichas capas. Añadiremos a su respectiva lista las salidas de las capas y sus respectivos nombres con el método *append()*.

```
from tensorflow.keras import layers

layer_outputs = []
layer_names = []
for layer in model.layers:
    if isinstance(layer, (layers.Conv2D, layers.MaxPooling2D)):
        layer_outputs.append(layer.output)
        layer_names.append(layer.name)
activation_model = keras.Model(inputs=model.input, outputs=layer_ou
tputs)
```

A continuación, asignaremos a *activations* las salidas en forma de array del modelo *activation_model* a través de la función *predict()*. También asignaremos la activación de la primera capa convolucional a *first_layer_activation*.

```
activations = activation_model.predict(np.expand_dims(img, axis=0))
first_layer_activation = activations[0]
```

Para visualizar los distintos canales de la activación de la primera capa solo tendremos que ir cambiando el índice según el canal que queramos mostrar. En este caso estaríamos visualizando el quinto canal.

```
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0,:, :, 5], cmap="viridis")
```

Por último, visualizaremos las activaciones intermedias de cada capa en una imagen diferente mediante el siguiente fragmento de código. Para ello, primero obtendremos mediante un bucle *for* el tamaño de cada capa de activación. A continuación, crearemos una rejilla o grid en la que situaremos las activaciones correspondientes mediante *display_grid*. Posteriormente, normalizaremos los valores de las imágenes para que estén en un rango entre 0 y 255. Por último, mostraremos la rejilla para cada capa como se muestra en las figuras 4-1, 4-2, 4-3, 4-4, 4-5 y 4-6.

```
images_per_row = 16
for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]
```

```
    size = layer_activation.shape[1]
    n_cols = n_features // images_per_row
    display_grid = np.zeros(((size + 1) * n_cols - 1,
                             images_per_row * (size + 1) - 1))
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_index = col * images_per_row + row
            channel_image = layer_activation[0,:, :, channel_index]
.copy()
            if channel_image.sum() != 0:
                channel_image -= channel_image.mean()
                channel_image /= channel_image.std()
                channel_image *= 64
                channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype("
uint8")
            display_grid[
                col * (size + 1): (col + 1) * size + col,
                row * (size + 1) : (row + 1) * size + row] = channe
l_image
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.axis("off")
    plt.imshow(display_grid, aspect="auto", cmap="viridis")
```

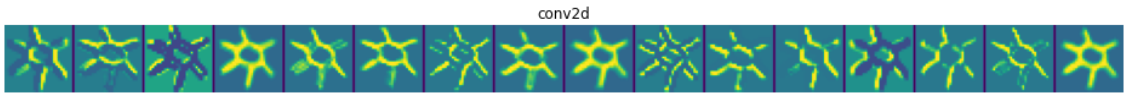


Figura 4-1. Representación de las activaciones intermedias de la primera capa convolucional.

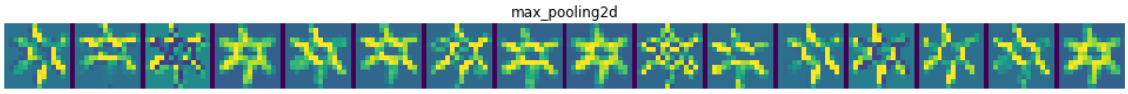


Figura 4-2. Representación de las activaciones intermedias de la primera capa de agrupación.

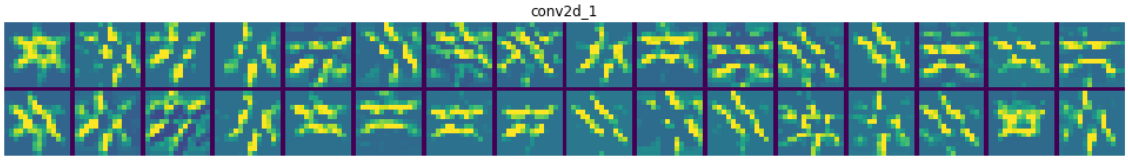


Figura 4-3. Representación de las activaciones intermedias de la segunda capa convolucional.

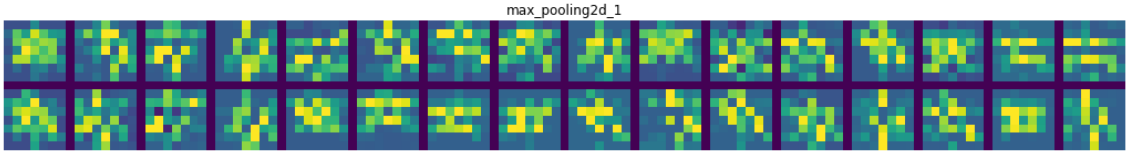


Figura 4-4. Representación de las activaciones intermedias de la segunda capa de agrupación.

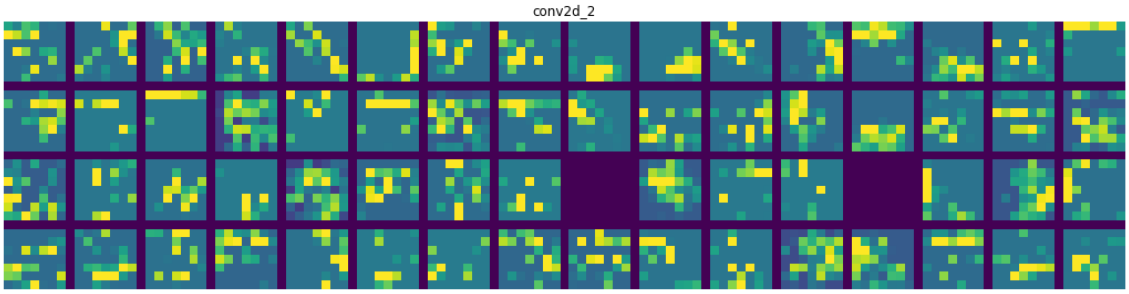


Figura 4-5. Representación de las activaciones intermedias de la tercera capa convolucional.

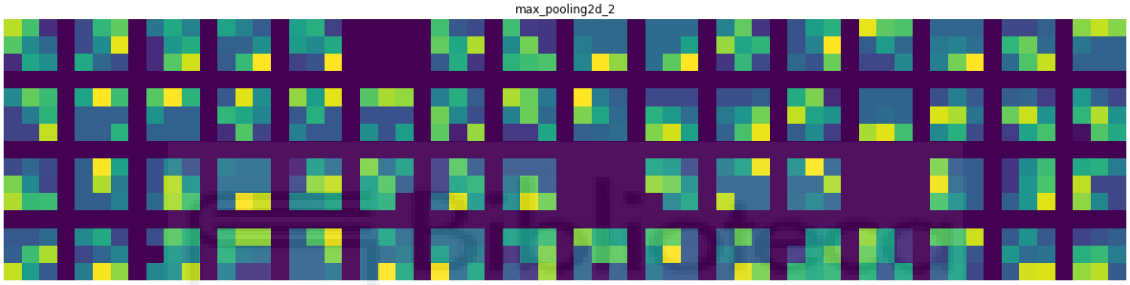


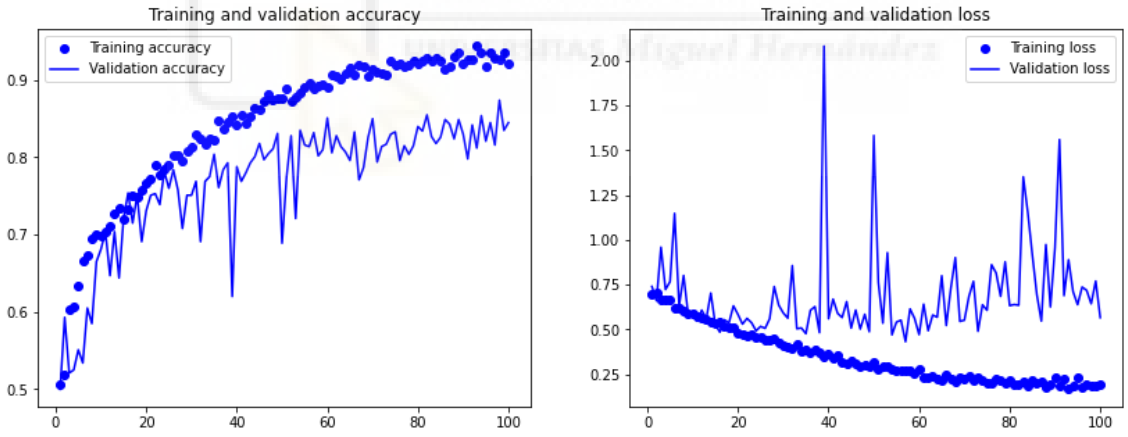
Figura 4-6. Representación de las activaciones intermedias de la tercera capa de agrupación.

CAPÍTULO 5. EXPERIMENTOS REALIZADOS Y ANÁLISIS DE RESULTADOS

En este apartado veremos los resultados obtenidos tras las modificaciones realizadas a los modelos de clasificación de imágenes explicados anteriormente, mostrando las gráficas de resultados y porcentajes obtenidos. También veremos varios ejemplos de activaciones intermedias e intentaremos definir que interpreta la red para distinguir las distintas imágenes mediante la visualización de activaciones intermedias.

5.1. PRUEBAS CLASIFICACIÓN PERROS Y GATOS

En este apartado se muestran los resultados obtenidos en las pruebas realizadas en al apartado 4.1.1. para la aplicación de clasificación de imágenes de perros y gatos. Como referencia partimos de un porcentaje de acierto del 82.3% y un error del 51.94%.



Gráfica 5-1. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la arquitectura sin modificar.

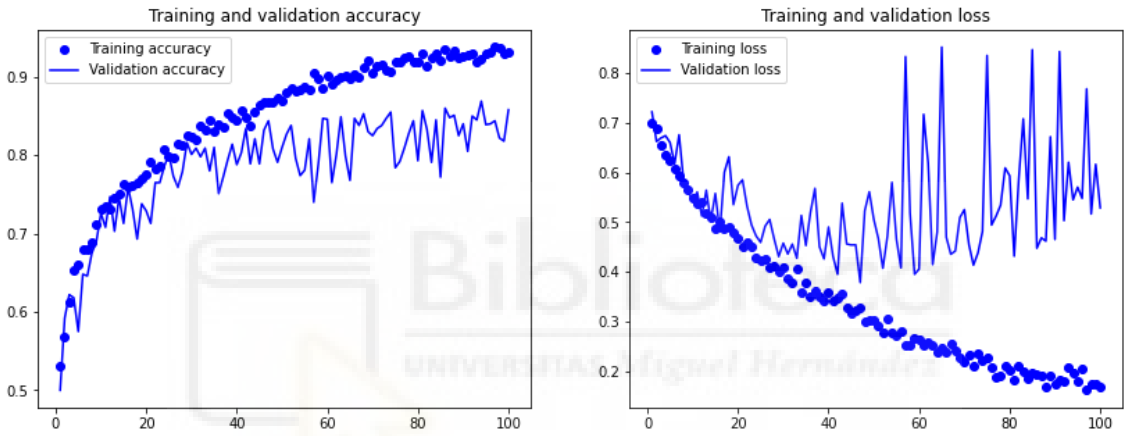
5.1.1. PRUEBA 1

Podemos observar que si reducimos a la mitad el número de filtros de todas las capas convolucionales logramos un porcentaje de acierto del 81.8%, lo que supone un descenso del 0.6% al 82.3% respecto del número de filtros original. Asimismo,

obtenemos un error del 46.3%, lo que supone un descenso del 10.85% al 51.94% del original.

Como se puede observar en la gráfica 5-2, el mayor porcentaje de acierto se alcanza sobre la época 95, en cambio en el modelo original se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 47 mientras que en el modelo original alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 40 mientras que en el modelo original alrededor de la época 45.



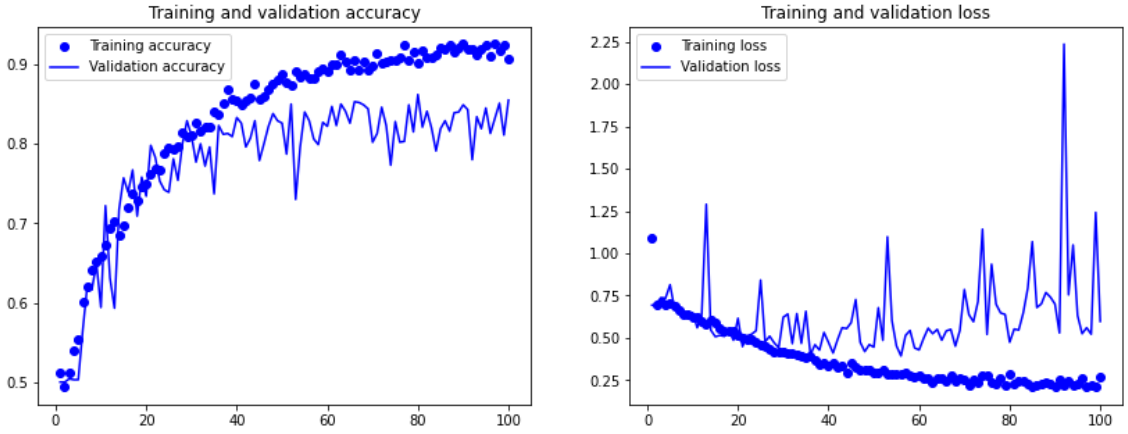
Gráfica 5-2. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 1.

5.1.2. PRUEBA 2

Podemos observar que si duplicamos el número de filtros de todas las capas convolucionales obtenemos un porcentaje de acierto del 81.2%, lo que supone un descenso del 1.34% al 82.3% respecto del número de filtros original. Asimismo, obtenemos un error del 48.68%, lo que supone un descenso del 6.28% al 51.94% del original.

Como se puede observar en la gráfica 5-3, el mayor porcentaje de acierto se alcanza sobre la época 80, en cambio en el modelo original se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se logra alrededor de la época 55 mientras que en el modelo original alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 40 mientras que en el modelo original alrededor de la época 45.



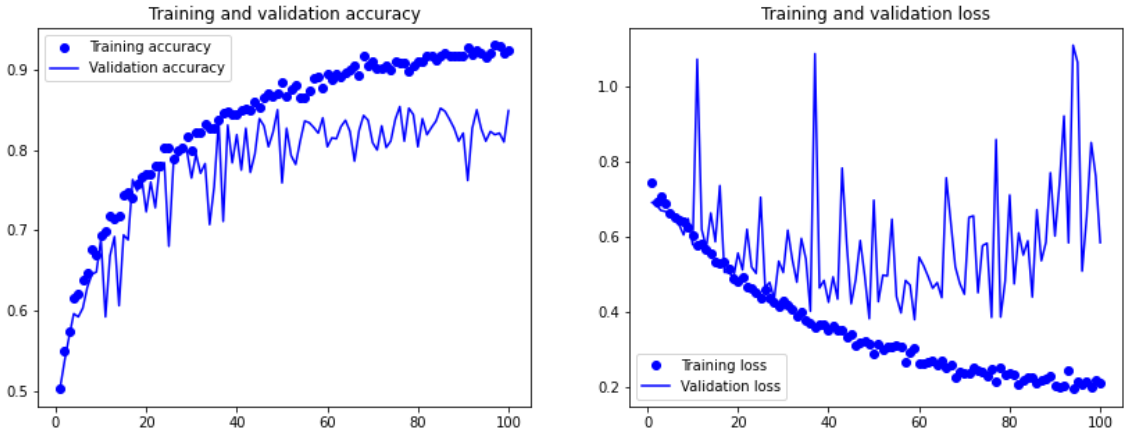
Gráfica 5-3. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 2.

5.1.3. PRUEBA 3

Podemos observar que si añadimos a la arquitectura una capa densa con cien neuronas y función de activación ReLU, sin modificar el número de filtros inicial de las capas convolucionales alcanzamos una precisión del 83.35%, lo que supone un aumento del 1.28% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 41.58%, lo que supone un descenso del 19.95% al 51.94%.

Como se puede observar en la gráfica 5-4, el mayor porcentaje de acierto se alcanza sobre la época 76, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 60 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50 mientras que en el modelo sin modificar alrededor de la época 45.



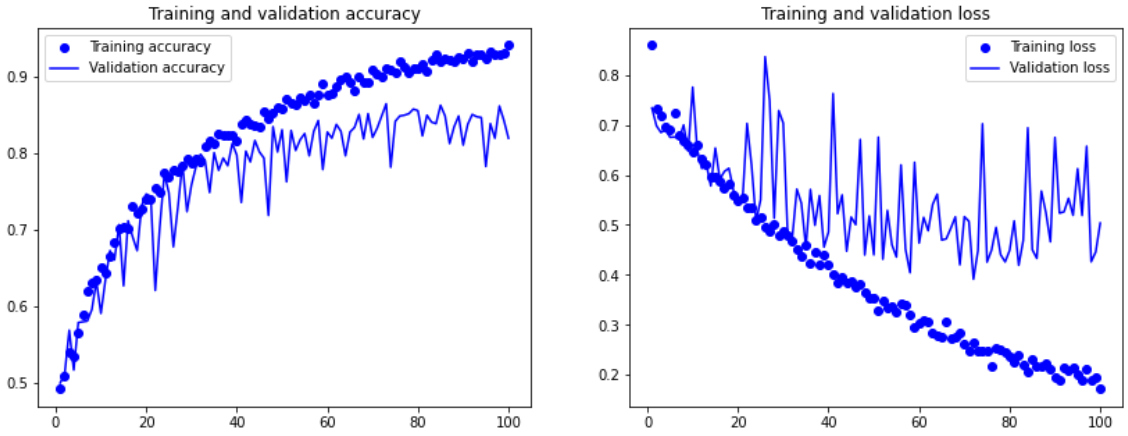
Gráfica 5-4. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 3.

5.1.4. PRUEBA 4

Podemos observar que si le aplicamos a la capa densa añadida a la modificación 3 la regularización L2 con un valor de regularización de 0.001, manteniendo el número de neuronas a cien, logramos un porcentaje de acierto del 85.1%, lo que supone un aumento del 3.4% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 42.42%, lo que supone un descenso del 18.33% al 51.94%.

Como se puede observar en la gráfica 5-5, el mayor porcentaje de acierto se alcanza sobre la época 70, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se logra alrededor de la época 72 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50 mientras que en el modelo sin modificar alrededor de la época 45.



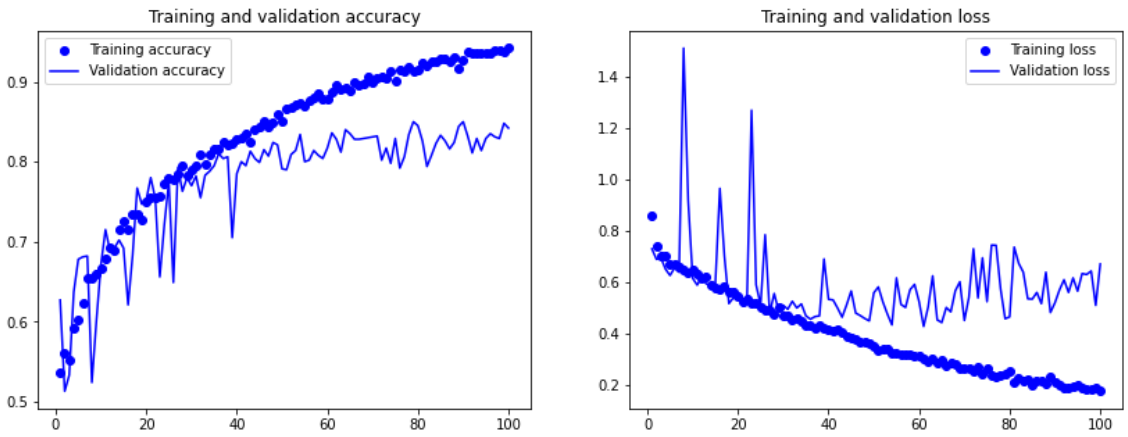
Gráfica 5-5. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 4.

5.1.5. PRUEBA 5

Podemos observar que variando en la modificación anterior el número de neuronas de la capa densa a 250 neuronas obtenemos una precisión de 81.95%, lo que supone un descenso del 0.43% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 42.95%, lo que supone un descenso del 17.31% al 51.94%.

Como se puede observar en la gráfica 5-6, el mayor porcentaje de acierto se alcanza sobre la época 79, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error alrededor se alcanza alrededor de la época 62 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 45, de igual manera que en el modelo sin modificar.



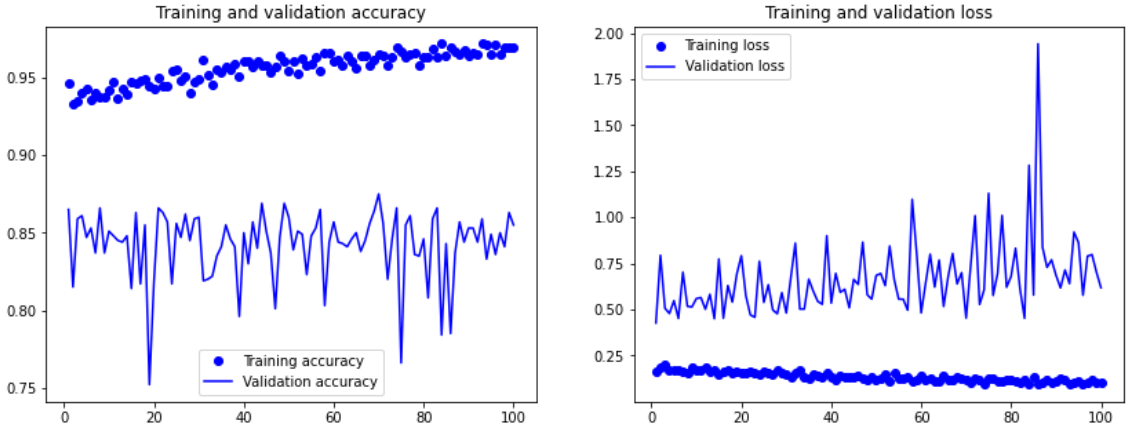
Gráfica 5-6. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 5.

5.1.6. PRUEBA 6

Podemos observar que si modificamos el número de neuronas de la capa densa a 50 obtenemos un porcentaje de acierto del 85.5%, lo que supone un aumento del 3.89% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 45.37%, lo que supone un descenso del 12.65% al 51.94%.

Como se puede observar en la gráfica 5-7, el mayor porcentaje de acierto se alcanza sobre la época 70, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 1 mientras que en el modelo sin modificar alrededor de la época 68.

También podemos comprobar que se produce sobreajuste desde el principio.



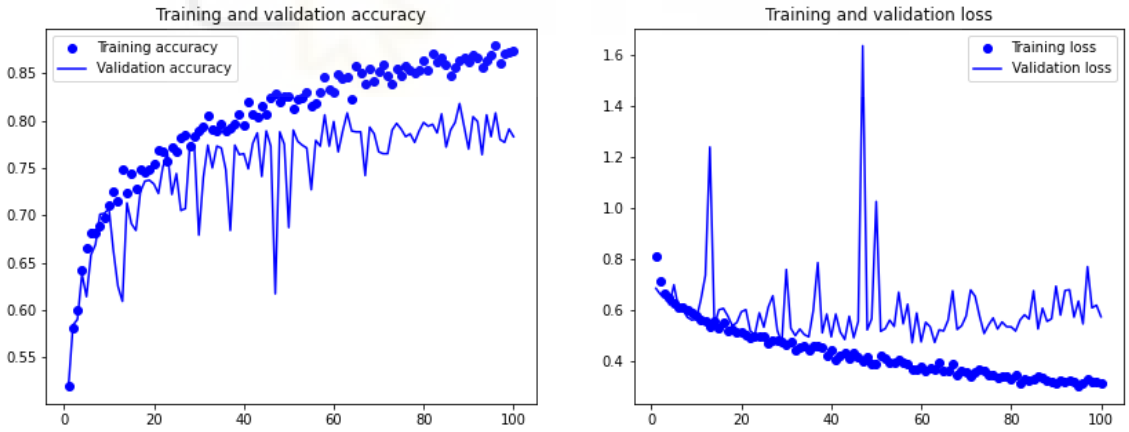
Gráfica 5-7. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 6.

