UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

RECONOCIMIENTO DE INDIVIDUOS Y EXPRESIONES FACIALES CON REDES NEURONALES

TESIS

Para obtener el título de:

Ingeniero en Computación

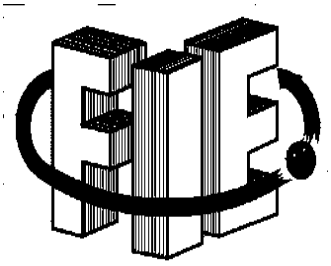
PRESENTA:

Abelino Osorio Martínez

ASESOR

MC. Luis Fernando Guzmán Nateras

Morelia Michoacán Agosto del 2018



Agradecimientos

A mi familia.

Por su cariño, compresión y apoyo, mis padres, esposa, abuela y hermano, por sus palabras de ánimo que hicieron que llegara a la meta, a la conclusión de mi licenciatura. Muchas gracias por brindarme la fortaleza necesaria para seguir adelante en la realización de mis metas y objetivos.

A la Universidad.

Por toda la formación recibida en el plano académico, por las enseñanzas de vida.

Por los amigos, compañeros y profesores.

Al MC. Luis Fernando Guzmán Nateras.

Por haberme bridado la oportunidad de recurrir a su capacidad y conocimiento, así como también haberme tenido toda la paciencia para guiarme durante todo el desarrollo de la tesis.

A los profesores

Por sus diferentes formas de enseñar, quienes me motivaron en varios sentidos a seguir adelante y sin su apoyo esto no hubiera sido posible.

Dedicatoria

A mis padres Martin Osorio Francisco, Micaela Martínez Osorio por ser el pilar fundamental en todo lo que soy, en toda mi educación, tanto académica, como de la vida, por su incondicional apoyo perfectamente mantenido a través del tiempo.

A Elvia Martínez Hernández por la paciencia y las palabras de apoyo durante todo este tiempo.

A mi hermano Manuel Osorio Martínez, que a pesar de ya no estar con nosotros siempre fue mi inspiración.

A mi Abuela Zenaida Francisca Cruz que con sus palabras y sabios consejos me han hecho llegar hasta este punto.

Todo este trabajo ha sido posible gracias a ellos.

Índice

Lista de imágenes

Lista de tablas

Resumen

En el presente documento se expone el diseño y la implementación de una Red Neuronal Convolucional (CNN) utilizada para realizar 2 tareas de clasificación de imágenes: el reconocimiento de individuos y el reconocimiento de expresiones faciales. El reconocimiento de individuos consiste en poder identificar a un individuo de manera visual. El reconocimiento de expresiones faciales tiene como objetivo determinar el estado de ánimo de una persona a partir de los gestos de su rostro.

Se utilizaron distintas arquitecturas para probar el desempeño de la CNN en ambas tareas. Se varía el número de capas de convolución y el número de capas completamente conectadas para conocer el impacto que tiene cada una de ellas en el modelo en cuanto la eficiencia de la clasificación.

Los resultados obtenidos son presentados gráficamente para mayor facilidad de su análisis. Éstos se encontraron adecuados, sí bien por debajo del estado del arte, pues cumplen con el objetivo de este trabajo de analizar el comportamiento y desempeño de una CNN en dos tareas complejas de visión computacional. Los resultados reafirman la importancia que tienen de los datos de entrenamiento, la tarea a realizar y la arquitectura de la red en el desempeño de la misma.

Palabras clave

Clasificación, Imágenes,

Visión, Computacional

Redes Neuronales Convolucionales.

Abstract

In this document we present the desing and implementation of a Convolutional Neural Network (CNN) used for 2 distinct image classification tasks: individual recognition and facial expression recognition. Individual recognition consists in being able to identify a person visually. Facial expression recognition has the objective of determining the emotions of a person from the expressions on their face.

Several distinct architectures were used to test the performance of the CNN in both tasks. The number of convolutional layers and fully-connected layers was varied in order to assess the impact that they each have on the model regarding classification efficiency.

The obtained results are presented graphically to ease their analysis. These results are considered adequate, if surely behind the current state-of-the-art, because they fulfill this project's objective of analysing the behaviour and performance of a CNN in 2 computer-vision complex tasks. The results reaffirm the importance that training data, task at hand and network architecture all have on the performance of the model.

Keywords

Image Classification

Computer Vision

Convolutional Neural Networks

Capítulo 1
Introducción

La inteligencia artificial (IA) es la inteligencia exhibida por maquinas que perciben su entorno y llevan a cabo acciones que maximizan sus posibilidades de éxito en algún objetivo [1].

Hoy en día es muy común utilizar sistemas que cuentan con inteligencia artificial para realizar sus funciones diarias como por ejemplo: celulares, refrigeradores, hornos microondas, cámaras, e inclusive medios de transporte.

Por lo general los métodos utilizados en la inteligencia artificial son una forma de aproximar la computación actual, con la forma en la que razonamos los seres humanos, tratando de emular ciertas características, como el poder “resolver problemas” y “aprender”.

Con el presente trabajo se pretende realizar un clasificador de imágenes utilizando una variante del modelo computacional llamado red neuronal, con el cuál una computadora puede obtener las características de una imagen analizando los pixeles, y clasificar dichas imágenes mediante un entrenamiento previo, gracias a que las redes neuronales pueden “aprender” en base a experiencia.

El elemento básico de un sistema neuronal biológico es la neurona. Un sistema neuronal biológico está compuesto por millones de neuronas organizadas en capas que a su vez están interconectadas. En una red neuronal el elemento esencial es la neurona artificial, la cual se puede organizar en capas. Finalmente una red neuronal junto con los interfaces de entrada y salida constituirá el sistema global de procesamiento.

Entre las motivaciones principales para la aplicación y el estudio de las redes neuronales artificiales se encuentran las redes neuronales convulucionales, una variante de las redes neuronales, utilizado en visión computacional que agrega un proceso llamado convolución, el cual de manera muy general se puede decir que extrae características de la imagen de entrada, las cuales le ayudan a identificar la clase a la que pertenece dicha imagen y hacer la predicción, aunque el proceso se explica más a detalle en el capítulo 2 correspondiente al marco teórico.

1.1 Planteamiento del problema

En la actualidad existen diversas aplicaciones cuyo funcionamiento está basado en el reconocimiento de patrones en imágenes aplicadas en sistemas de seguridad electrónica, acceso a internet, cajeros automáticos, teléfonos móviles etc. Entre estas aplicaciones se puede mencionar Google y Facebook.

Mientras que para los seres humanos, el hecho de reconocer patrones en imágenes, así como expresiones faciales es relativamente sencillo, para una computadora es un verdadero reto, por tal motivo se pretende poner a prueba un método de la inteligencia artificial llamado Redes Neuronales Convolucionales, para identificar las ventajas que presenta el hecho de integrar a la red neuronal cierto número de capas convolucionales y capas ocultas, de este modo poder identificar el número de capas óptimo para aprovechar el aprendizaje.

En base a la necesidad de que la red neuronal aprenda a identificar imágenes, se buscara entre las diferentes configuraciones de las redes, la que mejor se adapte y proporcione mejores resultados. El modelo tiene que ser capaz de resolver la tarea tan compleja de reconocer características dentro de una imagen y así poder realizar una clasificación correcta.

Lo anterior tiene la finalidad de llegar a una conclusión respecto al número de capas óptimo para que una red convolucional aprenda a extraer de manera correcta las características de las imágenes de entrenamiento y reconocer imágenes nuevas.

1.2 Hipótesis

A través del proceso de entrenamiento las redes neuronales convolucionales extraen características del lote de imágenes. La eficiencia de la red neuronal convolucional depende del número de capas convolucionales, el número de capas completamente conectadas y el set de datos para el proceso de entrenamiento de la red.

Al incrementar el número de capas convolucionales y capas completamente conectadas, incrementa también la eficiencia del modelo.

1.3 Antecedentes

En las últimas décadas, el desarrollo de nuevas tecnologías ha dado impulso al estudio de las redes neuronales artificiales para la solución de diversos problemas, enseguida se muestra una recopilación de algunos trabajos previos que centran su investigación en el área de las redes neuronales para clasificar imágenes.

- Aplicación de redes neuronales en la clasificación de imágenes

  Florencia Mihaich

  2014

  Este trabajo pretende comprar la efectividad y eficiencia de métodos basados en redes neuronales artificiales respecto a procedimientos ampliamente usados para categorizar imágenes. Mediante la implementación de un software de la clasificación de imágenes ANNIC (Artificial Neural Network Image Classification) usando, en la fase de entrenamiento, una red neuronal artificial perceptrón multicapas, un mapa de auto organización de Kohonen (red SOM) y el tradicional algoritmo K-means [2].