5.1.7. PRUEBA 7

Podemos observar que si reducimos el número de capas de nuestra arquitectura inicial, pasando a tener tres capas de convolución y dos de agrupación alternándose sucesivamente obtenemos un porcentaje de acierto del 80.3%, lo que supone un descenso del 2.43% al 82.3% respecto del número de capas original. Asimismo, obtenemos un error del 49.03%, lo que supone un descenso del 5.6% al 51.94% del original.

Como se puede observar en la gráfica 5-8, el mayor porcentaje de acierto se alcanza sobre la época 88, en cambio en el modelo original se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 58 mientras que en el modelo original alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 35 o 40, mientras que en el modelo original alrededor de la época 45.



Gráfica 5-8. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 7.

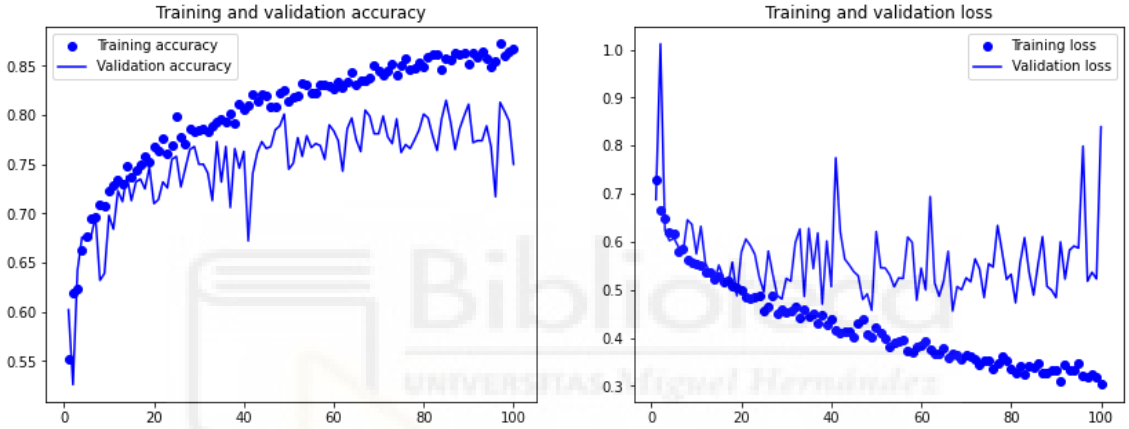
5.1.8. PRUEBA 8

Podemos observar que si reducimos el número de filtros a la mitad de la modificación anterior logramos una precisión del 79.85%, lo que supone un descenso del 2.98% al

82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 49.67%, lo que supone un descenso del 4.37% al 51.94%.

Como se puede observar en la gráfica 5-9, el mayor porcentaje de acierto se alcanza sobre la época 85, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 66 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 35 mientras que en el modelo sin modificar alrededor de la época 45.



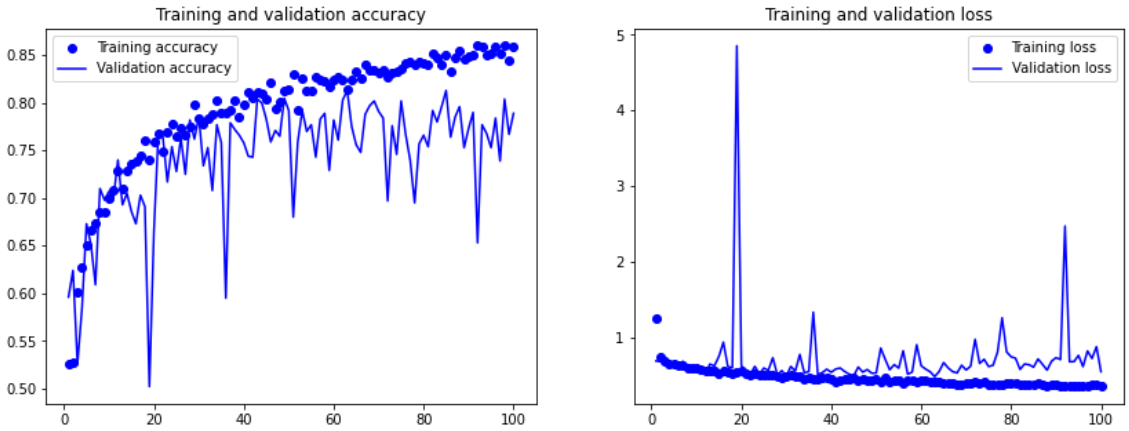
Gráfica 5-9. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 8.

5.1.9. PRUEBA 9

Podemos observar que si duplicamos el número de filtros de la modificación 7 alcanzamos una precisión del 81%, lo que supone un descenso del 1.58% al 82.3% respecto del número de filtros original. Asimismo, obtenemos un error del 46.64%, lo que supone un descenso del 10.2% al 51.94%.

Como se puede observar en la gráfica 5-10, el mayor porcentaje de acierto se alcanza sobre la época 85, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se logra alrededor la época 63 mientras que en el modelo sin modificar alrededor de la época 68.

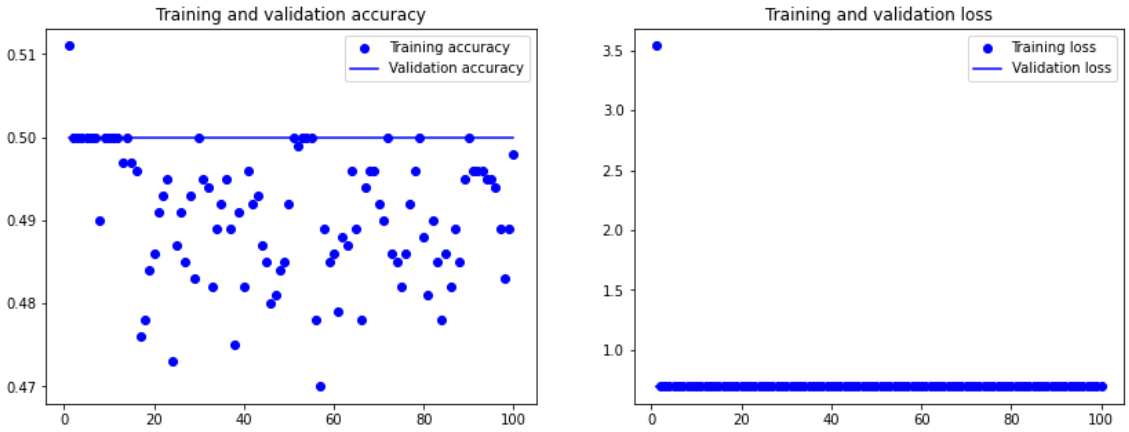
En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50 mientras que en el modelo sin modificar alrededor de la época 45.



Gráfica 5-10. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 9.

5.1.10. PRUEBA 10

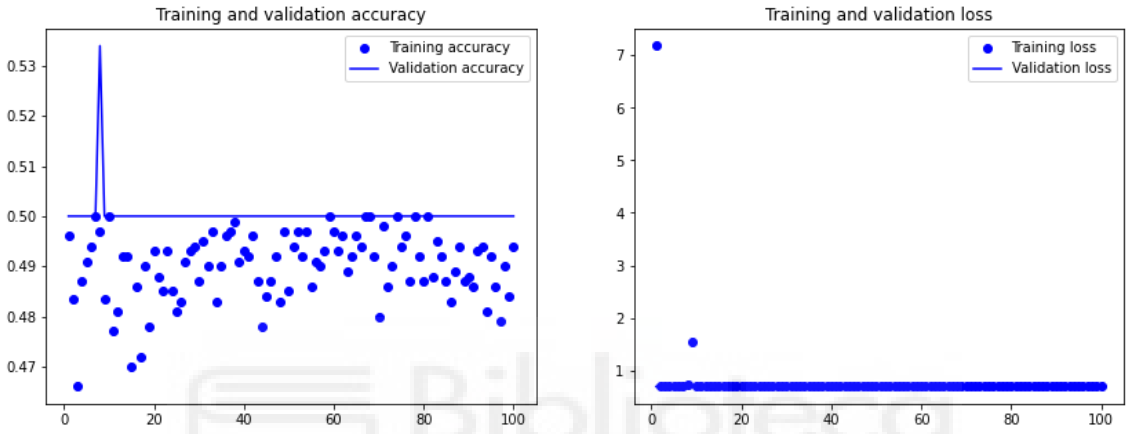
Podemos observar que si añadimos a la modificación 7 una capa densa con 100 neuronas y función de activación ReLU, obtenemos un porcentaje de acierto del 50%, lo que supone un descenso del 39.25% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 69.31%, lo que supone un aumento del 33.44% al 51.94% del original. También podemos comprobar que tenemos sobreajuste desde el principio.



Gráfica 5-11. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 10.

5.1.11. PRUEBA 11

Podemos observar que si variamos el número de neuronas de la capa densa a 500 alcanzamos una precisión del 54.65%, lo que supone un descenso del 33.6% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 69.31%, lo que supone un aumento del 33.44% al 51.94%. También podemos comprobar que tenemos sobreajuste desde el principio.



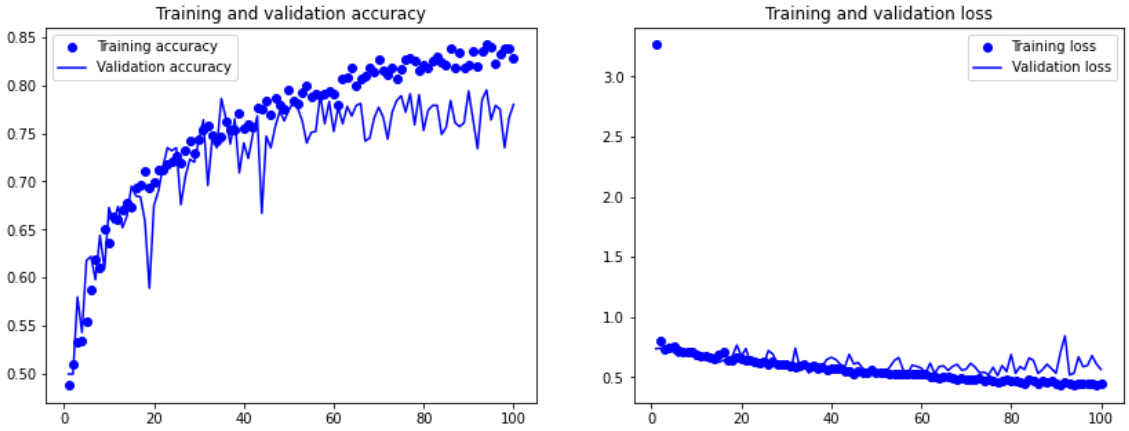
Gráfica 5-12. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 11.

5.1.12. PRUEBA 12

Podemos observar que si aplicamos la regularización L2 a la modificación 10, logramos un porcentaje de acierto del 78.5%, lo que supone un descenso del 4.62% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 55.22% lo que supone un aumento del 6.31% al 51.94%.

Como se puede observar en la gráfica 5-13, el mayor porcentaje de acierto se alcanza sobre la época 90, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 75 mientras que en el modelo sin modificar alrededor de la época 68.

También podemos comprobar que se produce una reducción en el sobreajuste, debido a la utilización de la técnica de regularización L2. Este se produce alrededor de la época 60 mientras que en el modelo original alrededor de la época 45.



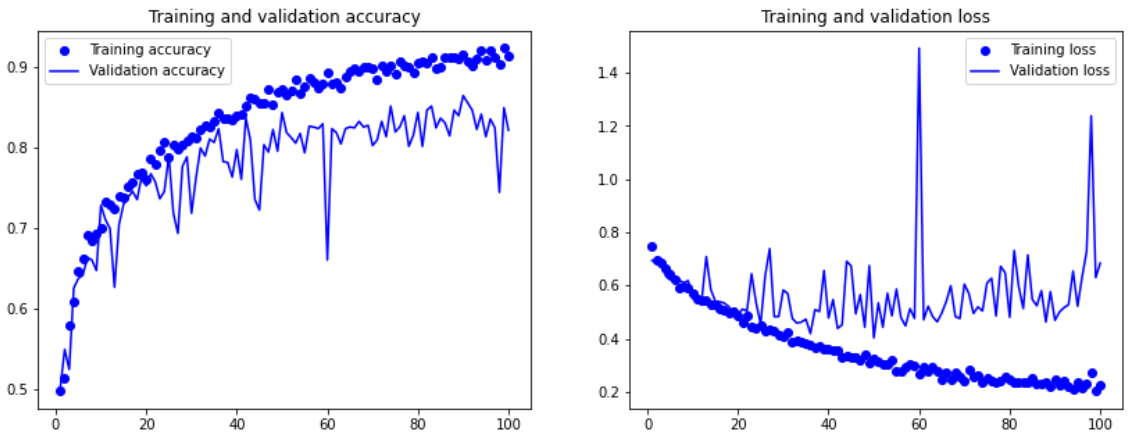
Gráfica 5-13. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 12.

5.1.13. PRUEBA 13

Podemos observar que si probamos con cuatro capas de convolución y tres de agrupación alternándose sucesivamente, siendo el número de filtros de las capas convolucionales 32, 64, 128 y 256 respectivamente, logramos un porcentaje de acierto del 81.85%, lo que supone un descenso del 0.55% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 42.1%, lo que supone un descenso del 18.94% al 51.94%.

Como se puede observar en la gráfica 5-14, el mayor porcentaje de acierto se alcanza sobre la época 90, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 50 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 35 mientras que en el modelo sin modificar alrededor de la época 45.



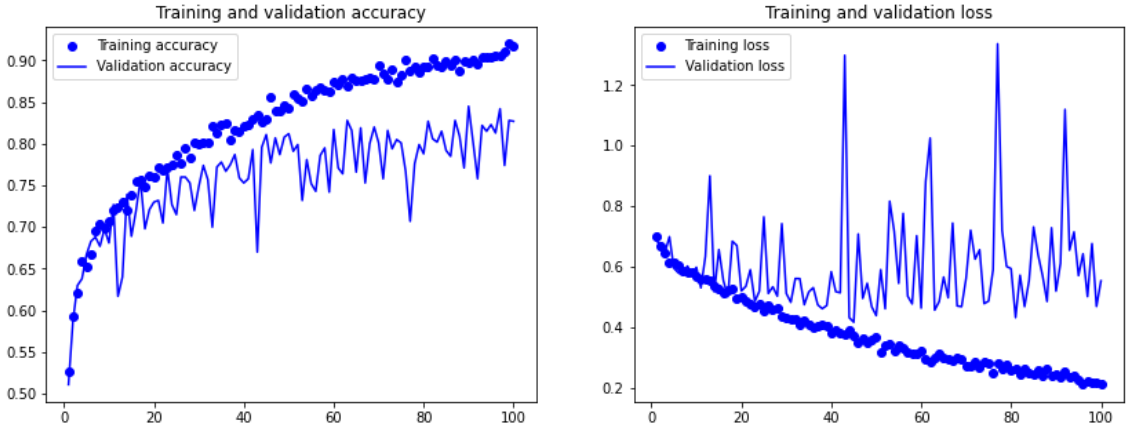
Gráfica 5-14. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 13.

5.1.14. PRUEBA 14

Podemos observar que si disminuimos el número de filtros a la mitad de la modificación anterior conseguimos un porcentaje de acierto del 80.1%, lo que supone un descenso del 2.67% al 82.3% respecto del número de capas original. Asimismo, obtenemos un error del 44.5% lo que supone un descenso del 14.32% al 51.94%.

Como se puede observar en la gráfica 5-15, el mayor porcentaje de acierto se alcanza sobre la época 90, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se logra alrededor de la época 45 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50 mientras que en el modelo sin modificar alrededor de la época 45.



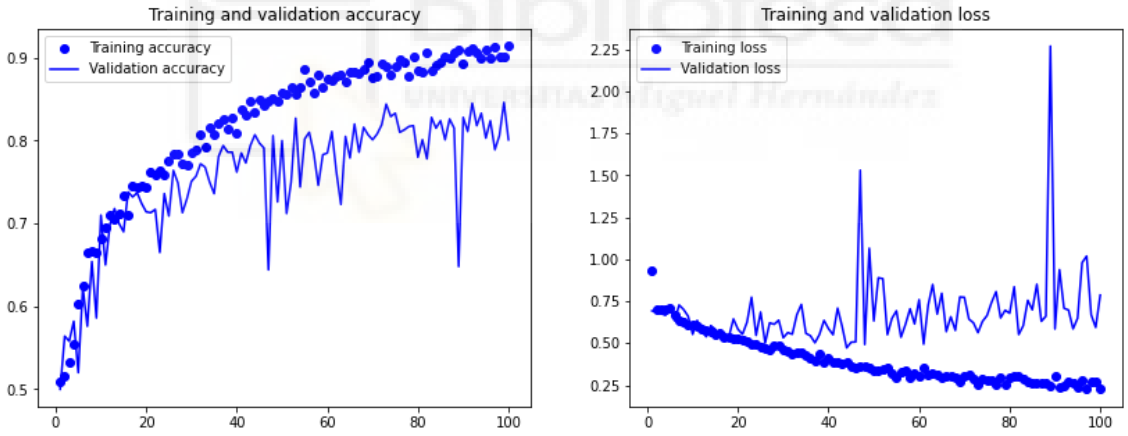
Gráfica 5-15. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 14.

5.1.15. PRUEBA 15

Podemos observar que si duplicamos el número de filtros de la modificación 13, logramos un porcentaje de acierto del 80.3%, lo que supone un descenso del 2.43% al 82.3% respecto del número de filtros original. Asimismo, obtenemos un error del 47.56%, lo que supone un descenso del 8.43% al 51.94%.

Como se puede observar en la gráfica 5-16, el mayor porcentaje de acierto se alcanza sobre la época 73, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 45 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 45, de igual manera que en el modelo sin modificar.



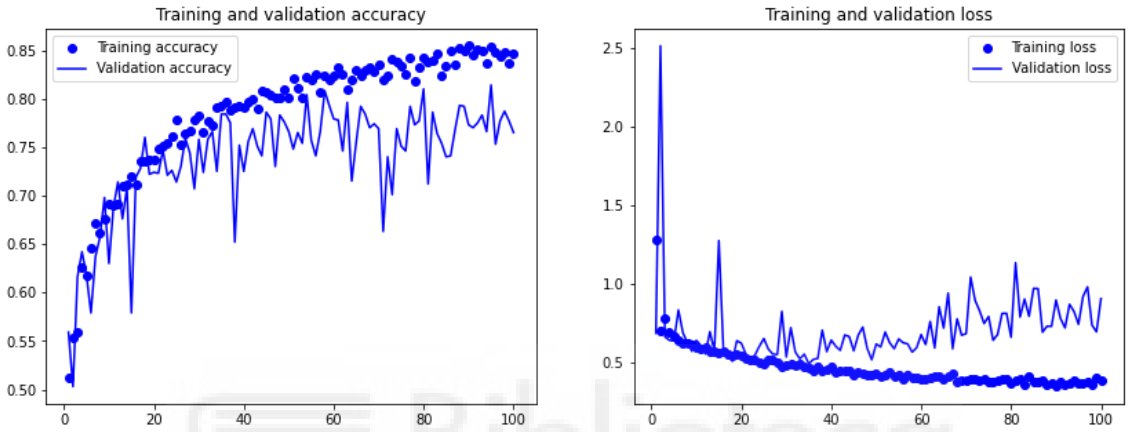
Gráfica 5-16. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 15.

5.1.16. PRUEBA 16

Podemos observar que si se prueba otra combinación de la arquitectura anterior, esta vez dos capas de convolución seguidas de una de agrupación y así sucesivamente y manteniendo el número de filtros de la modificación anterior logramos un porcentaje de acierto del 76.05%, lo que supone un descenso del 7.59% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 52.14%, lo que supone un aumento del 0.39% al 51.94%.

Como se puede observar en la gráfica 5-17, el mayor porcentaje de acierto se alcanza sobre la época 95, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 35 mientras que en el modelo sin modificar alrededor de la época 68.

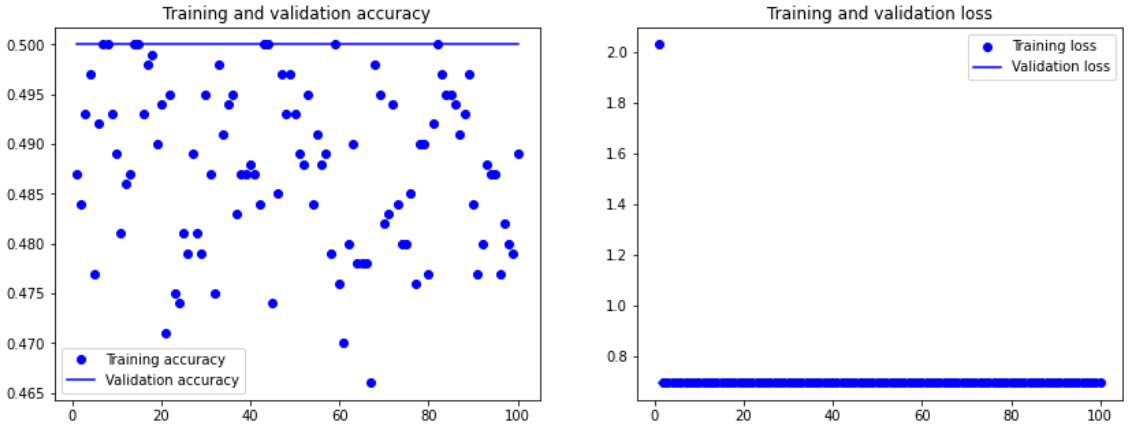
En este caso podemos comprobar que se produce sobreajuste alrededor de la época 60 mientras que en el modelo sin modificar alrededor de la época 45.



Gráfica 5-17. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 16.

5.1.17. PRUEBA 17

Podemos observar que si probamos con cuatro capas convolucionales y dos de agrupación, alternándose excepto las dos últimas que serán dos capas de convolución seguidas, añadimos una capa densa con doscientas cincuenta neuronas y función de activación ReLU y definimos el número de filtros de las capas convolucionales como 32, 64, 128 y 128 respectivamente, logramos un porcentaje de acierto del 50%, lo que supone un descenso del 39.25% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 69.31%, lo que supone un aumento del 33.44% al 51.94%. También podemos comprobar que se produce sobreajuste desde el principio.



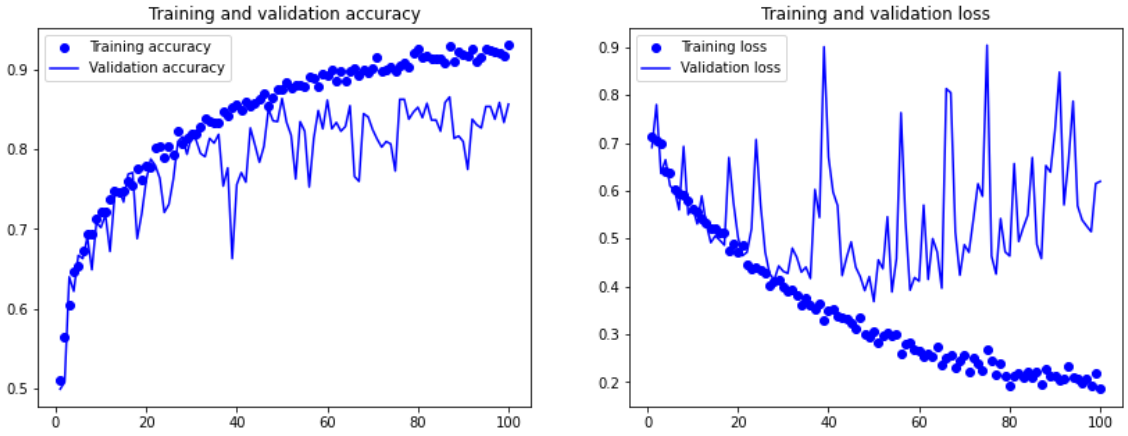
Gráfica 5-18. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 17.