- Desarrollo de un sistema basado en un clasificador neuronal para reconocimiento de orugas

  Marco Antonio Rodríguez Flores

  2010

  Se realiza la construcción del sistema a partir de un conjunto de imágenes de entrenamiento y prueba con diferentes imágenes de orugas, estas imágenes están guardadas en una base de datos y usan redes neuronales para identificar el tipo de oruga [3].

- Técnicas de reconocimiento facial mediante redes neuronales

  Enrique Cabello Pardos

  2004

  En esta tesis se presenta las soluciones que se tienen para la verificación facial. Se abordan técnicas basadas en imágenes bidimensionales y se muestra un modelo tridimensional, además de un experimento que permite dar una idea del funcionamiento de un sistema de verificación facial basado en datos tridimensionales, mediante caras de 3 dimensiones [4].

- Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional

  Francisco José Núñez Sánchez-Agustino

  2016

Este trabajo se centra en reconocer el contenido de las matrículas de los vehículos a partir de las imágenes capturadas por una cámara fotográfica [5].

- Interpretación del lenguaje de señas utilizando redes neuronales

  Carlos Alberto Monares Zabaleta

  2017

  El objetivo general es la creación de un sistema que permita interpretar los signos utilizados en un lenguaje de señas para facilitar la comunicación de las personas que padecen de pérdida de audición incapacitante [6].

1.4 Objetivos

1.4.1 Objetivos generales

El objetivo de este trabajo es realizar un clasificador de imágenes mediante redes neuronales convolucionales utilizando varias capas ocultas, y comparar la efectividad de cada una de las redes con características diferentes, mediante tablas y gráficas que muestren los resultados obtenidos. De igual forma se pretende estudiar a fondo las redes neuronales convolucionales tanto en su aspecto teórico, como en la aplicación de éstas mismas para la solución de problemas.

1.4.2 Objetivos Específicos

En el actual apartado se menciona los objetivos particulares a los que se pretende llegar con el trabajo.

- Diseñar una red neuronal que tenga la capacidad de aprender y registrar el valor que muestre la efectividad de la misma, cuando se realicen las pruebas con imágenes nuevas.
- Comparar la eficiencia de la red neuronal convolucional al momento de tener más capas de convolucion, y registrar el efecto mediante tablas que demuestren las variaciones en diferentes corridas.
- Generar gráficas de resultados variando el número de capas completamente conectadas (fully connect) por cada capa convolucional agregada, de modo que pueda tener registro de los cambios que se generan al cambiar el número de capas completamente conectadas dentro de una red neuronal convolucional.
- Entrenar la red neuronal con un número de imágenes adecuado para poder hacer la clasificación de manera correcta.
- Aplicar una librería que facilite la implementación de redes neuronales.

- Determinar el optimizador que mejor se adapte para poder clasificar imágenes de manera correcta.

1.5 Descripción de los capítulos

El trabajo se muestra desarrollado en 5 capítulos los cuales se describen brevemente a continuación.

Capítulo 1 introducción: Se presenta una idea general del tema de tesis a desarrollar, motivaciones y se muestran algunos trabajos anteriores que se enfocan al tema de clasificación de imágenes con redes neuronales. Se hace mención también a los objetivos a los que se pretende llegar tanto generales, como específicos.

Capítulo 2 Marco teórico: Se hace mención a las bases de la inteligencia artificial, se mencionan conceptos que describen las diferentes arquitecturas de redes neuronales que se tienen en la actualidad y también contiene información necesaria para comprender el funcionamiento de la red neuronal convolucional, tanto la parte de la recepción de la imagen, los filtros, funciones de activación, y los procesos por los cuales pasa la imagen para poder tener un resultado que clasifique la imagen en alguno de las categorías de entrenamiento.

Capítulo 3 Diseño e Implementación: Durante el desarrollo del capítulo se expone paso a paso la implementación de la red neuronal convolucional para la clasificación de las imágenes, se describen las herramientas de apoyo utilizadas, lenguaje de programación en la cual se implementa la red neuronal, entornos de desarrollo, librerías, la porción de código donde se implementa la red neuronal y la forma de compilar el modelo.

Capítulo 4 Experimentos y Resultados: Se muestra el código con el cual se pone a prueba la red neuronal convolucional, se hacen corrimientos con variaciones en los números de capas completamente conectadas (fully connected), dando como resultado tablas y gráficas que muestran la efectividad de la red neuronal, permitiendo así la comparación entre cada una de las diferentes configuraciones.