5.1.18. PRUEBA 18

Podemos observar que probamos con cuatro capas convolucionales y cuatro de agrupación alternándose, siendo el número de filtros 32, 64, 128 y 256 respectivamente. Así, conseguimos una precisión del 84.5%, lo que supone un aumento del 2.67% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 39.91%, lo que supone un descenso del 23.16% al 51.94%.

Como se puede observar en la gráfica 5-19, el mayor porcentaje de acierto se alcanza sobre la época 50, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 50 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 55 o 60, mientras que en el modelo sin modificar alrededor de la época 45.



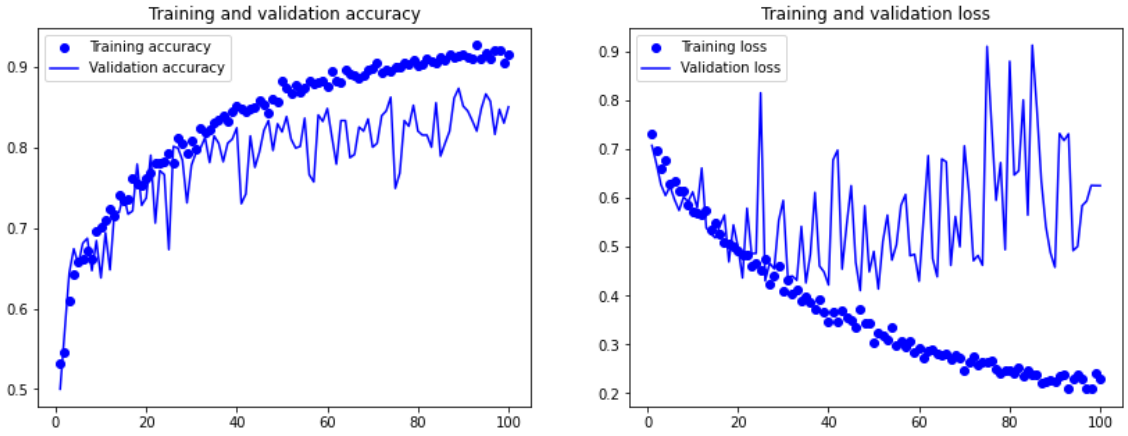
Gráfica 5-19. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 18.

5.1.19. PRUEBA 19

Podemos observar que si duplicamos el número de filtros de la anterior arquitectura obtenemos una precisión del 82.35%, lo que supone un aumento del 0.06% al 82.3% respecto del número de filtros original. Asimismo, obtenemos un error del 44.28%, lo que supone un descenso del 14.75% al 51.94%.

Como se puede observar en la gráfica 5-20, el mayor porcentaje de acierto se alcanza sobre la época 88, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 47 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 35 o 40, mientras que en el modelo sin modificar alrededor de la época 45.



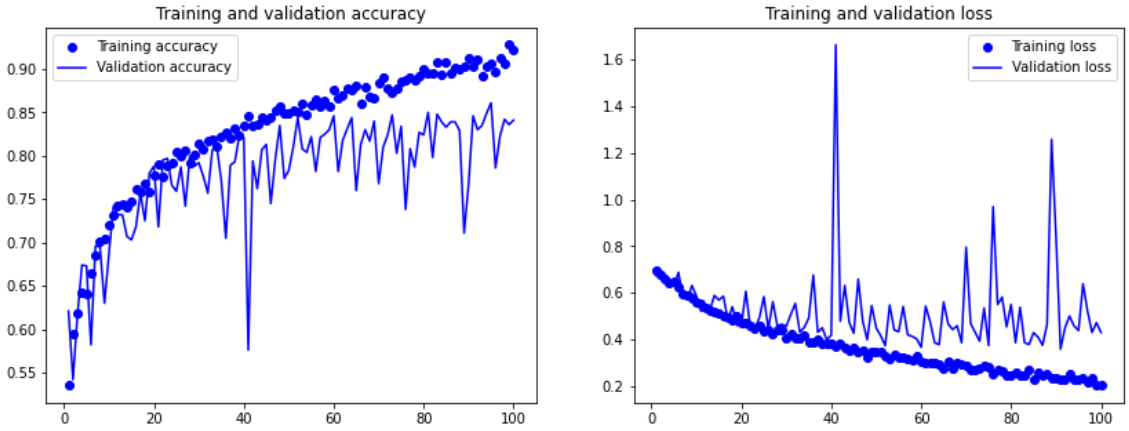
Gráfica 5-20. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 19.

5.1.20. PRUEBA 20

Podemos observar que si disminuimos el número de filtros a la mitad de la modificación 18, logramos una precisión del 82.95%, lo que supone un aumento del 0.79% al 82.3% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 44.07%, lo que supone un descenso del 15.15% al 51.94%.

Como se puede observar en la gráfica 5-21, el mayor porcentaje de acierto se alcanza sobre la época 95, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 60 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 60, mientras que en el modelo sin modificar alrededor de la época 45.



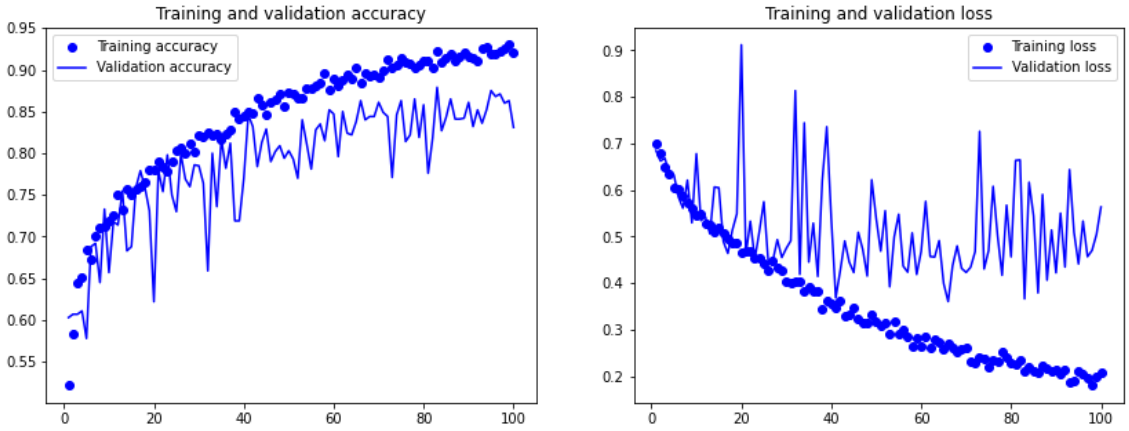
Gráfica 5-21. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 20.

5.1.21. PRUEBA 21

Podemos observar que si probamos ahora variando los filtros, modificándolos de manera que la primera capa convolucional tenga 32, la segunda 64 y las dos últimas 128 filtros, alcanzamos un porcentaje de acierto del 85%, lo que supone un aumento del 3.28% al 82.3% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 40.2%, lo que supone un descenso del 22.6% al 51.94%.

Como se puede observar en la gráfica 5-22, el mayor porcentaje de acierto se alcanza sobre la época 82, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 65 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50, mientras que en el modelo sin modificar alrededor de la época 45.



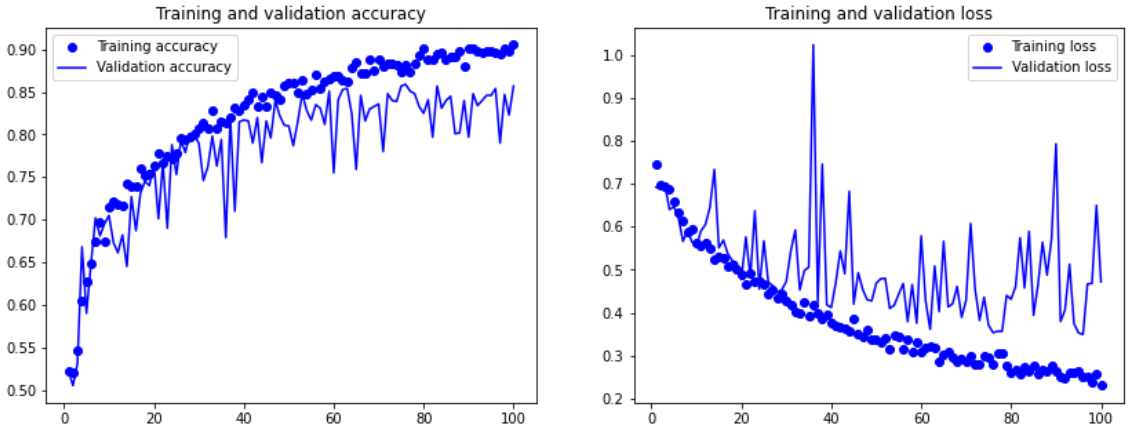
Gráfica 5-22. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 21.

5.1.22. PRUEBA 22

Podemos observar que si añadimos a la modificación 18 una capa densa con cien neuronas y función de activación ReLU, logramos un porcentaje de acierto del 84.15%, lo que supone un aumento del 2.25% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 38.14%, lo que supone un descenso del 26.57% al 51.94%.

Como se puede observar en la gráfica 5-23, el mayor porcentaje de acierto se alcanza sobre la época 76, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 95 mientras que en el modelo sin modificar alrededor de la época 68.

También podemos comprobar que se produce una reducción en el sobreajuste, debido a agregarle una capa densa. Este se produce alrededor de la época 65 mientras que en el modelo original alrededor de la época 45.



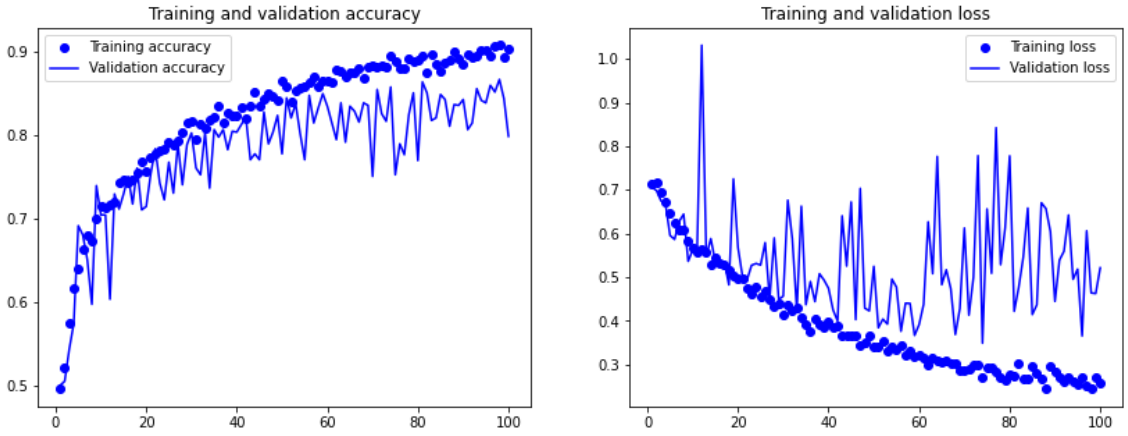
Gráfica 5-23. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 22.

5.1.23. PRUEBA 23

Podemos observar que si probamos cambiando el número de neuronas de la capa densa a doscientas cincuenta, alcanzamos así un porcentaje de acierto del 84%, lo que supone un aumento del 2.07% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 37.73%, lo que supone un descenso del 27.36% al 51.94%.

Como se puede observar en la gráfica 5-24, el mayor porcentaje de acierto se alcanza sobre la época 98, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 75 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 60, mientras que en el modelo sin modificar alrededor de la época 45.



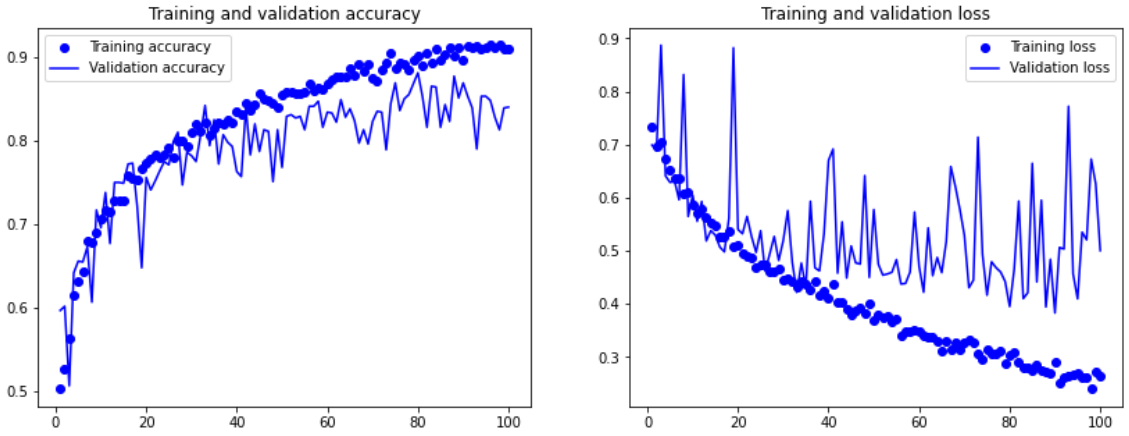
Gráfica 5-24. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 23.

5.1.24. PRUEBA 24

Podemos observar que si añadimos la técnica de regularización L2 a la capa densa con un valor de regularización de 0.001 y cambiamos el número de neuronas de dicha capa a 64, y modificamos el parámetro *rate* de la capa dropout a 0.2 conseguimos una precisión de 85.3%, lo que supone un aumento del 3.65% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 44.15%, lo que supone un descenso del 15% al 51.94% del original.

Como se puede observar en la gráfica 5-25, el mayor porcentaje de acierto se alcanza sobre la época 80, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 90 mientras que en el modelo sin modificar alrededor de la época 68.

También podemos comprobar que se produce un descenso en el sobreajuste, debido a la aplicación de una serie de regularizaciones. Este se produce alrededor de la época 70 mientras que en el modelo original alrededor de la época 45.



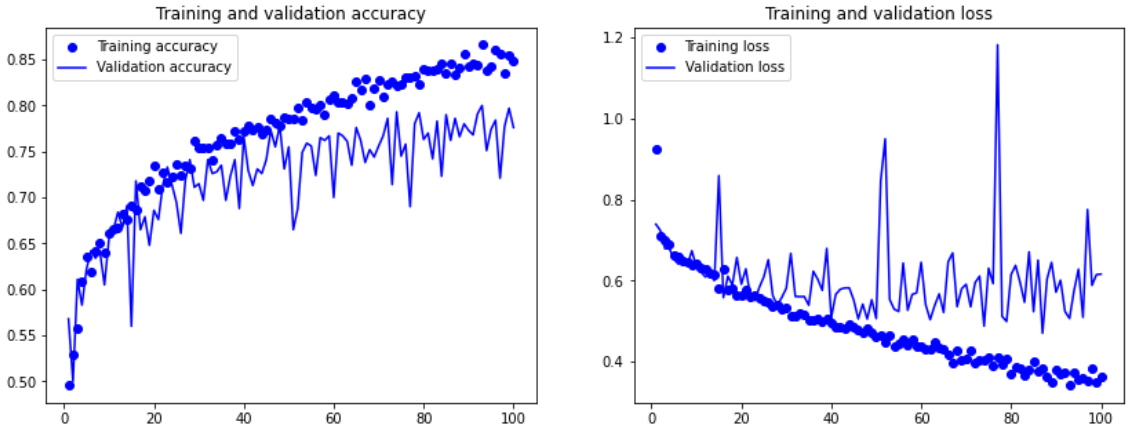
Gráfica 5-25. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 24.

5.1.25. PRUEBA 25

Podemos observar que si añadimos otra capa de convolución y otra de agrupación a la anterior modificación, teniendo cinco capas de convolución y cinco de agrupación; aumentamos el número de neuronas de la capa densa a 512 y cambiamos el número de filtros a 32, 64, 64, 128 y 128 respectivamente, obtenemos un porcentaje de acierto del 80%, lo que supone un descenso del 2.79% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 46.38%, lo que supone un descenso del 10.7% al 51.94% del original.

Como se puede observar en la gráfica 5-26, el mayor porcentaje de acierto se alcanza sobre la época 93, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 88 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50, mientras que en el modelo sin modificar alrededor de la época 45.



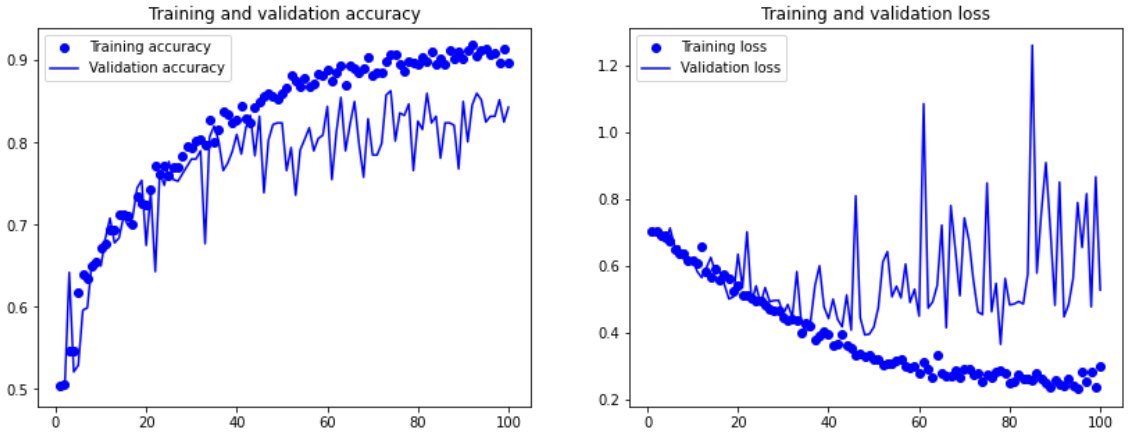
Gráfica 5-26. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 25.

5.1.26. PRUEBA 26

Podemos observar que si quitamos la capa densa que habíamos añadido a la modificación anterior y por tanto, la regularización L2, cambiamos el número de filtros a 64, 128, 256, 256 y 256 respectivamente y modificamos el parámetro *rate* de la capa dropout a 0.5, logramos un porcentaje de acierto del 83.8%, lo que supone un aumento del 1.82% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 46.6%, lo que supone un descenso del 10.28% al 51.94% del original.

Como se puede observar en la gráfica 5-27, el mayor porcentaje de acierto se alcanza sobre la época 75, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 78 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 50, mientras que en el modelo sin modificar alrededor de la época 45.



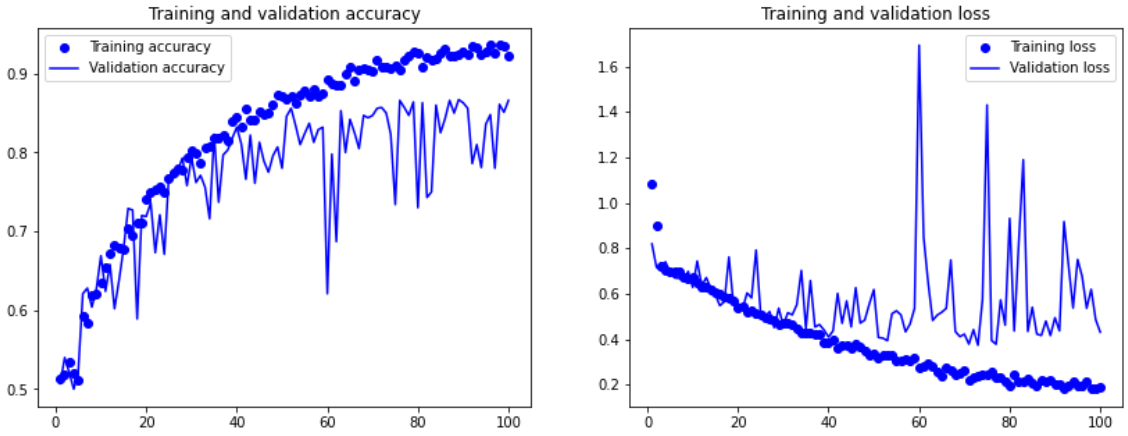
Gráfica 5-27. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 26.

5.1.27. PRUEBA 27

Podemos observar que si añadimos a la modificación anterior una capa densa con 512 neuronas y función de activación ReLU y además, le añadimos la regularización L2 con un valor de regularización de 0.001 obtenemos un porcentaje de acierto del 85%, lo que supone un aumento del 3.28% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 46.07%, lo que supone un descenso del 11.3% al 51.94%.

Como se puede observar en la gráfica 5-28, el mayor porcentaje de acierto se alcanza sobre la época 76, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 73 mientras que en el modelo sin modificar alrededor de la época 68.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 55, mientras que en el modelo sin modificar alrededor de la época 45.



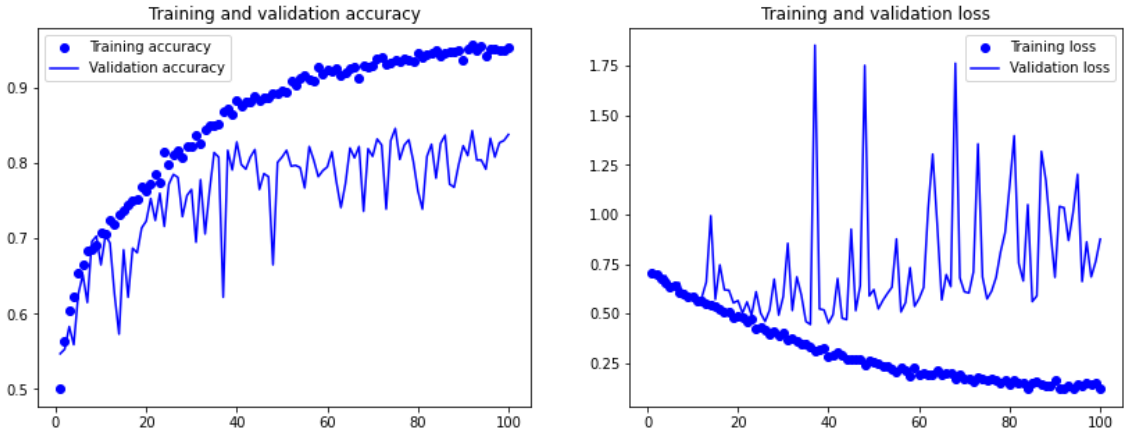
Gráfica 5-28. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 27.

5.1.28. PRUEBA 28

Podemos observar que si modificamos el número de filtros de la modificación anterior a 32, 64, 128, 256 y 256 filtros respectivamente. Además, cambiamos el valor del parámetro *rate* de la capa dropout a 0.2 y quitamos la capa densa con la regularización L2, alcanzamos un porcentaje de acierto del 81.45%, lo que supone un descenso del 1.03% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 46.45%, lo que supone un descenso del 10.57% al 51.94%.

Como se puede observar en la gráfica 5-29, el mayor porcentaje de acierto se alcanza sobre la época 75, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 36 mientras que en el modelo sin modificar alrededor de la época 68.

También podemos comprobar que se produce un aumento en el sobreajuste, debido a quitar la regularización L2 y reducir el número de filtros. Este se produce alrededor de la época 40 mientras que en el modelo original alrededor de la época 45.

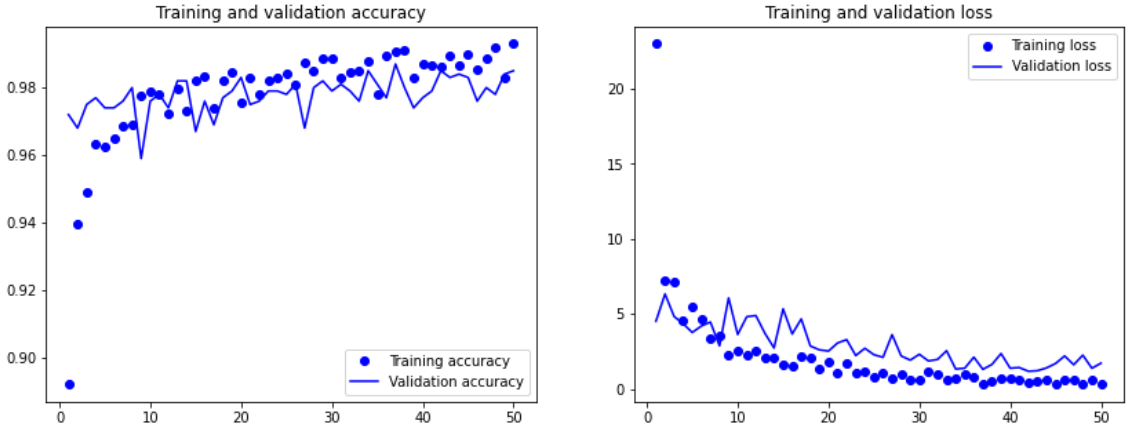


Gráfica 5-29. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 28.

5.1.29. PRUEBA 29

Podemos observar que si aplicamos la técnica de extracción de características con aumento de datos obtenemos un porcentaje de acierto del 97.5%, lo que supone un aumento del 18.46% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 200%, lo que supone un descenso del 285.06% al 51.94%.

Como se puede observar en la gráfica 5-30, el mayor porcentaje de acierto se alcanza sobre la época 37, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 42 mientras que en el modelo sin modificar alrededor de la época 68.

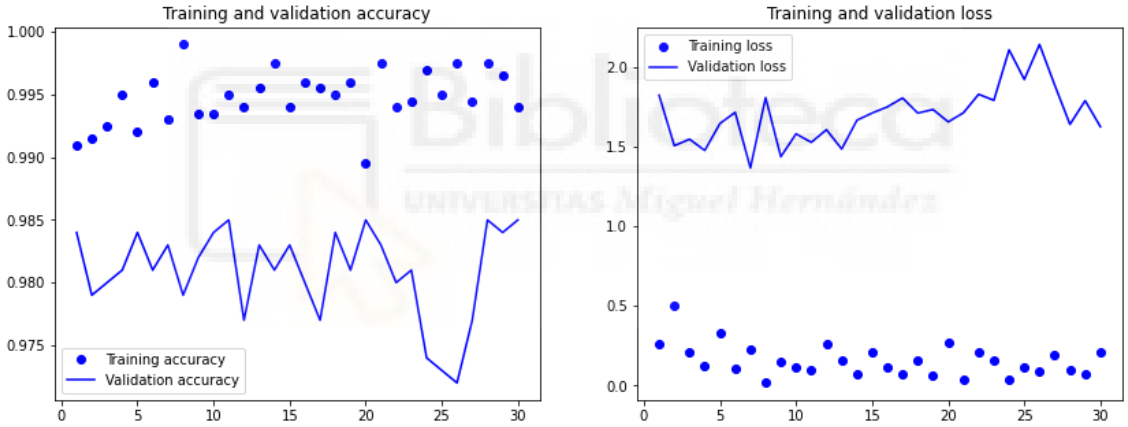


Gráfica 5-30. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 29.

5.1.30. PRUEBA 30

Podemos observar que si aplicamos la técnica de fine tuning, obtenemos un porcentaje de acierto del 98%, lo que supone un aumento del 19.08% al 82.3% respecto de la arquitectura sin modificar. Asimismo, obtenemos un error del 146%, lo que supone un descenso del 181.09% al 51.94%.

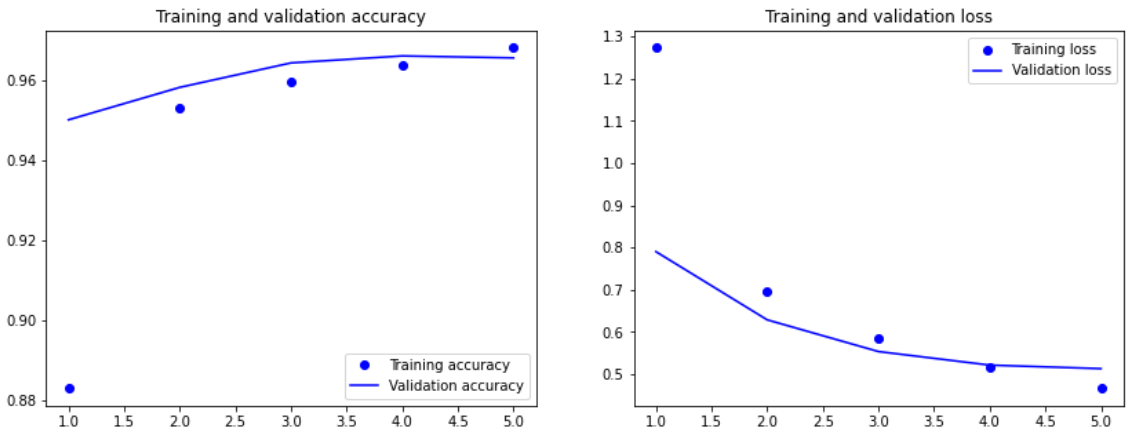
Como se puede observar en la gráfica 5-31, el mayor porcentaje de acierto se alcanza sobre la época 12, en cambio en el modelo sin modificar se produce alrededor de la época 98. Además, observamos que el menor porcentaje de error se alcanza alrededor de la época 8 mientras que en el modelo sin modificar alrededor de la época 68. También podemos comprobar que tenemos sobreajuste desde el principio.



Gráfica 5-31. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 30.

5.2. PRUEBAS RECONOCIMIENTO DE DIBUJOS

En este apartado se muestran los resultados obtenidos en las pruebas realizadas en al apartado 4.2.1. para la aplicación de reconocimiento de dibujos. Como referencia utilizaremos la red entrenada con 25 clases, así partimos de un porcentaje de acierto del 96.45% y un error del 49.86%.

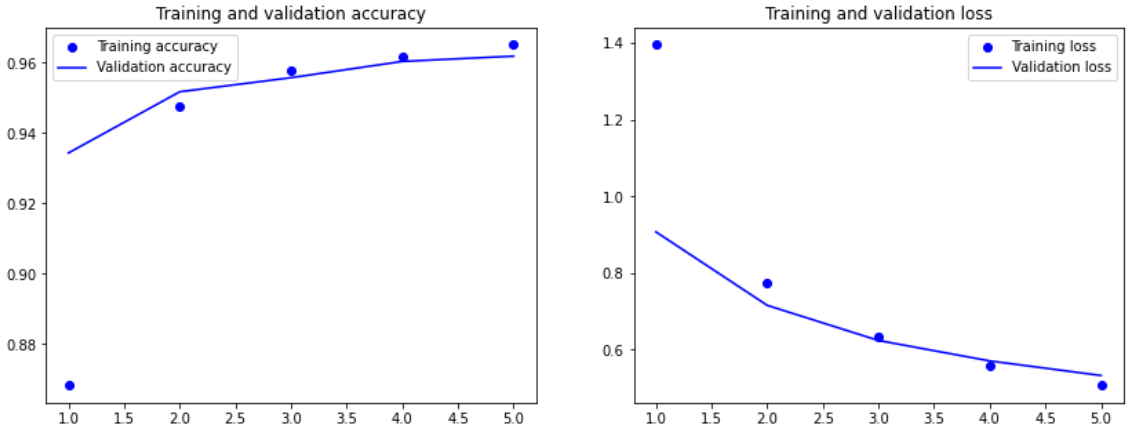


Gráfica 5-32. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación del sistema de reconocimiento de dibujos entrenado con 25 clases.

5.2.1. PRUEBA 1

Podemos observar que si cambiamos el número de filtros de las capas convolucionales, excepto la primera, y la penúltima capa densa a 16, 64 y 128 filtros respectivamente logramos una precisión del 96.4%, lo que supone un descenso del 0.05% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 53.23%, lo que supone un aumento del 6.76% al 49.86%.

Como se puede observar en la gráfica 5-33, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.



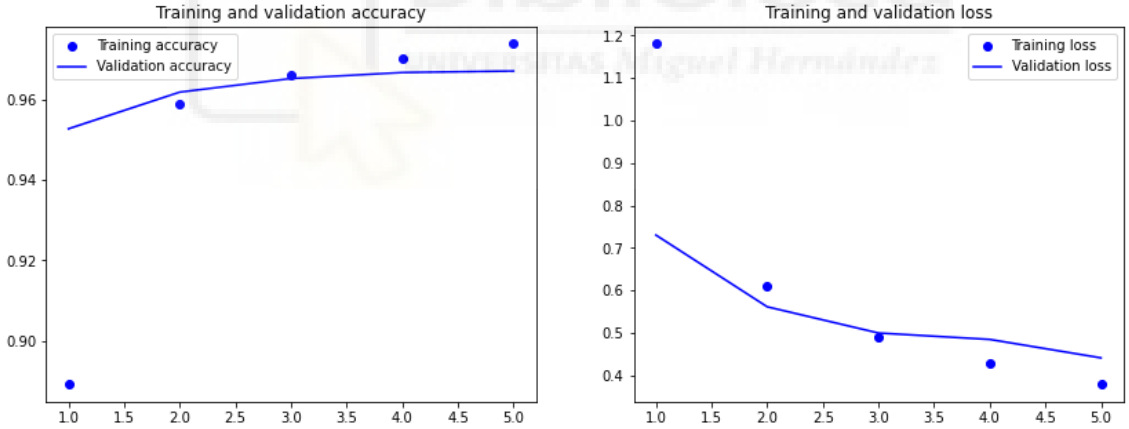
Gráfica 5-33. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 1 del sistema de reconocimiento de dibujos.

5.2.2. PRUEBA 2

Podemos observar que si cambiamos en la prueba 1 los filtros de la segunda capa convolucional a 64 y la tercera capa convolucional a 128, obtenemos una precisión del 96.75%, lo que supone un aumento del 0.31% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 44.96%, lo que supone un aumento del 9.83% al 49.86%.

Como se puede observar en la gráfica 5-34, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

En este caso podemos observar un ligera sobreajuste a partir de la época 3.



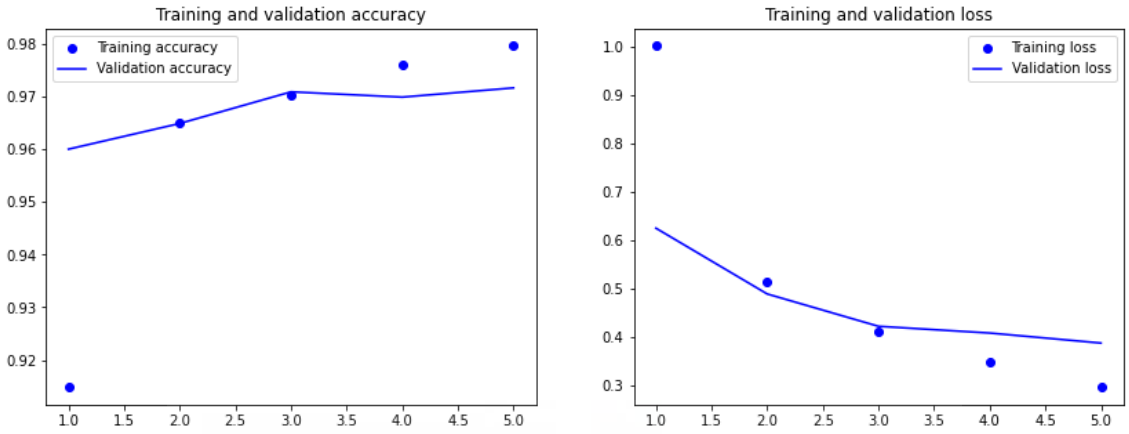
Gráfica 5-34. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 2 del sistema de reconocimiento de dibujos.

5.2.3. PRUEBA 3

Podemos observar que si cambiamos el número de filtros de las capas convolucionales y la penúltima capa densa a 32, 64, 256 y 256 respectivamente, obtenemos una precisión del 97.3%, lo que supone un aumento del 0.88% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 37.79%, lo que supone un descenso del 24.2% al 49.86%.

Como se puede observar en la gráfica 5-35, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

En este caso podemos comprobar que se produce sobreajuste en la época 3.



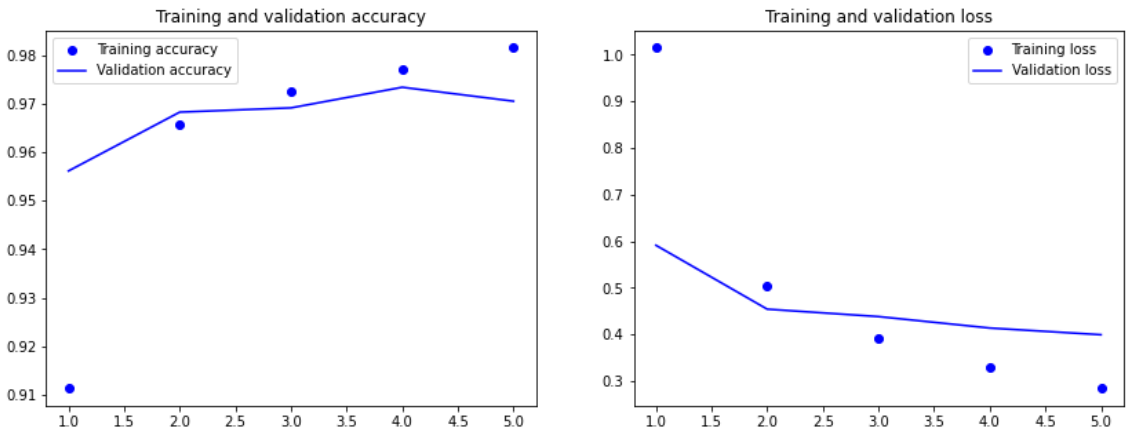
Gráfica 5-35 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 3 del sistema de reconocimiento de dibujos.

5.2.4. PRUEBA 4

Podemos observar que si cambiamos en la prueba 3 el número de filtros de la primera capa convolucional a 64, obtenemos una precisión del 97.02%, lo que supone un aumento del 0.59% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 40.56%, lo que supone un aumento del 18.65% al 49.86%.

Como se puede observar en la gráfica 5-36, el mayor porcentaje de acierto se alcanza en la época 4 y un menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

En este caso podemos comprobar que se produce sobreajuste en la época 4.

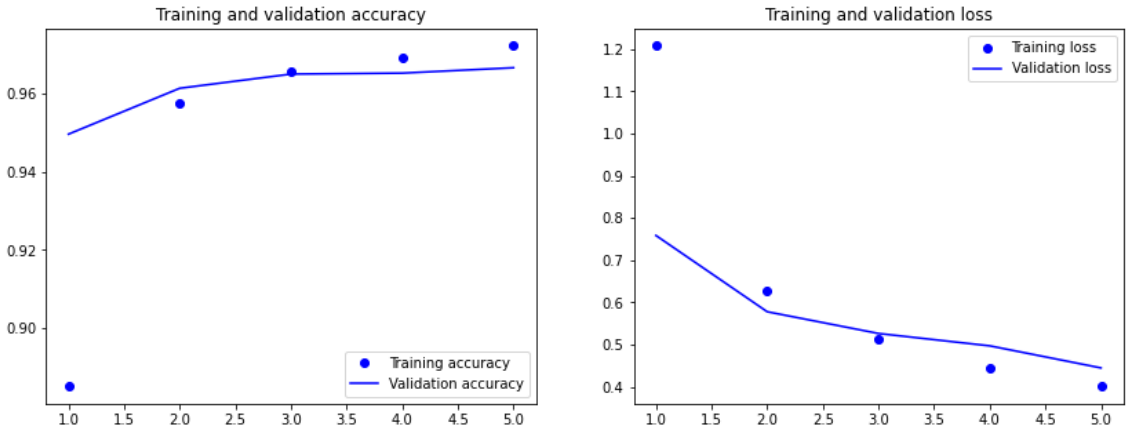


Gráfica 5-36 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 4 del sistema de reconocimiento de dibujos.

5.2.5. PRUEBA 5

Podemos observar que si cambiamos el número de filtros de las capas convolucionales y la penúltima capa densa a 32, 32, 128 y 128 respectivamente, obtenemos un porcentaje de acierto del 96.66%, lo que supone un descenso del 0.22% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 45.28%, lo que supone un aumento del 9.18% al 49.86%.

Como se puede observar en la gráfica 5-37, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

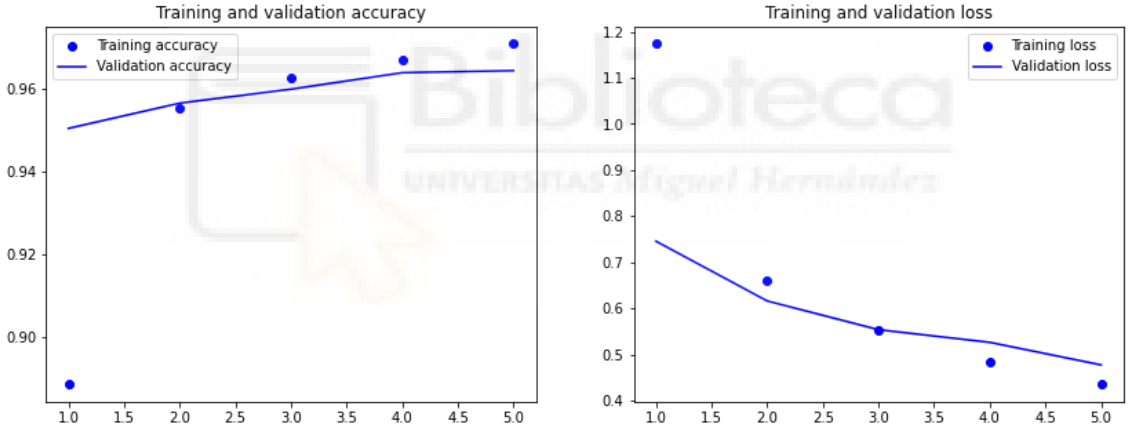


Gráfica 5-37 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 5 del sistema de reconocimiento de dibujos.

5.2.6. PRUEBA 6

Podemos observar que si variamos los filtros de todas las capas convolucionales y la penúltima capa densa a 16, 32, 64 y 64 respectivamente, logramos un porcentaje de acierto del 96.55%, lo que supone un aumento del 0.1% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 47.82%, lo que supone un descenso del 4.09% al 49.86%.

Como se puede observar en la gráfica 5-38, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.



Gráfica 5-38 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 6 del sistema de reconocimiento de dibujos.

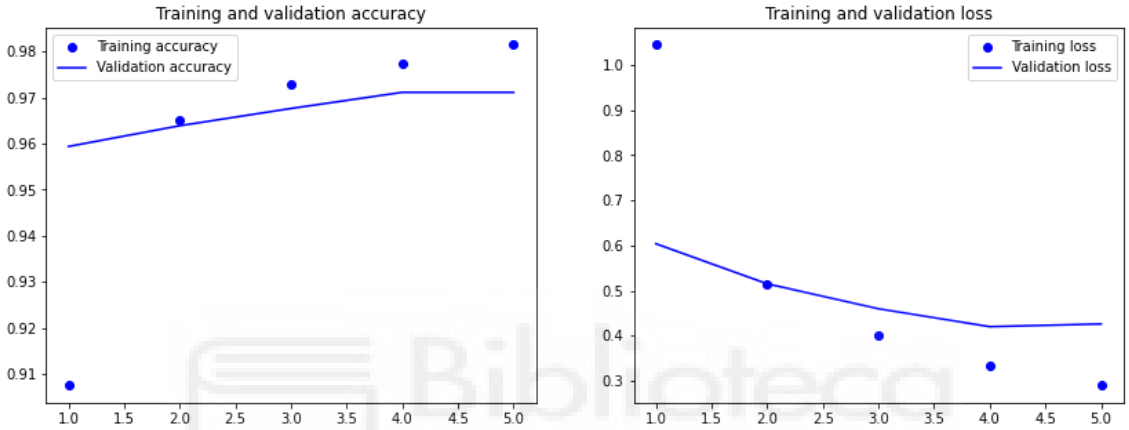
5.2.7. PRUEBA 7

Podemos observar que si modificamos la arquitectura añadiéndole tres capas convolucionales, con lo que tendríamos seis capas de convolución, agrupándolas de dos en dos seguidas de una capa de agrupación y variamos el número de filtros de las capas convolucionales y la penúltima capa densa a 16, 16, 32, 32, 128, 128 y 256 respectivamente, obtenemos un porcentaje de acierto del 97.15%, lo que supone un aumento del 0.73% al 96.45% respecto del número de filtros de la arquitectura sin

modificar. Asimismo, obtenemos un error del 41.14%, lo que supone un descenso del 17.49% al 49.86%.

Como se puede observar en la gráfica 5-39, el mayor porcentaje de acierto se alcanza en la época 4, de igual manera que en el modelo sin modificar. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 4, en cambio en el modelo sin modificar se produce alrededor de la época 5.

En este caso podemos comprobar que se produce sobreajuste en la época 4.



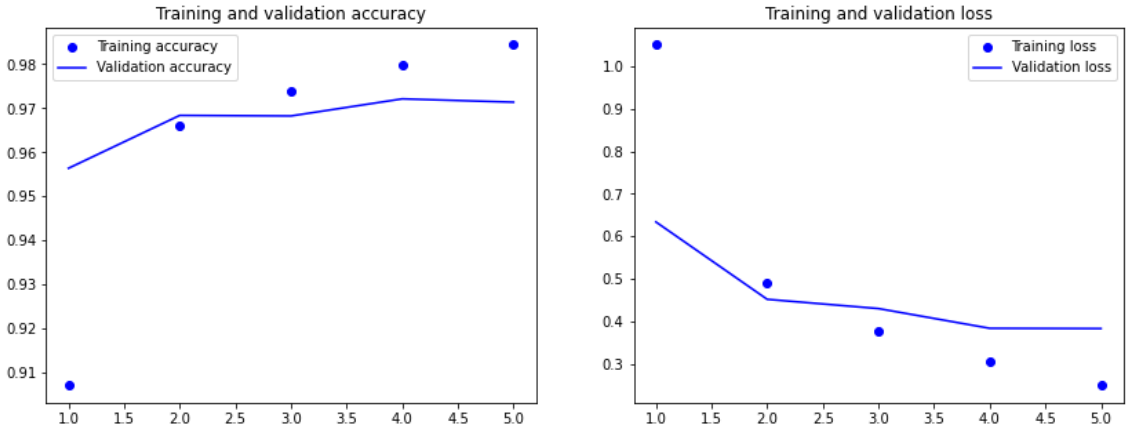
Gráfica 5-39 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 7 del sistema de reconocimiento de dibujos.

5.2.8. PRUEBA 8

Podemos observar que si duplicamos el número de filtros de las cuatro primeras capas convolucionales de la prueba anterior, logramos un porcentaje de acierto del 97.26%, lo que supone un aumento del 0.83% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 37.5%, lo que supone un descenso del 24.79% al 49.86%.

Como se puede observar en la gráfica 5-40, el mayor porcentaje de acierto se alcanza en la época 4, de igual manera que en el modelo sin modificar. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 4, en cambio en el modelo sin modificar se produce alrededor de la época 5.

En este caso podemos comprobar que se produce sobreajuste en la época 4.