Capítulo 5 Conclusiones: Se presentan las conclusiones a las cuales se ha podido llegar después de realizar las diferentes pruebas con diferentes configuraciones de la red neuronal convolucional y se proponen trabajos futuros a realizar a partir de los resultados obtenidos.

Capítulo 2
Marco teórico

Este capítulo contiene la información teórica, conceptos y definiciones necesarias para adentrarse en la inteligencia artificial, en especial a las redes neuronales. La Inteligencia Artificial siempre ha tenido como modelo las funcionalidades inteligentes del hombre, enfocándose en distintos aspectos. Su primera motivación fue intentar construir máquinas que pudieran pensar como el ser humano, o al menos emular alguna capacidad de tal modo que denotara cierta inteligencia [2]. Esta capacidad se basa en los conocimientos previos, la habilidad para manipular y reformular apropiadamente los conocimientos en base a los datos que se proporcionan.

Cuando se hace referencia a una inteligencia artificial quiere decir que el origen de estos conocimientos no es natural, sino que fue hecho por la mano del hombre, intentando simular la inteligencia humana mediante máquinas [2].

Wairren McCulloch y Walter Pitts (1943) han sido reconocidos como los autores del primer trabajo de IA. [2]. Partieron de tres fuentes: conocimientos sobre la fisiología básica y funcionamiento de las neuronas en el cerebro, el análisis funcional de la lógica proposicional de Kussell y Whitehead y la teoría de la computación de Turing. Propusieron un modelo constituido por neuronas artificiales, en el que cada una de ellas se caracterizaba por estar activada o desactivada.

2.1 Historia de las Redes Neuronales

A principios de los años sesenta había muchas expectativas de los alcances que se podrían tener con las redes neuronales, sin embargo no se lograron los resultados esperados y la inversión de tiempo y dinero en redes neurales decreció hasta los años ochenta. Durante esta década se empezó a retomar la estructura neuronal, ya que el equipo con que se contaba era muy superior al de los años sesenta.

La tecnología neuronal trata de reproducir el proceso de solución de problemas del cerebro. Así como los humanos aplican el conocimiento ganado con la experiencia a nuevos problemas o situaciones, una red neuronal toma como ejemplos problemas resueltos para construir un sistema que toma decisiones y realiza clasificaciones.

Los problemas adecuados para la solución con redes neuronales son aquellos que no tienen solución computacional precisa o que requieren algoritmos muy extensos como en el caso del reconocimiento de imágenes.

Alrededor de 1943 los investigadores Warren McCulloch y Walter Pitts propusieron el primer modelo simple de la neurona [2]

En las décadas de los cincuenta y los setenta, el movimiento en redes neurales fue liderado por B. Widrow y M. E. Hoof., [2]quienes trabajaron con una máquina llamada Adaline (Adaptive Linear Element).

En 1959, Rosenblatt construyó una máquina neural simple que llamó perceptrón [3]. Ésta tenía una matriz con 400 fotoceldas que se conectaban aleatoriamente a 512 unidades tipo neurona. Cuando se representaba un patrón a las unidades sensoras, éstas enviaban una señal a un banco de neuronas que indicaba la categoría del patrón. El perceptrón de Rosenblatt reconoció todas las letras del alfabeto.

Al final de los años setenta Minsky y Papert [3] demostraron que los perceptrones eran incapaces de hacer tareas simples tales como sintetizar la función lógica XOR.

En 1986 Mc Clelland y Rumelhart [3] publicaron un libro en dos volúmenes titulado: Parallel Distributed Processing: Explorations in the Microstruture of Cognition. Este libro se considera un clásico en el área de redes neurales y se puede decir que su aparición significó un nuevo impulso a la investigación en sistemas neurales al mostrar las ventajas y desventajas de las redes neurales artificiales (RNA).

2.2 Fundamento Biológico

Como las redes neuronales son una forma de simular el proceso neuronal que realiza el cerebro humano, es importante explicar la forma en que las neuronas biológicas aceptan la información y la procesan. La misión de las neuronas biológicas comprende generalmente cinco funciones que se enlistan a continuación.

- Recogen la información que llega a ellas en forma de impulsos procedentes de otras neuronas
- La integran en un código de activación propia de la célula
- Transmiten información codificada en forma de frecuencia de impulsos a través de su axón
- A través de sus ramificaciones, el axón efectúa la distribución espacial de los mensajes
- En sus terminales transmite los impulsos a las neuronas subsiguientes o a células efectoras.