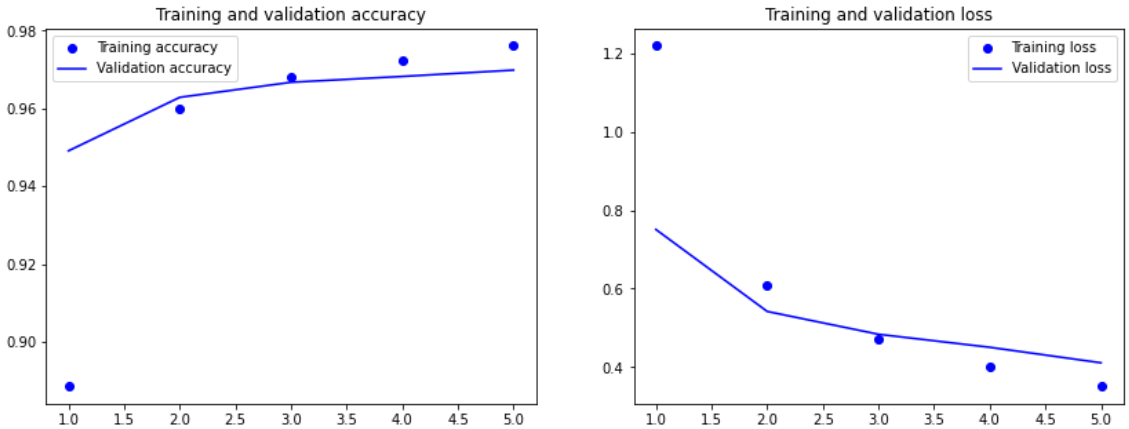


Gráfica 5-40 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 8 del sistema de reconocimiento de dibujos.

5.2.9. PRUEBA 9

Podemos observar que si modificamos la arquitectura de manera que tenemos cuatro capas convolucionales y cuatro capas de agrupación alternándose sucesivamente y definimos el número de filtros de las capas convolucionales y la penúltima capa densa a 32, 64, 128, 128 y 256 respectivamente, obtenemos un porcentaje de acierto del 97%, lo que supone un descenso del 0.57% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 41.34%, lo que supone un descenso del 17.09% al 49.86%.

Como se puede observar en la gráfica 5-41, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.



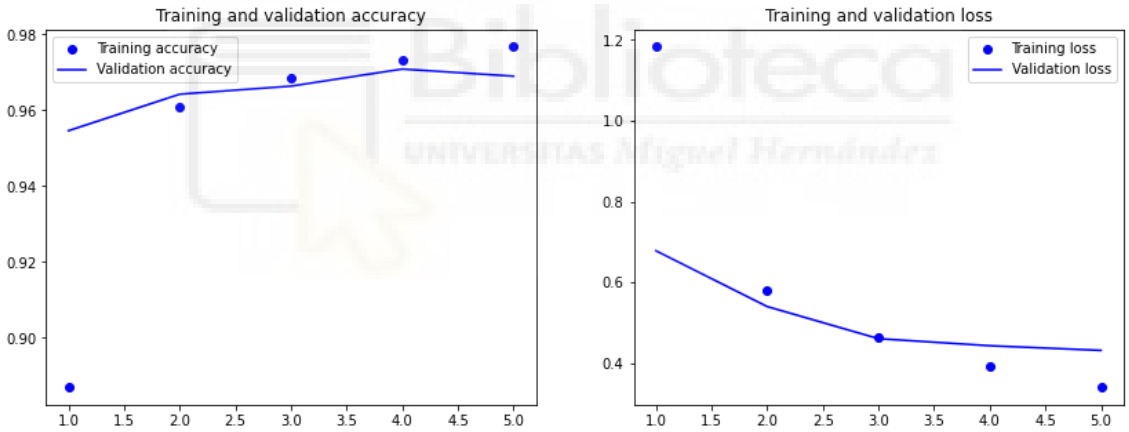
Gráfica 5-41 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 9 del sistema de reconocimiento de dibujos.

5.2.10. PRUEBA 10

Podemos observar que si variamos el número de filtros de la primera capa convolucional de la prueba anterior a 64, obtenemos una precisión del 96.9%, lo que supone un aumento del 0.47% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 42.21%, lo que supone un aumento del 15.34% al 49.86%.

Como se puede observar en la gráfica 5-42, el mayor porcentaje de acierto se alcanza en la época 4 y un menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

En este caso podemos comprobar que se produce un leve sobreajuste a partir de la época 4.

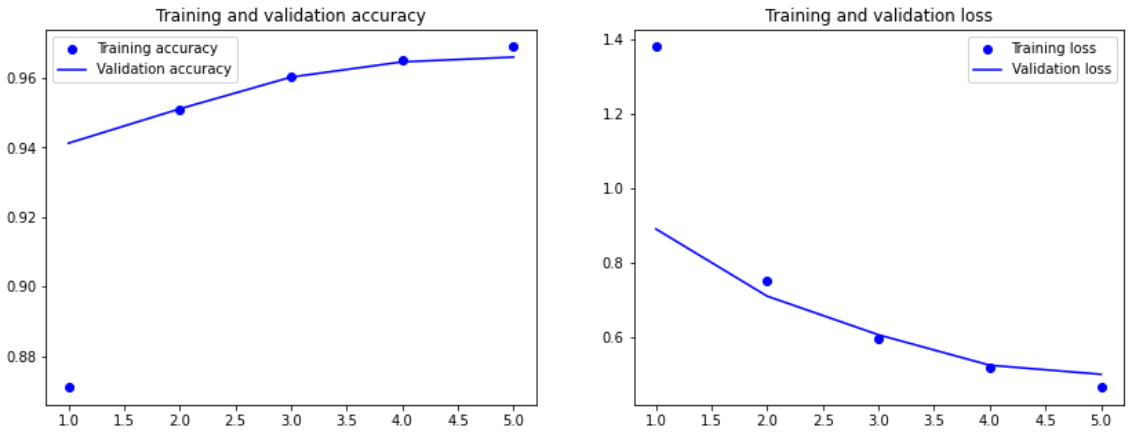


Gráfica 5-42 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 10 del sistema de reconocimiento de dibujos.

5.2.11. PRUEBA 11

Podemos observar que si cambiamos el número de filtros de las tres primeras capas convolucionales de la prueba 9 a 16, 32 y 64 respectivamente, logramos porcentaje de acierto del 96.42%, lo que supone un descenso del 0.03% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 50.01%, lo que supone un aumento del 0.3% al 49.86%.

Como se puede observar en la gráfica 5-43, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

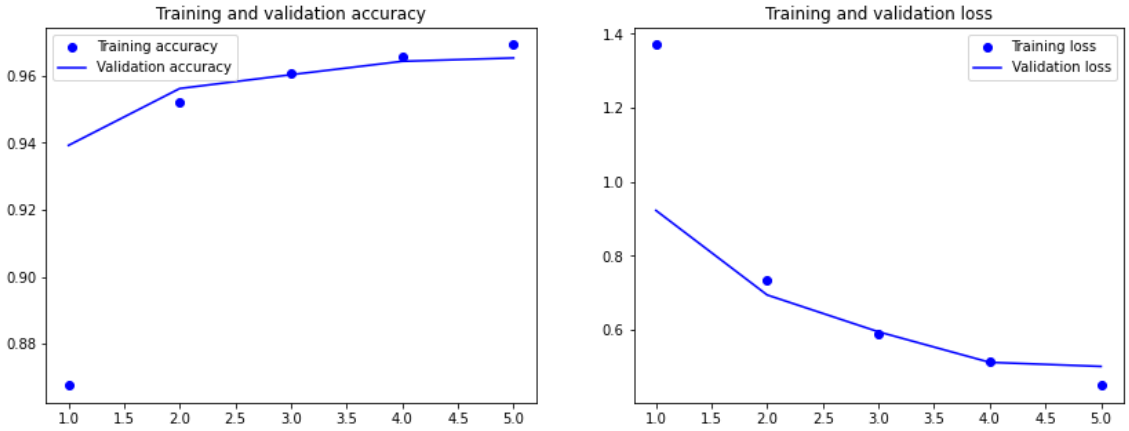


Gráfica 5-43 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 11 del sistema de reconocimiento de dibujos.

5.2.12. PRUEBA 12

Podemos observar que si modificamos el número de filtros de la penúltima capa densa de la prueba anterior a 128, obtenemos porcentaje de acierto del 96.64%, lo que supone un aumento del 0.2% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 50.25%, lo que supone un aumento del 0.78% al 49.86%.

Como se puede observar en la gráfica 5-44, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.



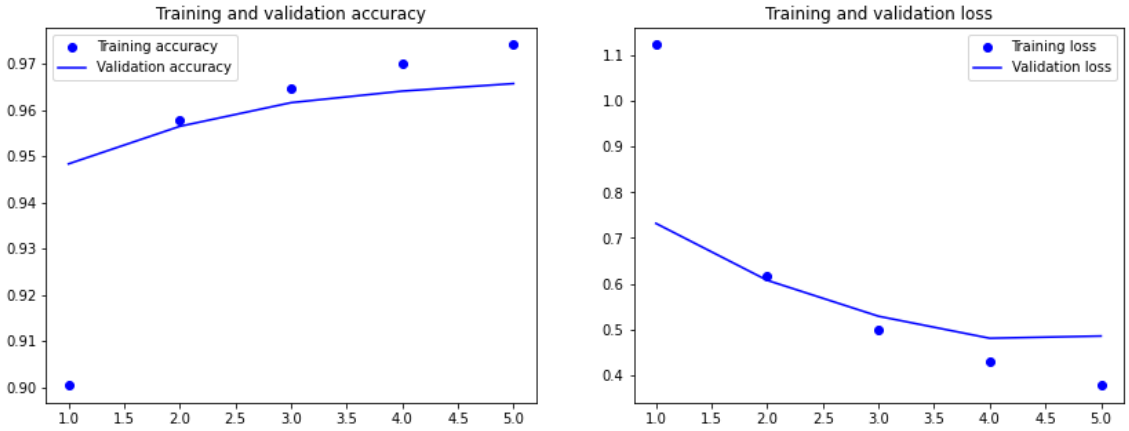
Gráfica 5-44 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 12 del sistema de reconocimiento de dibujos.

5.2.13. PRUEBA 13

Podemos observar que si modificamos el número de capas de la arquitectura a cuatro de convolución y dos de agrupación, de manera que agrupamos las capas convolucionales de dos en dos seguidas de una capa de agrupación y definimos el número de filtros como 16 el primer grupo de capas convolucionales, 32 el segundo y 128 la penúltima capa densa, logramos así porcentaje de acierto del 96.63%, lo que supone un aumento del 0.19% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 47.69%, lo que supone un descenso del 4.35% al 49.86%.

Como se puede observar en la gráfica 5-45, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 4, en cambio en el modelo sin modificar se produce alrededor de la época 5.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 3.



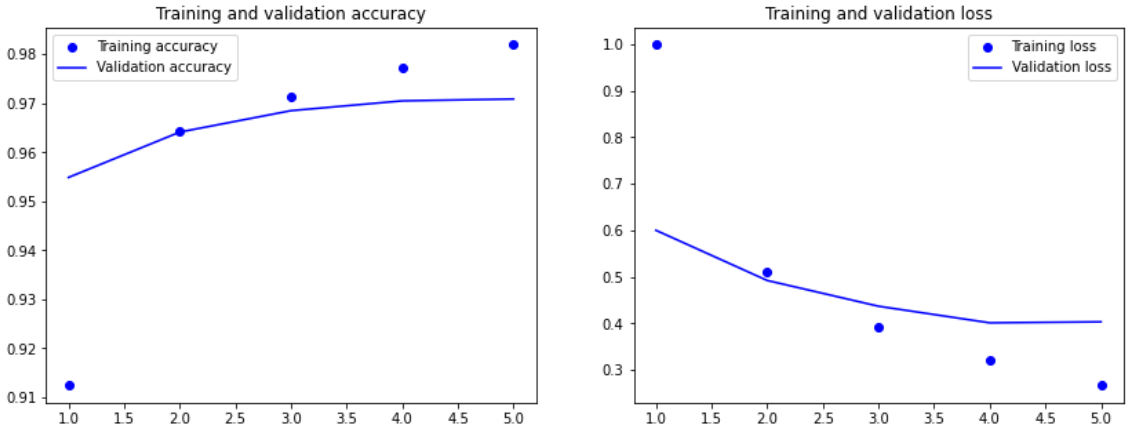
Gráfica 5-45 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 13 del sistema de reconocimiento de dibujos.

5.2.14. PRUEBA 14

Podemos observar que si cambiamos de la prueba anterior el número de filtros del primer grupo de capas convolucionales a 32 y del segundo grupo a 64, obtenemos una precisión del 97.07%, lo que supone un aumento del 0.64% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 40.46%, lo que supone un descenso del 18.85% al 49.86%.

Como se puede observar en la gráfica 5-46, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 4, en cambio en el modelo sin modificar se produce alrededor de la época 5.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 3.



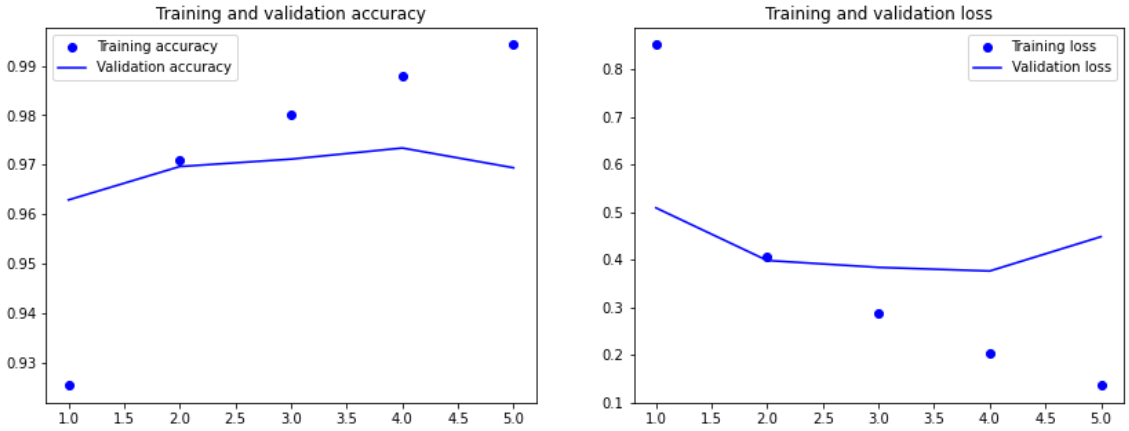
Gráfica 5-46 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 14 del sistema de reconocimiento de dibujos.

5.2.15. PRUEBA 15

Podemos observar que si variamos el número de filtros de la prueba 13 a 64 el primer grupo de capas convolucionales, 128 el segundo y 256 la penúltima capa densa; alcanzamos porcentaje de acierto del 96.9%, lo que supone un aumento del 0.47% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 46.01%, lo que supone un descenso del 7.72% al 49.86%.

Como se puede observar en la gráfica 5-47, el mayor porcentaje de acierto se alcanza en la época 4, de igual manera que en el modelo sin modificar. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 4, en cambio en el modelo sin modificar se produce alrededor de la época 5.

En este caso podemos comprobar que se produce sobreajuste alrededor de la época 2.

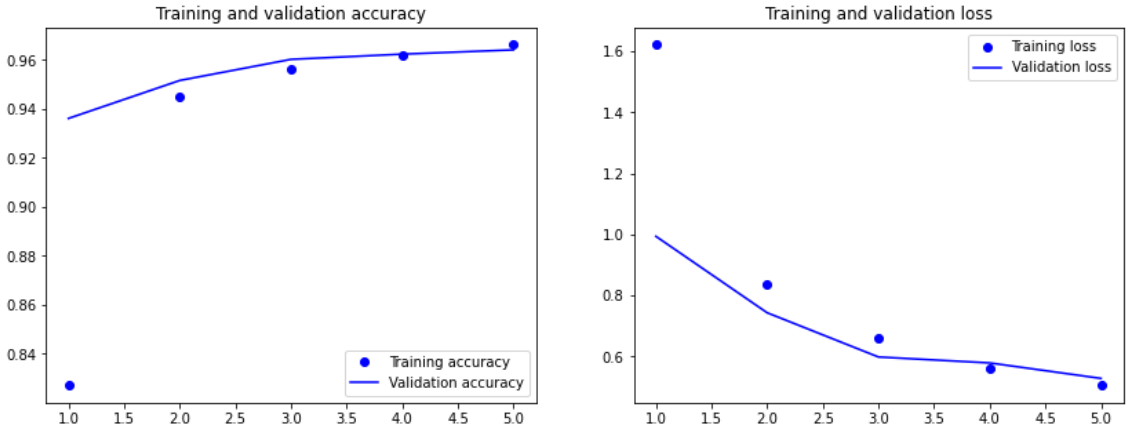


Gráfica 5-47 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 15 del sistema de reconocimiento de dibujos.

5.2.16. PRUEBA 16

Podemos observar que si modificamos el número de capas de la arquitectura a cinco capas convolucionales y cuatro de agrupación las cuales van alternándose sucesivamente, siendo el número de filtros de 16, 32, 64, 128 y 128 respectivamente, y cambiamos el número de filtros de la penúltima capa densa a 256, obtenemos así porcentaje de acierto del 96.5%, lo que supone un aumento del 0.05% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 53.09%, lo que supone un aumento del 6.47% al 49.86%.

Como se puede observar en la gráfica 5-48, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

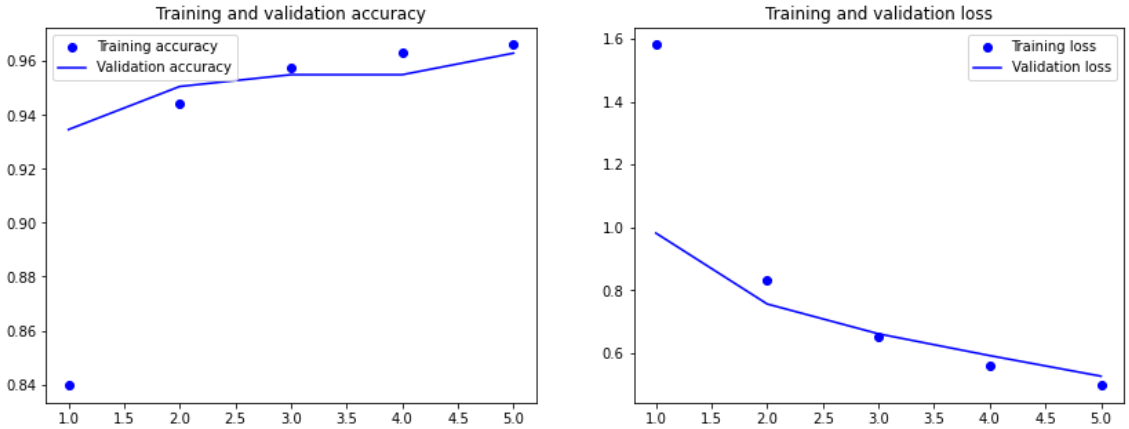


Gráfica 5-48 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 16 del sistema de reconocimiento de dibujos.

5.2.17. PRUEBA 17

Podemos observar que si cambiamos el número de filtros de la primera capa convolucional de la prueba anterior a 32, conseguimos porcentaje de acierto del 96.42%, lo que supone un descenso del 0.03% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 52.26%, lo que supone un aumento del 4.81% al 49.86%.

Como se puede observar en la gráfica 5-49, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

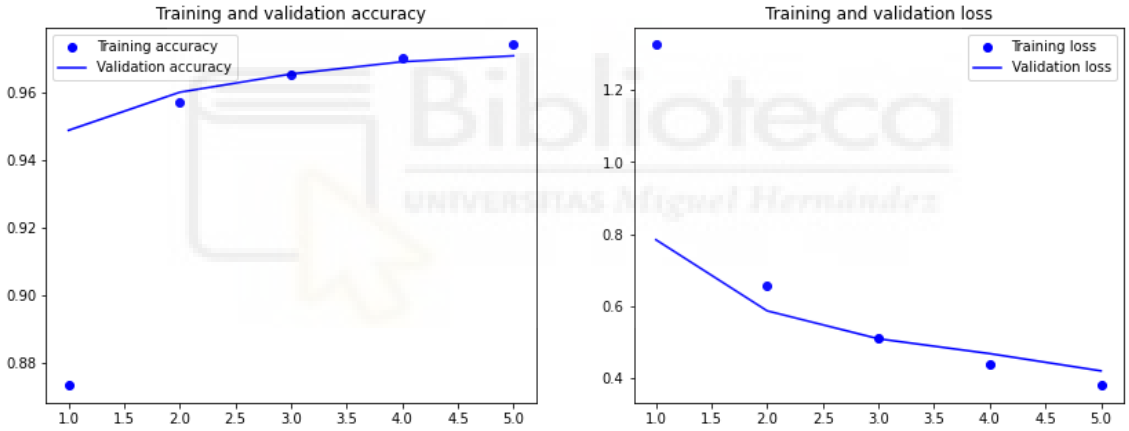


Gráfica 5-49 Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 17 del sistema de reconocimiento de dibujos.

5.2.18. PRUEBA 18

Podemos observar que si variamos el número de filtros de la prueba 16 de la primera y segunda capa convolucional a 64 y los de la quinta capa convolucional a 256, obtenemos una precisión del 97%, lo que supone un aumento del 0.57% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 43.12%, lo que supone un descenso del 13.52% al 49.86%.

Como se puede observar en la gráfica 5-50, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.



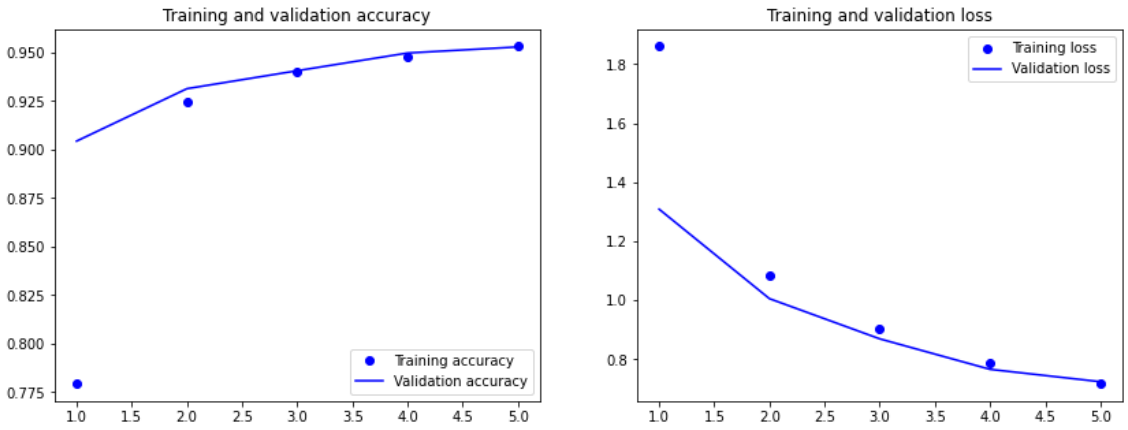
Gráfica 5-50. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 18 del sistema de reconocimiento de dibujos.

5.2.19. PRUEBA 19

Podemos observar que si cambiamos de la prueba 16 el número de filtros de las capas convolucionales y la penúltima capa densa a 16, 16, 32, 32, 64 y 64 respectivamente, obtenemos porcentaje de acierto del 95.13%, lo que supone un descenso del 1.37% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 72.6%, lo que supone un aumento del 45.6% al 49.86%.

Como se puede observar en la gráfica 5-51, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4.

Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

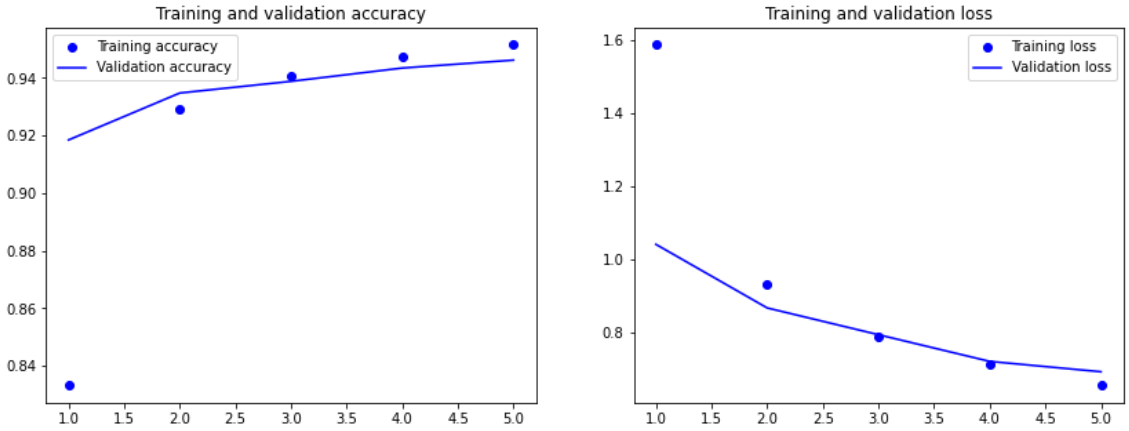


Gráfica 5-51. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 19 del sistema de reconocimiento de dibujos.