Enseguida se describe la forma en la que una neurona biológica procesa la información que llega a ella, y se compara con la neurona biológica, la imagen 2.1 muestra una forma de representar esa neurona artificial y muestra también una neurona biológica.

El procesamiento de las señales empieza cuando las sinapsis recogen información electroquímica procedente de las células vecinas a las que está conectada, para la neurona artificial este proceso es la recepción de los datos de entrada.

La información llega al núcleo donde es procesada hasta generar una respuesta que es propagada por el axón, esa parte de la neurona artificial es la sumatoria de los datos de entrada multiplicados por los pesos de entrada, la cual se le aplica una función de activación.

Más tarde la señal única propagada por el axón es ramificada y llega a las dendritas de otras células través de lo que se llama sinapsis, esa parte es la salida de la neurona artificial.

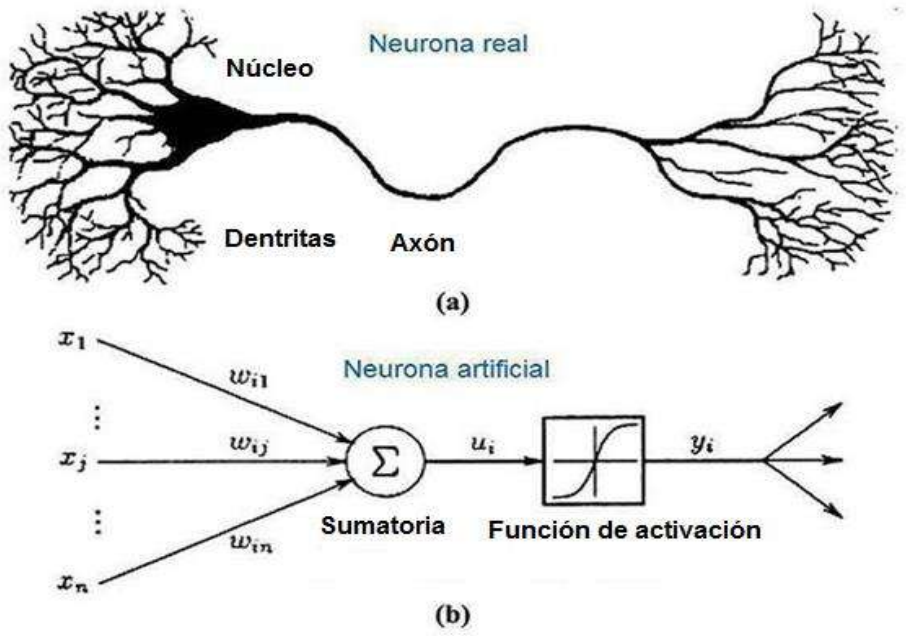


Imagen 2. 1 Comparación de (a) una neurona biológica y (b) neurona una artificial

2.3 Arquitecturas de Redes Neuronales Artificiales

Se denomina arquitectura a la topología, estructura o patrón de conexión en una red neuronal, por lo general las neuronas se suelen agrupar en unidades estructurales llamadas capas y el conjunto de una o más capas construye una red neuronal.

Considerando la estructura de la red se puede hablar de redes monocapa, compuestas por una única capa de neuronas, o redes multicapa donde las neuronas se organizan en varias capas. Teniendo en cuenta el flujo de datos, se pueden tener redes unidireccionales donde la información fluye en un solo sentido y redes recurrentes o retroalimentados donde la información fluye en distintas capas de la red en cualquier sentido.

2.3.1 Perceptrón

El perceptrón es un modelo que es capaz de realizar tareas de clasificación de forma automática. La idea inicial era de disponer de un sistema que a partir de un conjunto de ejemplos de clases diferentes, éste fuera capaz de determinar las ecuaciones de las superficies que hacían de frontera de dichas clases [4]

La arquitectura es muy simple como se muestra en la imagen 2.2. Es una estructura mono capa, donde hay un conjunto de entradas, tantas como sea necesario, dependiendo de los términos del problema, cada una de las entradas tiene conexión con cada una de las células de la capa de la salida y son estas conexiones las que determinan las superficies de discriminación del sistema.