5.2.20. PRUEBA 20

Podemos observar que si trabajamos con 50 clases en la arquitectura original, obtenemos una precisión del 94.52%, lo que supone un descenso del 2% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 70.38%, lo que supone un aumento del 41.16% al 49.86%.

Como se puede observar en la gráfica 5-52, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.

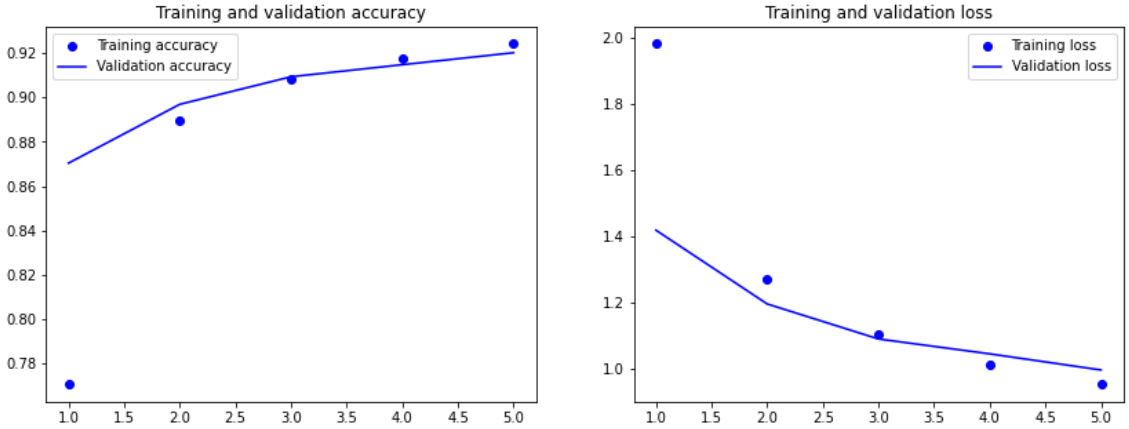


Gráfica 5-52. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 20 del sistema de reconocimiento de dibujos.

5.2.21. PRUEBA 21

Podemos observar que si trabajamos con 100 clases en la arquitectura original, obtenemos porcentaje de acierto del 92.24%, lo que supone un descenso del 4.36% al 96.45% respecto del número de filtros de la arquitectura sin modificar. Asimismo, obtenemos un error del 97.17%, lo que supone un aumento del 94.89% al 49.86%.

Como se puede observar en la gráfica 5-53, el mayor porcentaje de acierto se alcanza en la época 5, en cambio en el modelo sin modificar se produce alrededor de la época 4. Además, observamos que el menor porcentaje de error alrededor se alcanza en la época 5 de igual manera que en el modelo sin modificar.



Gráfica 5-53. Representación de la precisión y el error de los datos de entrenamiento frente a los datos de validación de la prueba 21 del sistema de reconocimiento de dibujos.

5.3. VISUALIZACIÓN DE ACTIVACIONES INTERMEDIAS

5.3.1. EJEMPLO ESTRELLA

Primero veremos como ejemplo el dibujo de una estrella como se muestra en la figura 5-1. Si observamos individualmente cada filtro podemos ver que patrón intenta identificar la red.

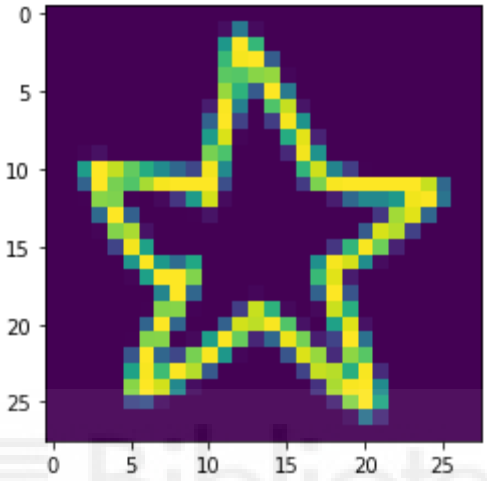


Figura 5-1. Dibujo de una estrella

En la figura 5-2 se muestra la representación interna correspondiente al primer canal, la cual parece detectar diagonales inclinadas hacia la derecha.

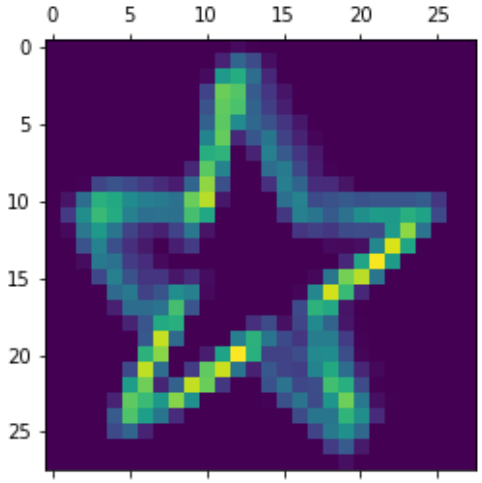


Figura 5-2. Primer canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-3 se muestra la representación interna correspondiente al segundo canal, la cual parece detectar diagonales inclinadas hacia la izquierda.

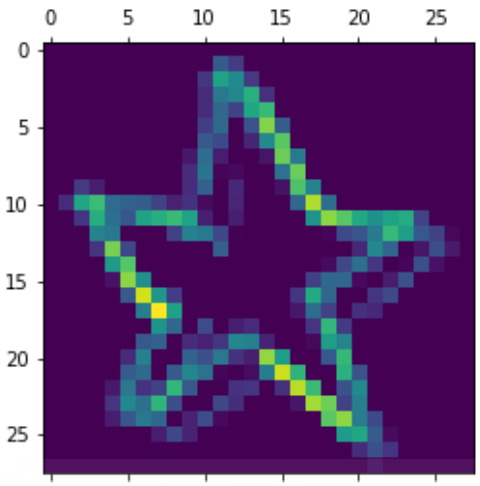


Figura 5-3. Segundo canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-4 se muestra la representación interna correspondiente al tercer canal, la cual parece detectar vértices.

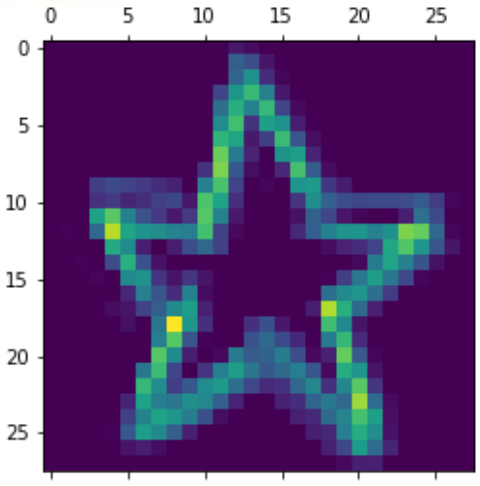


Figura 5-4. Tercer canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-5 se muestra la representación interna correspondiente al cuarto canal, la cual parece identificar líneas diagonales.

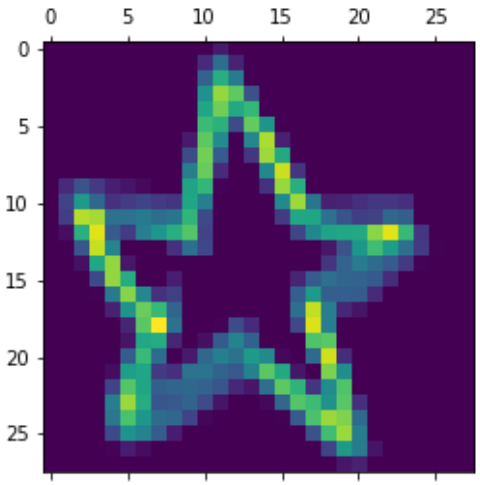


Figura 5-5. Cuarto canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-6 se muestra la representación interna correspondiente al quinto canal, la cual parece que parece detectar líneas diagonales.

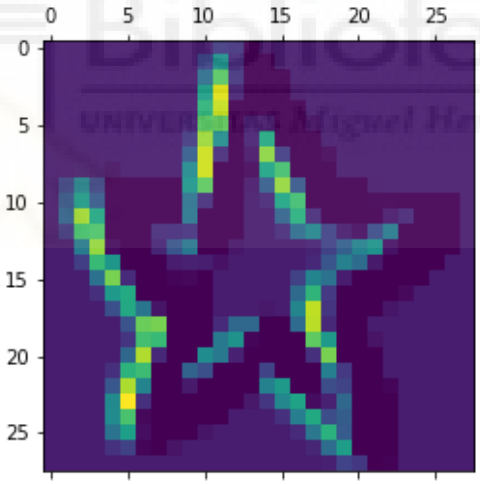


Figura 5-6. Quinto canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-7 se muestra la representación interna correspondiente al sexto canal, la cual parece detectar líneas horizontales y diagonales.

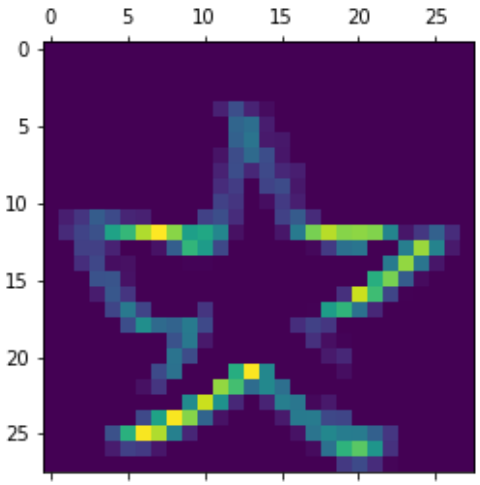


Figura 5-7. Sexto canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-8 se muestra la representación interna correspondiente al séptimo canal, la cual parece detectar vértices.

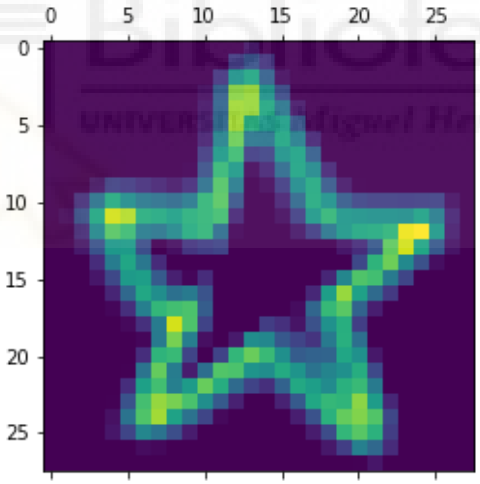


Figura 5-8. Séptimo canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-9 se muestra la representación interna correspondiente al octavo canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

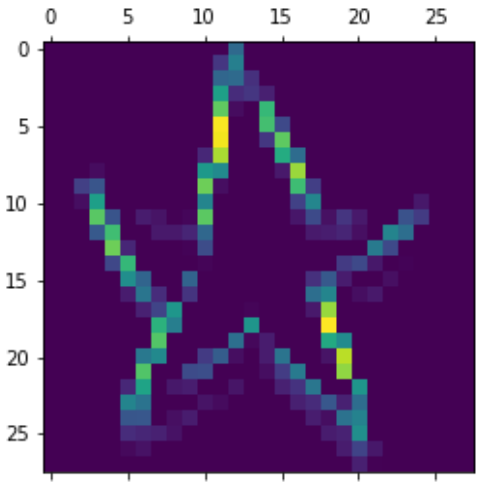


Figura 5-9. Octavo canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-10 se muestra la representación interna correspondiente al noveno canal, la cual parece detectar líneas horizontales y diagonales.

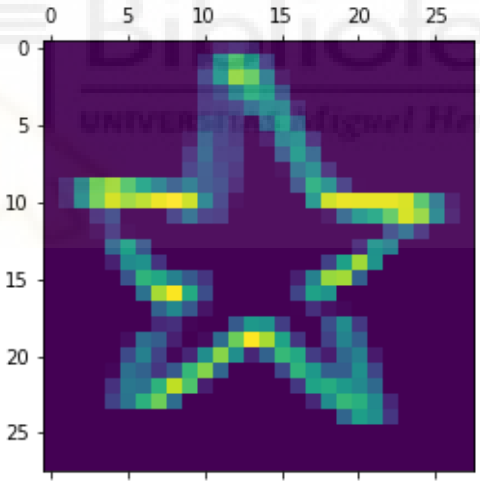


Figura 5-10. Noveno canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-11 se muestra la representación interna correspondiente al décimo canal, la cual parece detectar líneas diagonales.

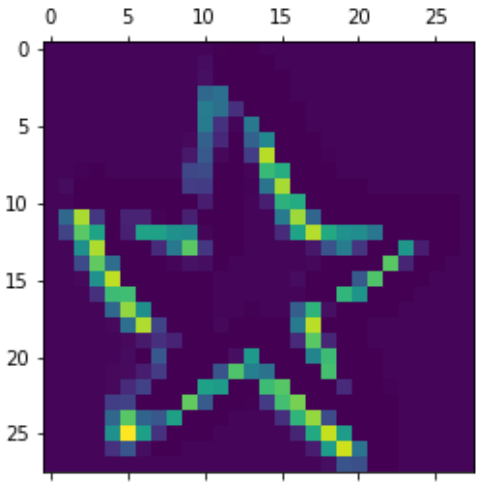


Figura 5-11. Décimo canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-12 se muestra la representación interna correspondiente al undécimo canal, la cual parece identificar líneas horizontales.

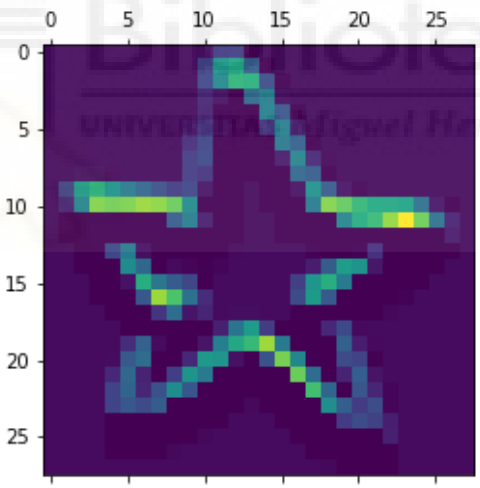


Figura 5-12. Undécimo canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-13 se muestra la representación interna correspondiente al duodécimo canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

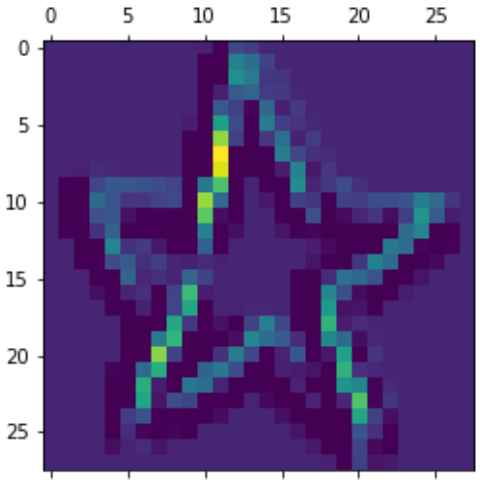


Figura 5-13. Duodécimo canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-14 se muestra la representación interna correspondiente al decimotercer canal, la cual parece identificar líneas diagonales.

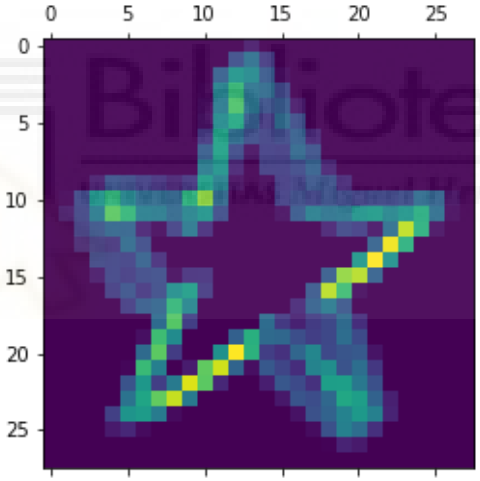


Figura 5-14. Decimotercer canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-15 se muestra la representación interna correspondiente al decimocuarto canal, la cual parece detectar líneas diagonales.

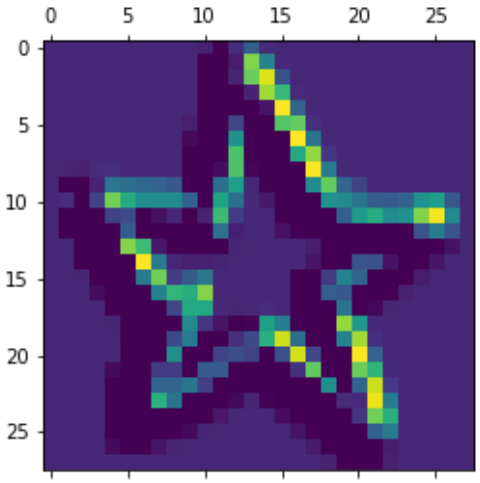


Figura 5-15. Decimocuarto canal de la activación de la primera capa del dibujo de una estrella.

En la figura 5-16 y 5-17 se muestra la representación interna correspondiente al decimoquinto y decimosexto canal, para las que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

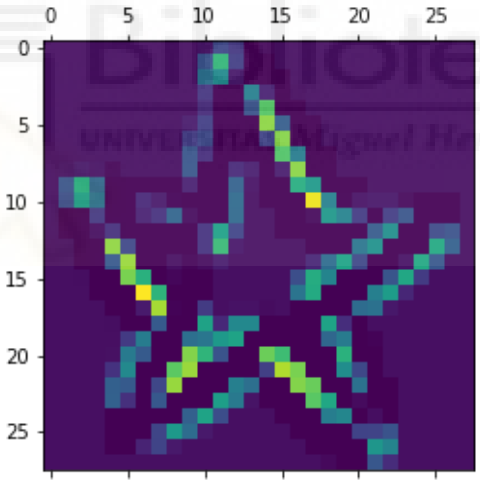


Figura 5-16. Decimoquinto canal de la activación de la primera capa del dibujo de una estrella.

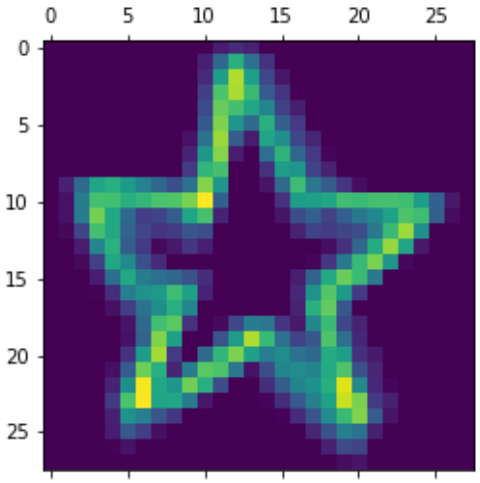


Figura 5-17. Decimosexto canal de la activación de la primera capa del dibujo de una estrella.

5.3.2. EJEMPLO CARA

Como segundo ejemplo utilizaremos el dibujo de una cara sonriente como se muestra en la figura 5-18.

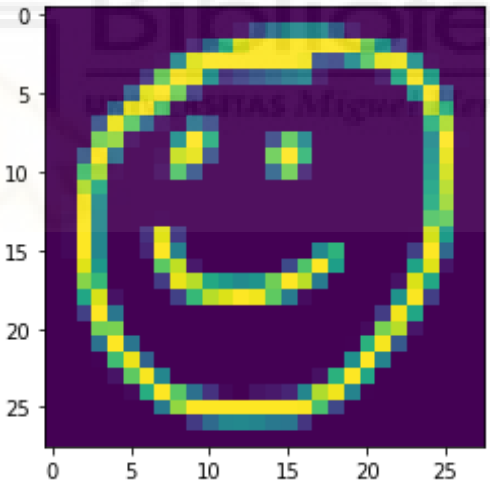


Figura 5-18. Dibujo de una cara sonriente

En la figura 5-19 se muestra la representación interna correspondiente al primer canal, la cual parece detectar bordes curvos.

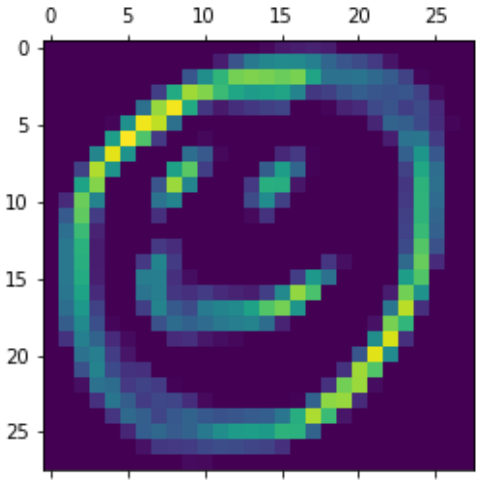


Figura 5-19. Primer canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-20 se muestra la representación interna correspondiente al segundo canal, la cual parece detectar bordes curvos.

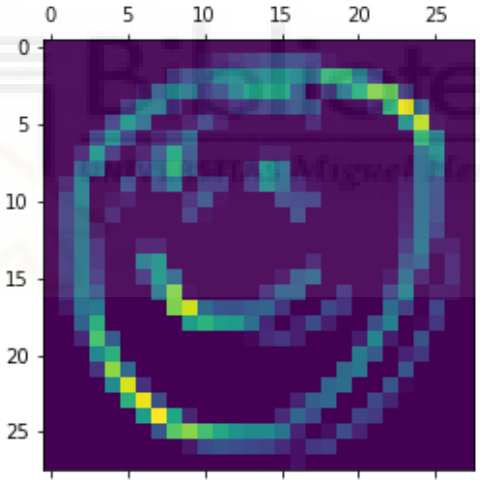


Figura 5-20. Segundo canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-21 se muestra la representación interna correspondiente al tercer canal, para las que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

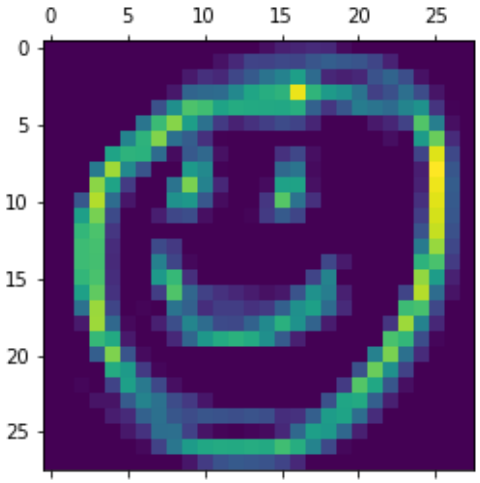


Figura 5-21. Tercer canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-22 se muestra la representación interna correspondiente al cuarto canal, la cual parece detectar bordes curvos.

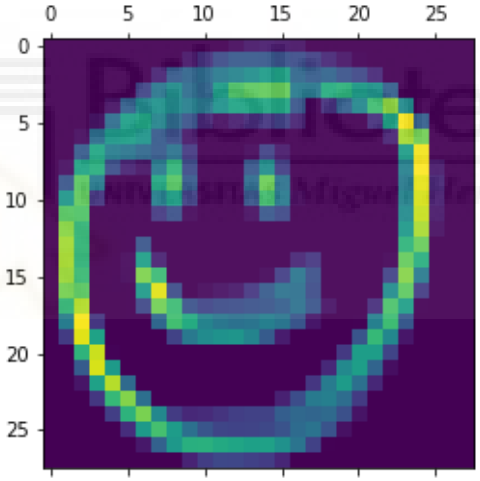


Figura 5-22. Cuarto canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-23 se muestra la representación interna correspondiente al quinto canal, la cual parece detectar bordes curvos.

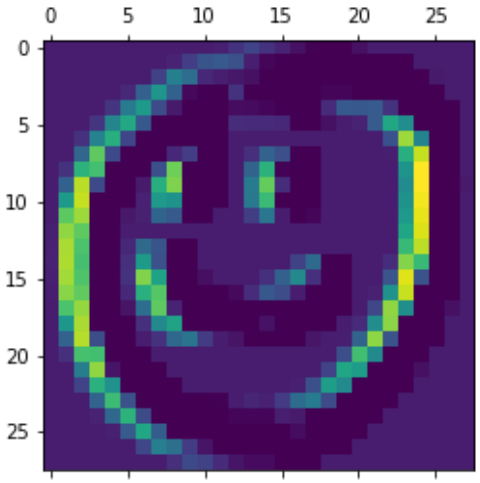


Figura 5-23. Quinto canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-24 se muestra la representación interna correspondiente al sexto canal, la cual parece detectar bordes curvos.

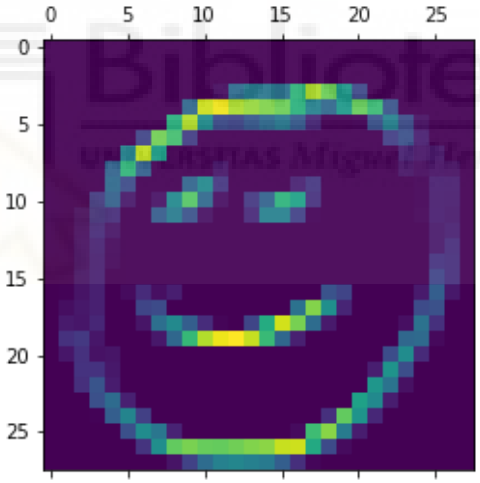


Figura 5-24. Sexto canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-25 se muestra la representación interna correspondiente al séptimo canal, la cual parece detectar bordes curvos.

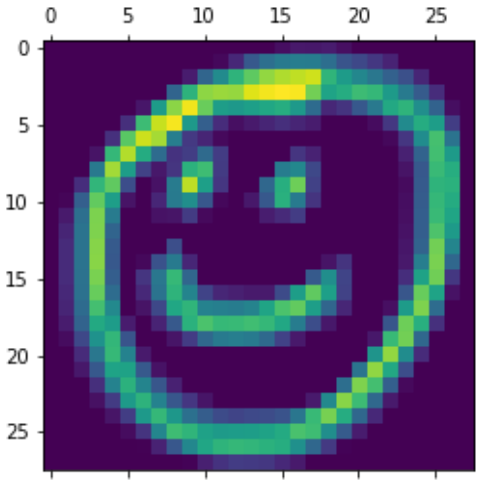


Figura 5-25. Séptimo canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-26 se muestra la representación interna correspondiente al octavo canal, para las que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

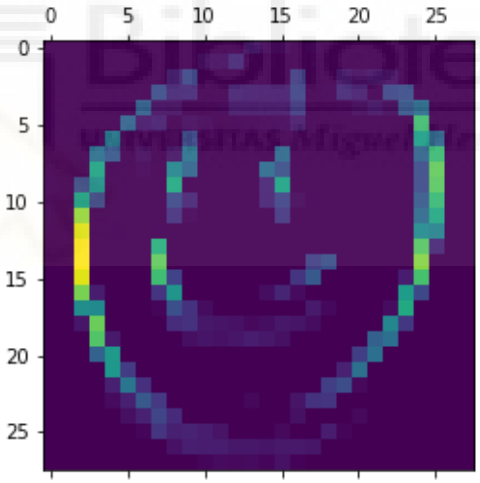


Figura 5-26. Octavo canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-27 se muestra la representación interna correspondiente al noveno canal, la cual parece detectar bordes curvos.

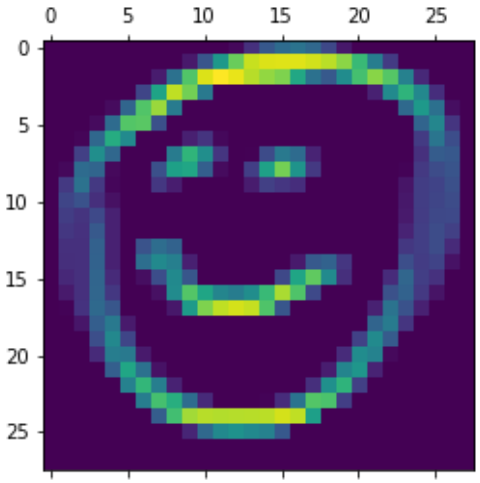


Figura 5-27. Noveno canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-28 se muestra la representación interna correspondiente al décimo canal, para las que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

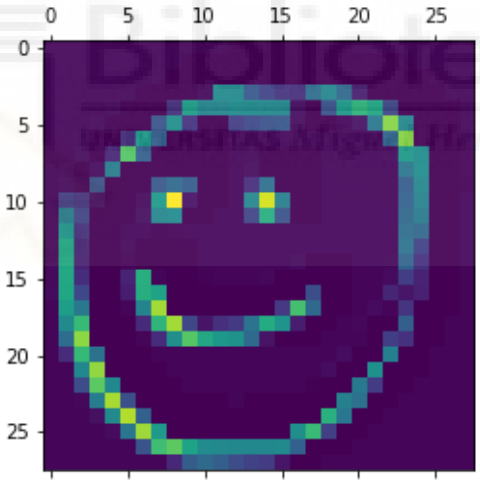


Figura 5-28. Décimo canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-29 se muestra la representación interna correspondiente al undécimo canal, la cual parece detectar bordes curvos.

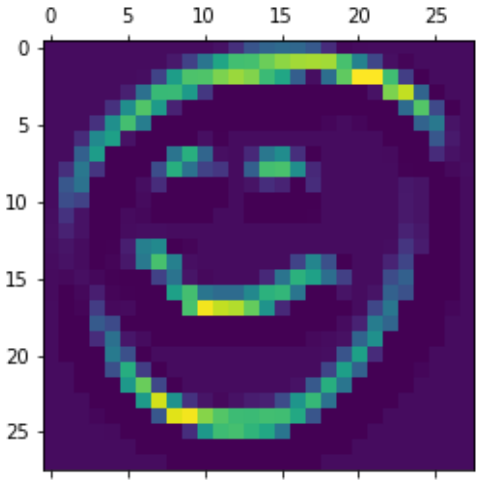


Figura 5-29. Undécimo canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-30 se muestra la representación interna correspondiente al duodécimo canal, para las que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

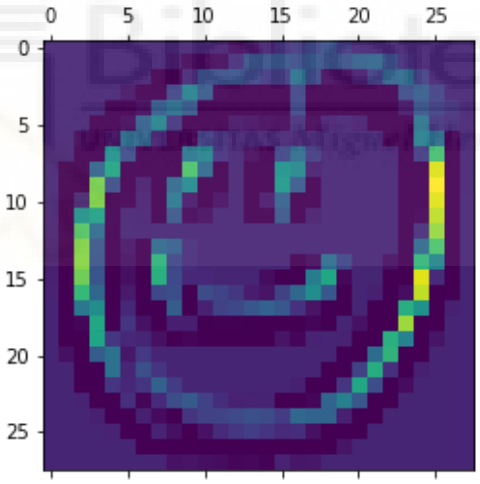


Figura 5-30. Duodécimo canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-31 se muestra la representación interna correspondiente al decimotercer canal, la cual parece detectar bordes curvos.

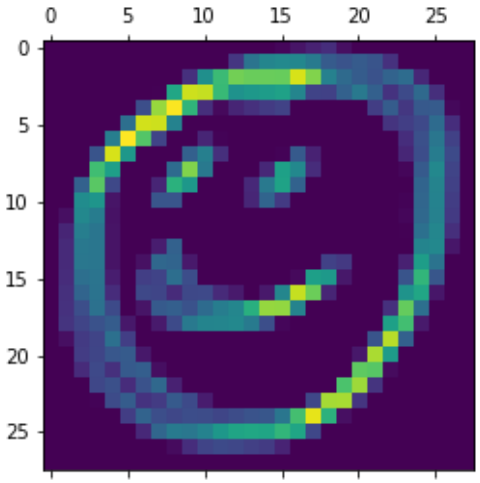


Figura 5-31. Decimotercer canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-32 y 5-33 se muestra la representación interna correspondiente al decimocuarto y decimoquinto canal, para las que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

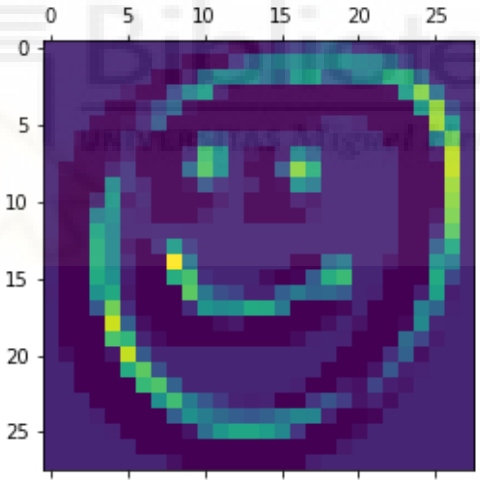


Figura 5-32. Decimocuarto canal de la activación de la primera capa del dibujo de una cara sonriente.

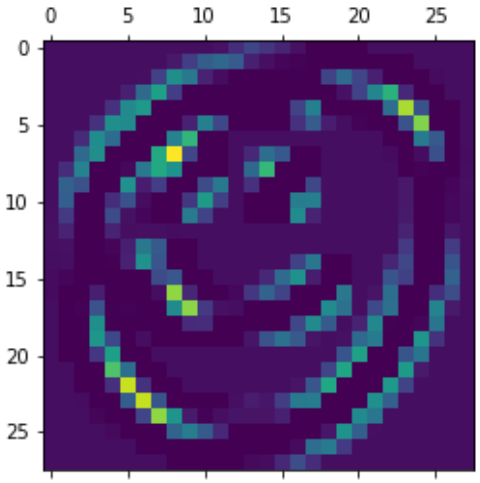


Figura 5-33. Decimoquinto canal de la activación de la primera capa del dibujo de una cara sonriente.

En la figura 5-34 se muestra la representación interna correspondiente al decimosexto canal, la cual parece detectar bordes curvos.

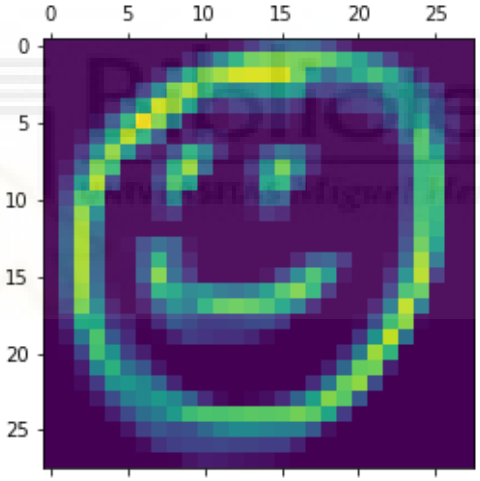


Figura 5-34. Decimosexto canal de la activación de la primera capa del dibujo de una cara sonriente.

5.3.3. EJEMPLO MESA

Como tercer ejemplo utilizaremos el dibujo de una mesa como se muestra en la figura 5-35.

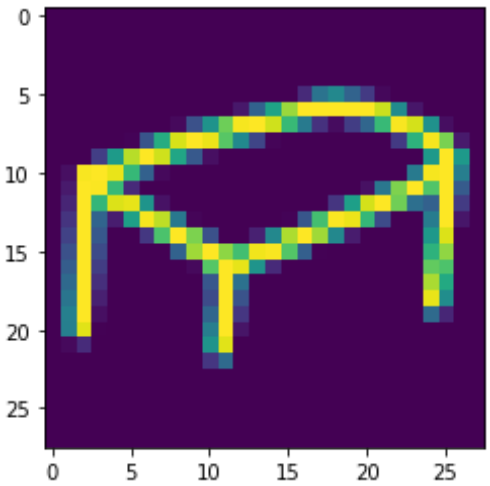


Figura 5-35. Dibujo de una mesa

En la figura 5-36 se muestra la representación interna correspondiente al primer canal, la cual parece detectar líneas diagonales inclinadas hacia la derecha.

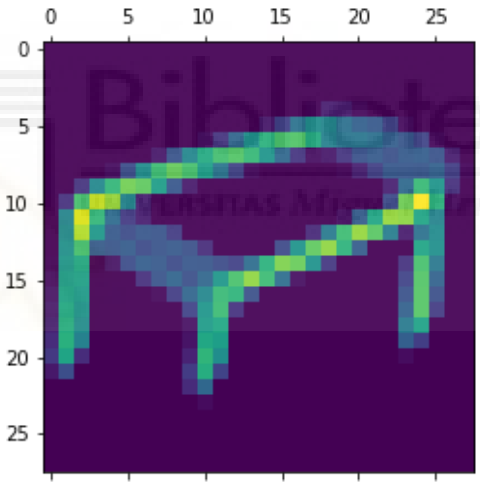


Figura 5-36. Primer canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-37 se muestra la representación interna correspondiente al segundo canal, la cual parece detectar líneas diagonales inclinadas hacia la izquierda.

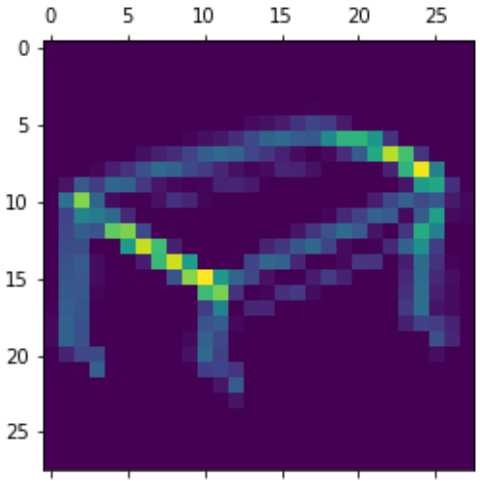


Figura 5-37. Segundo canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-38 se muestra la representación interna correspondiente al tercer canal, la cual detectar líneas verticales.

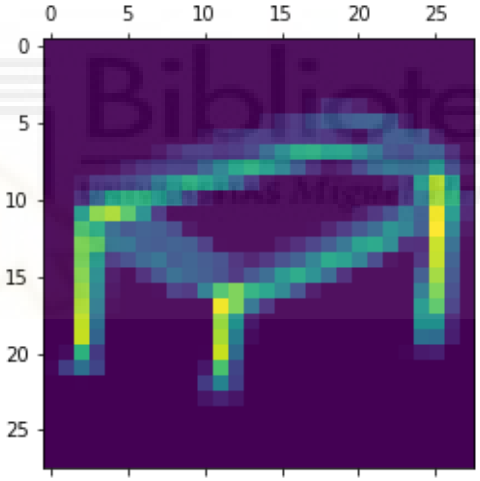


Figura 5-38. Tercer canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-39 se muestra la representación interna correspondiente al cuarto canal, la cual parece detectar líneas verticales.

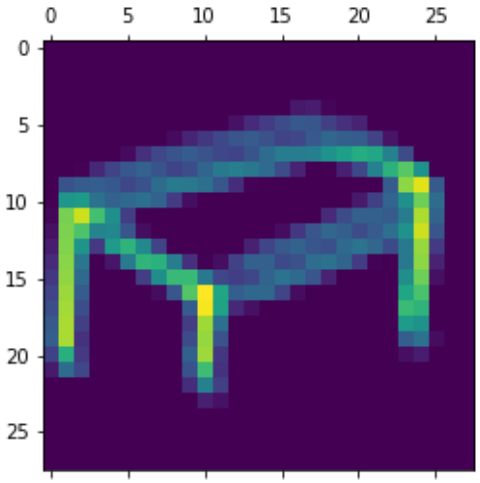


Figura 5-39. Cuarto canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-40 se muestra la representación interna correspondiente al quinto canal, la cual parece detectar líneas verticales.

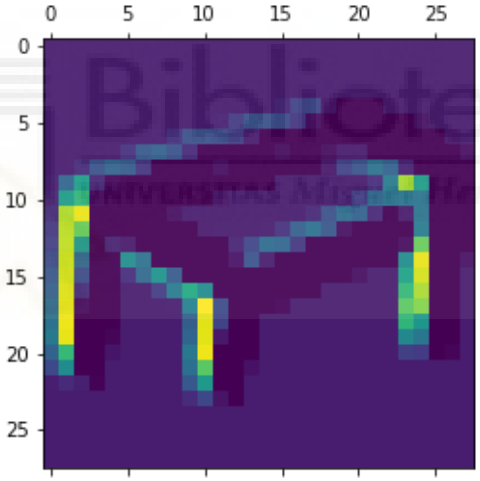


Figura 5-40. Quinto canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-41 se muestra la representación interna correspondiente al sexto canal, la cual parece identificar las líneas diagonales inclinadas hacia la derecha.

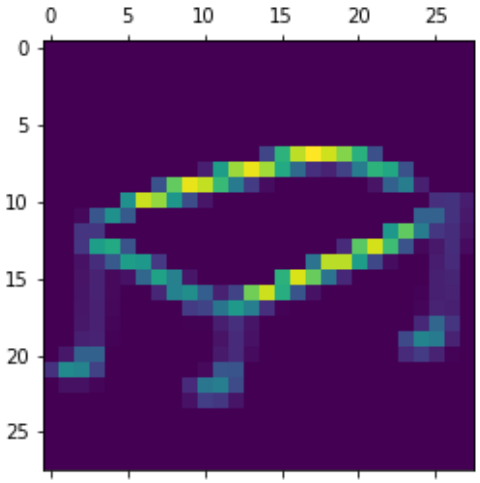


Figura 5-41. Sexto canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-42 se muestra la representación interna correspondiente al séptimo canal, la cual parece identificar vértices.

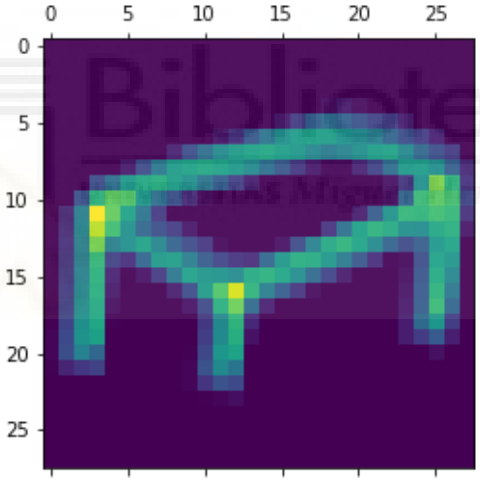


Figura 5-42. Séptimo canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-43 se muestra la representación interna correspondiente al octavo canal, la cual parece detectar líneas verticales

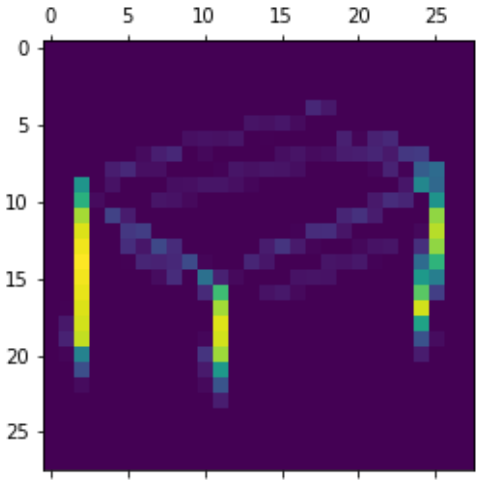


Figura 5-43. Octavo canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-44 se muestra la representación interna correspondiente al noveno canal, la cual parece identificar líneas diagonales.

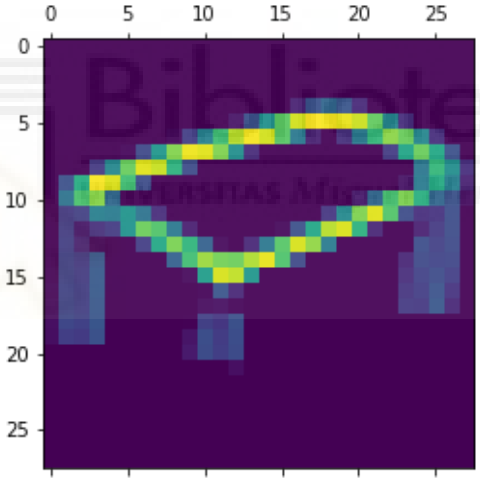


Figura 5-44. Noveno canal de la activación de la primera capa del dibujo de una mesa.

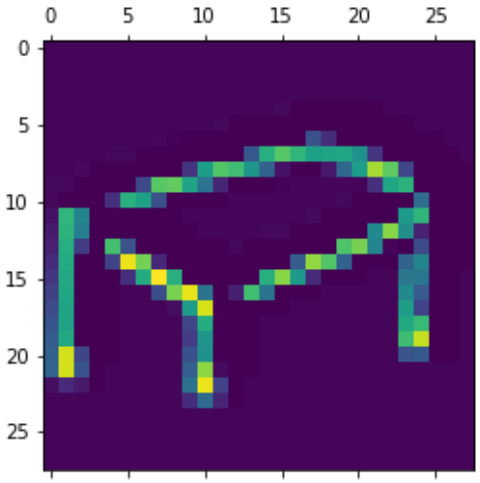


Figura 5-45. Décimo canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-46 se muestra la representación interna correspondiente al undécimo canal, la cual parece identificar líneas diagonales.

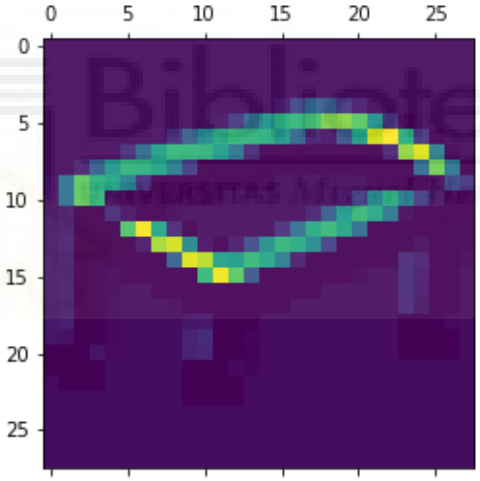


Figura 5-46. Undécimo canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-47 se muestra la representación interna correspondiente al duodécimo canal, la cual parece detectar líneas verticales.

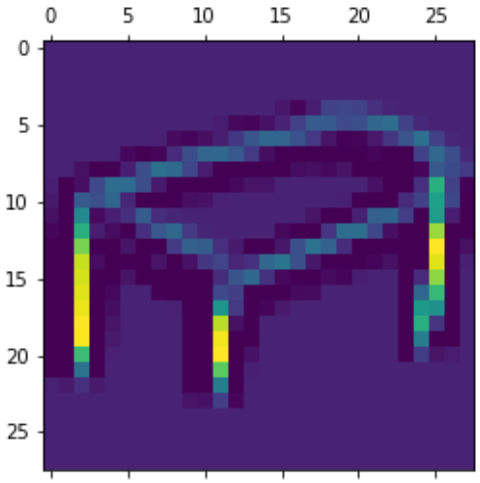


Figura 5-47. Duodécimo canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-48 se muestra la representación interna correspondiente al decimotercer canal, la cual parece detectar líneas diagonales.

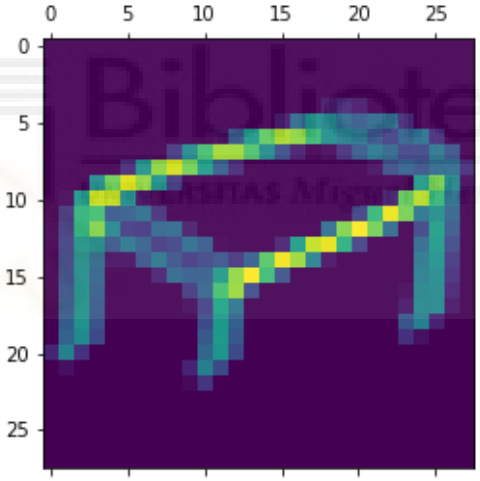


Figura 5-48. Decimotercer canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-49 se muestra la representación interna correspondiente al decimocuarto canal, la cual parece detectar líneas verticales.

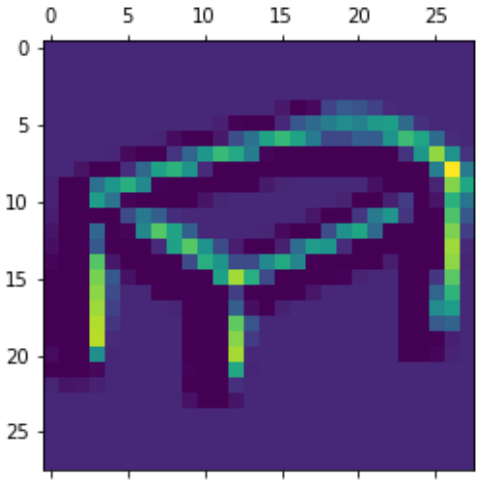


Figura 5-49. Decimocuarto canal de la activación de la primera capa del dibujo de una mesa.

En la figura 5-50 y 5-51 se muestra la representación interna correspondiente al decimoquinto y decimosexto canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

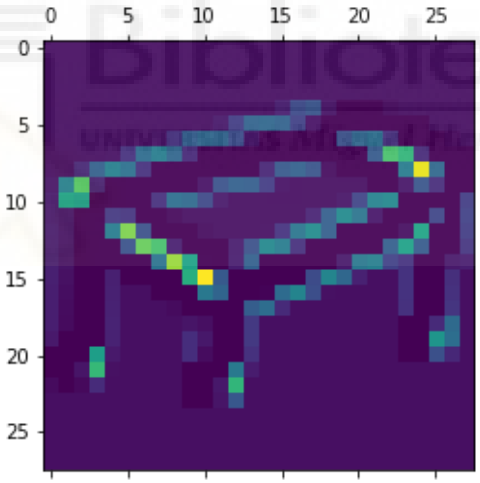


Figura 5-50. Decimoquinto canal de la activación de la primera capa del dibujo de una mesa.

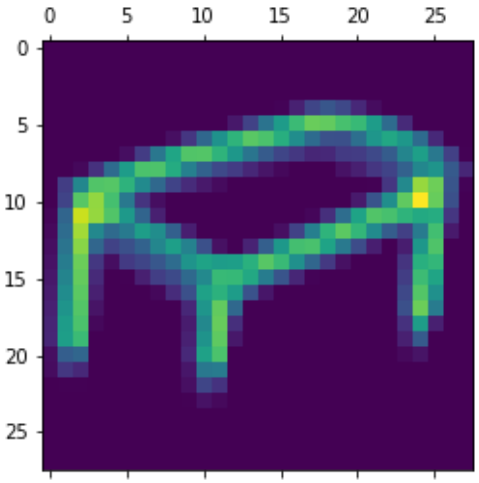


Figura 5-51. Decimosexto canal de la activación de la primera capa del dibujo de una mesa.

5.3.4. EJEMPLO SILLA

Como cuarto ejemplo usaremos el dibujo de una silla como se muestra en la figura 5-53.

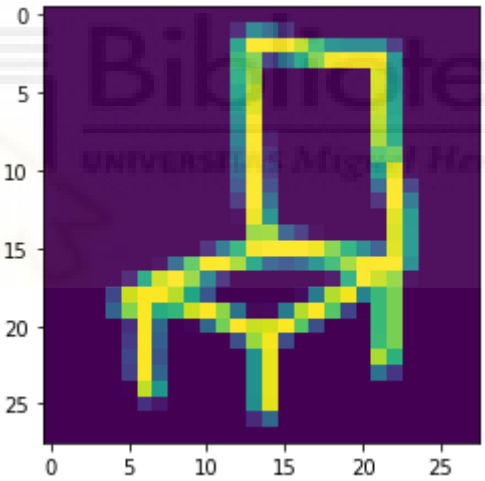


Figura 5-52. Dibujo de una silla.

En la figura 5-53 se muestra la representación interna correspondiente al primer canal, la cual parece detectar líneas verticales.

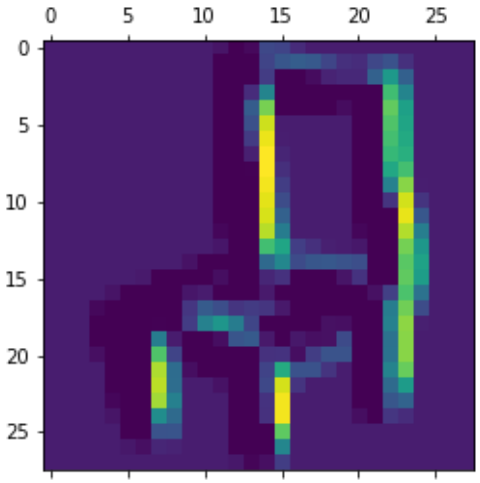


Figura 5-53. Primer canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-54 se muestra la representación interna correspondiente al segundo canal, la cual parece detectar líneas horizontales.

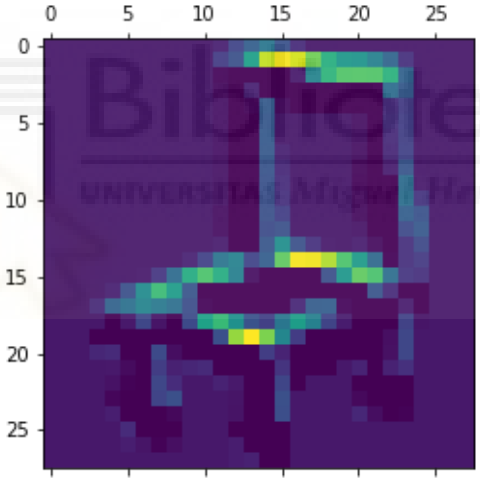


Figura 5-54. Segundo canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-55 se muestra la representación interna correspondiente al tercer canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

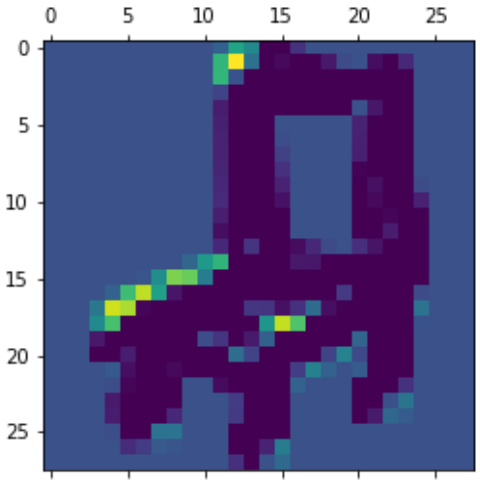


Figura 5-55. Tercer canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-56 se muestra la representación interna correspondiente al cuarto canal, la cual parece detectar esquinas.

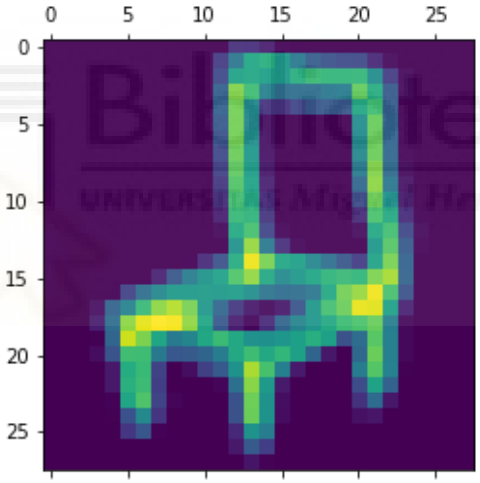


Figura 5-56. Cuarto canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-57 se muestra la representación interna correspondiente al quinto canal, la cual parece detectar líneas diagonales.

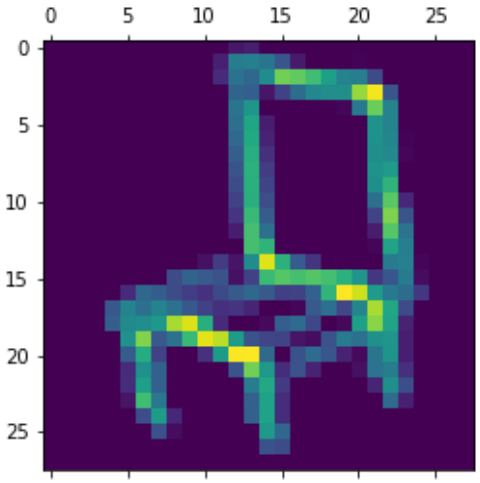


Figura 5-57. Quinto canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-58 se muestra la representación interna correspondiente al sexto canal, la cual parece detectar líneas horizontales inclinadas hacia la izquierda.

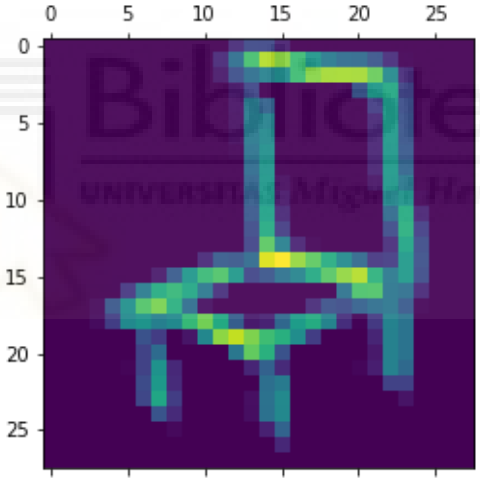


Figura 5-58. Sexto canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-59 se muestra la representación interna correspondiente al séptimo canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

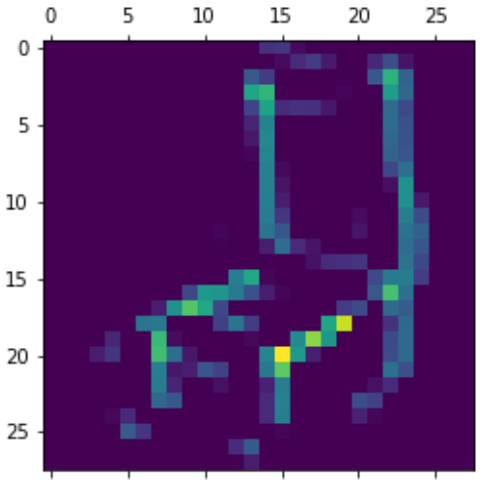


Figura 5-59. Séptimo canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-60 se muestra la representación interna correspondiente al octavo canal, la cual parece detectar líneas horizontales y diagonales.

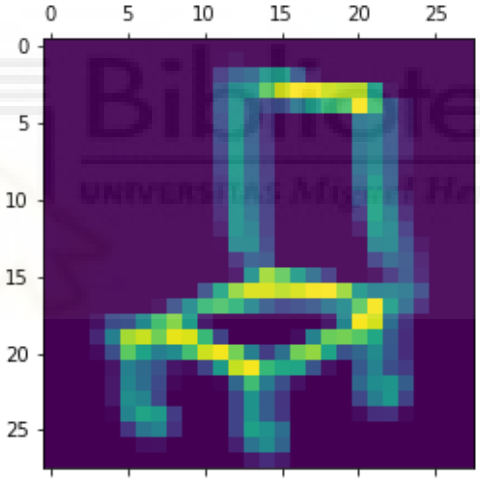


Figura 5-60. Octavo canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-61 se muestra la representación interna correspondiente al noveno canal, la cual 5-61 parece detectar vértices.

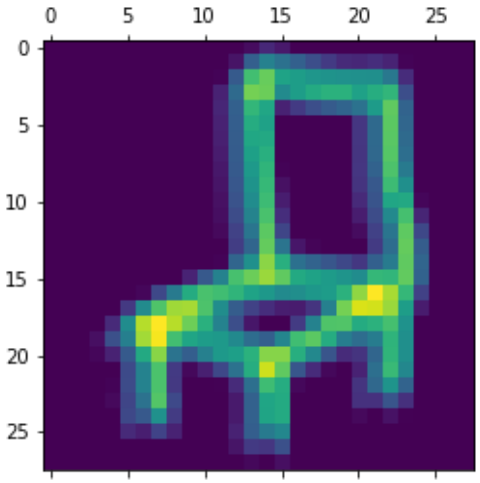


Figura 5-61. Noveno canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-62 se muestra la representación interna correspondiente al décimo canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

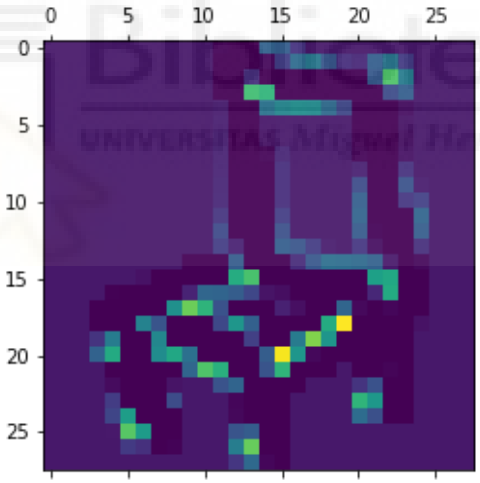


Figura 5-62. Décimo canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-63 se muestra la representación interna correspondiente al undécimo canal, la cual parece identificar líneas horizontales.

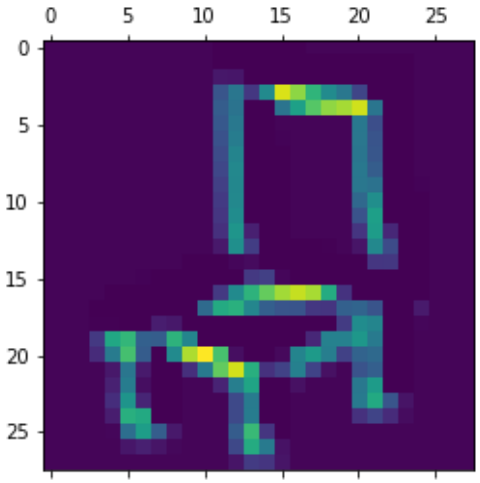


Figura 5-63. Undécimo canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-64 se muestra la representación interna correspondiente al duodécimo canal, la cual parece identificar líneas verticales.

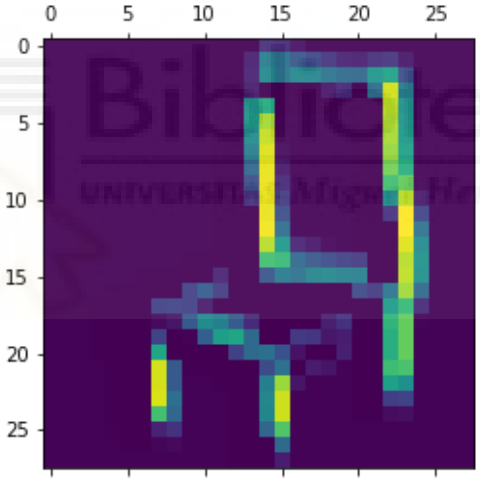


Figura 5-64. Duodécimo canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-65 se muestra la representación interna correspondiente al decimotercer canal, la cual parece detectar líneas verticales.

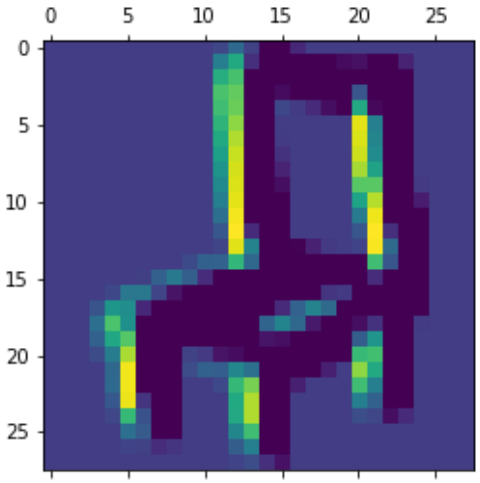


Figura 5-65. Decimotercer canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-66 se muestra la representación interna correspondiente al decimocuarto canal, la cual parece detectar líneas verticales.

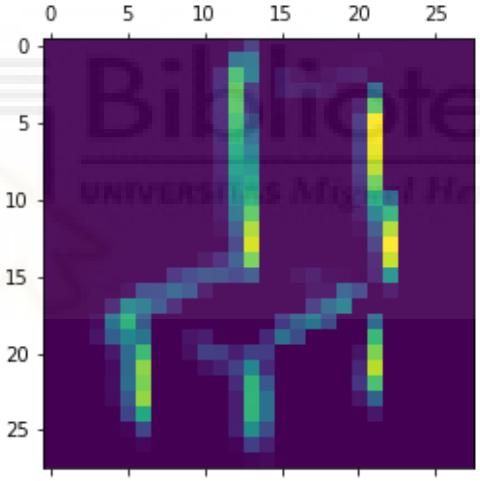


Figura 5-66. Decimocuarto canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-67 se muestra la representación interna correspondiente al decimoquinto canal, para la que no se ha podido obtener una interpretación clara e intuitiva del aprendizaje de la red.

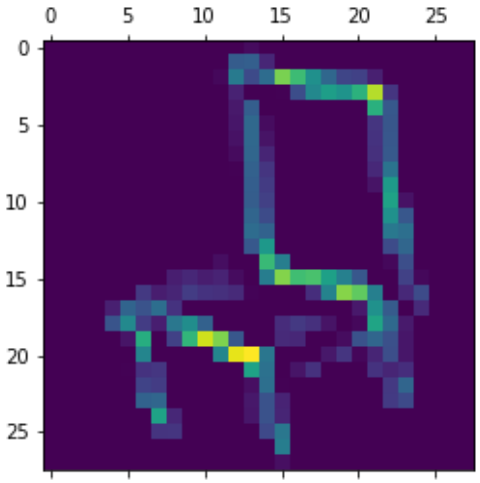


Figura 5-67. Decimoquinto canal de la activación de la primera capa del dibujo de una silla.

En la figura 5-68 se muestra la representación interna correspondiente al decimosexto canal, la cual parece detectar vértices.

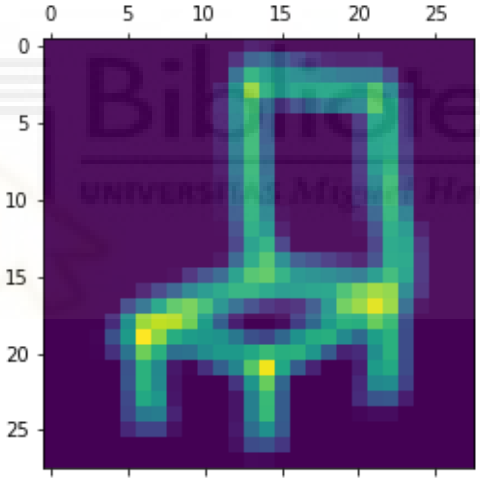


Figura 5-68. Decimosexto canal de la activación de la primera capa del dibujo de una silla.

CAPÍTULO 6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1. CONCLUSIONES

A continuación se resumen las principales conclusiones que se han obtenido durante la realización del trabajo.

- Se ha estudiado y analizado cómo pueden realizarse aplicaciones de reconocimiento y clasificación de imágenes usando arquitecturas de redes convolucionales y la librería Keras integrada en TensorFlow.

- Se ha comprobado que las redes convolucionales son capaces de aprender a reconocer imágenes con dibujos hechos a mano de forma rápida ("esbozos" de dibujos), además del reconocimiento de imágenes fotográficas.

- Se ha comprobado la influencia de la variación de distintos parámetros en la arquitectura de las redes convolucionales y su influencia en la precisión del reconocimiento y clasificación.

- Se han estudiado e implementado técnicas de representación del aprendizaje realizado por las redes y se ha comprobado que las representaciones intermedias tienden a reconocer determinados patrones según la profundidad del kernel.

- Se ha demostrado que las representaciones intermedias aprendidas por las redes pueden ser comprendidas de tal forma que pueden extraerse explicaciones o razonamientos sobre el reconocimiento realizado.

6.2. TRABAJOS FUTUROS

A continuación se enumeran algunos posibles trabajos futuros o de ampliación del presente proyecto.

- Implementación de un reconocedor de dibujos hechos a mano para un mayor número de muestras (por ejemplo, usando todas las clases disponibles en el conjunto de datos de Google Quickdraw). Para la realización de esta implementación se necesita una capacidad de cómputo mucho mayor de la que se disponía durante la realización del presente TFG.

- Implementación de técnicas adicionales para la interpretación del aprendizaje de las redes convolucionales como, por ejemplo, técnicas de visualización de filtros y generación de mapas de calor de las activaciones.

- Desarrollo de otra aplicaciones de las redes convolucionales para visión artificial como por ejemplo la segmentación de imágenes y la detección de objetos.

BIBLIOGRAFÍA

[1] A. Krizhevsky, I. Sutskever, y G. E. Hinton, «ImageNet classification with deep convolutional neural networks», *Commun. ACM*, vol. 60, n.º 6, pp. 84-90, may 2017, doi: 10.1145/3065386.

[2] T. N. Sainath, A. Mohamed, B. Kingsbury, y B. Ramabhadran, «Deep convolutional neural networks for LVCSR», en *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, may 2013, pp. 8614-8618. doi: 10.1109/ICASSP.2013.6639347.

[3] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *Bull. Math. Biophys.*, vol. 5, n.º 4, pp. 115-133, dic. 1943, doi: 10.1007/BF02478259.

[4] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain», *Psychol. Rev.*, vol. 65, n.º 6, pp. 386-408, 1958, doi: 10.1037/h0042519.

[5] M. Minsky y S. Papert, *Perceptrons*. Oxford, England: M.I.T. Press, 1969.

[6] D. E. Rumelhart, G. E. Hinton, y R. J. Williams, «Learning representations by back-propagating errors», *Nature*, vol. 323, n.º 6088, pp. 533-536, oct. 1986, doi: 10.1038/323533a0.

[7] Y. LeCun *et al.*, «Handwritten Digit Recognition with a Back-Propagation Network», en *Advances in Neural Information Processing Systems*, 1989, vol. 2. Accedido: 6 de septiembre de 2022. [En línea]. Disponible en: https://proceedings.neurips.cc/paper/1989/hash/53c3bce66e43be4f209556518c2fcb54-Abstract.html

[8] Y. Lecun, L. Bottou, Y. Bengio, y P. Haffner, «Gradient-based learning applied to document recognition», *Proc. IEEE*, vol. 86, n.º 11, pp. 2278-2324, nov. 1998, doi: 10.1109/5.726791.

[9] G. E. Hinton, S. Osindero, y Y.-W. Teh, «A Fast Learning Algorithm for Deep Belief Nets», *Neural Comput.*, vol. 18, n.º 7, pp. 1527-1554, jul. 2006, doi: 10.1162/neco.2006.18.7.1527.

[10] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», *ArXiv 14091556*, sep. 2014.

[11] C. Szegedy *et al.*, «Going Deeper With Convolutions», 2015, pp. 1-9. Accedido: 6 de septiembre de 2022. [En línea]. Disponible en: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html

[12] I. Goodfellow, Y. Bengio, y A. Courville, *Deep Learning*. MIT Press, 2016.