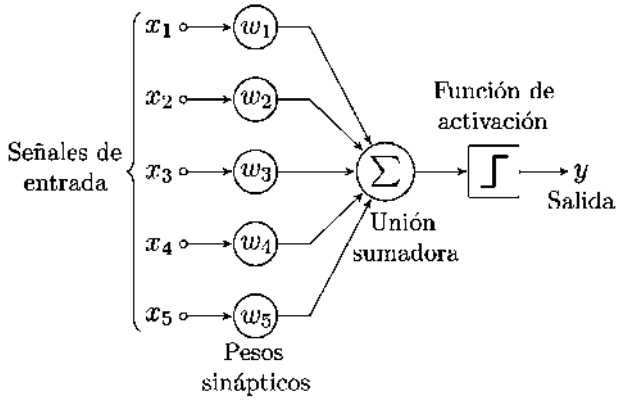


Imagen 2. 2 Ejemplo de un perceptrón

En el modelo perceptrón de ejemplificado en la imagen 2.2, la salida de la red se obtiene mediante el siguiente procedimiento:

- Se calcula la activación de la célula de salida mediante la suma de todas las entradas multiplicadas por los pesos de entrada.

$$Y' = \sum_{i=1}^{n} W_i X_i$$

- La salida definitiva del perceptrón se produce al aplicarle la función de activación de la célula y, si llega al valor del umbral θ, la neurona se activa, de lo contrario, no.

$$Y = F(Y', \theta)$$

2.3.2 Redes Multicapa

La clasificación según la topología o arquitectura de una red, consiste en la organización y la disposición de las neuronas en la red. Las neuronas se agrupan formando capas, que pueden tener muy distintas características. Además, las capas se organizan para formar la estructura de la red.

Las redes multicapa están formadas por varias capas de neuronas (2,3...n). Las redes multicapa pueden clasificarse dependiendo de la forma en la que fluye la información: propagación hacia adelante y retropropagación los cuales se describen enseguida [4].

- Conexiones Feedforward o hacia delante: Usualmente, las capas se encuentran establecidas por el orden en que reciben la señal desde la entrada hasta la salida y están unidas en ese orden. Como ejemplo de este tipo de red se tiene:

  El perceptrón multicapa las conexiones entre neuronas son siempre hacia delante, las conexiones van desde las neuronas de una determinada capa hacia las neuronas de la siguiente capa; no hay conexiones laterales ni conexiones hacia atrás. Por tanto, la información siempre se transmite desde la capa de entrada hacia la capa de salida. La topología básica se muestra en la imagen 2.3, que es una red neuronal de 1 capa de entrada, 1 capa oculta y 1 capa de salida.

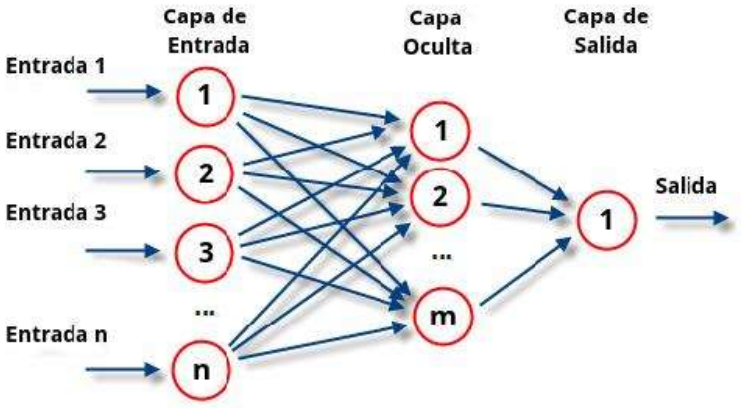


Imagen 2. 3 Perceptrón multicapa

- Feedback o retroalimentadas: Estas redes por el contrario cuentan con capas que están unidas desde la salida hasta la entrada en el orden inverso en que viajan las señales de información.

2.4 Funciones de activación

Para las redes neuronales artificiales, la regla que logra establecer el efecto total de las entradas en la neurona se denomina función de activación [2], la Función de Activación define la salida de una neurona, dada una entrada o un conjunto de entradas. Algunas de las funciones de activación más utilizadas en los distintos modelos de redes neuronales son:

Función Identidad

Es una función de activación muy simple que siempre devuelve como salida su valor de entrada, su rango es (-∞, +∞) y la función que la describe se muestra a continuación además de la gráfica que representa su comportamiento se encuentra en la imagen 2.4

$$F(x) = x$$

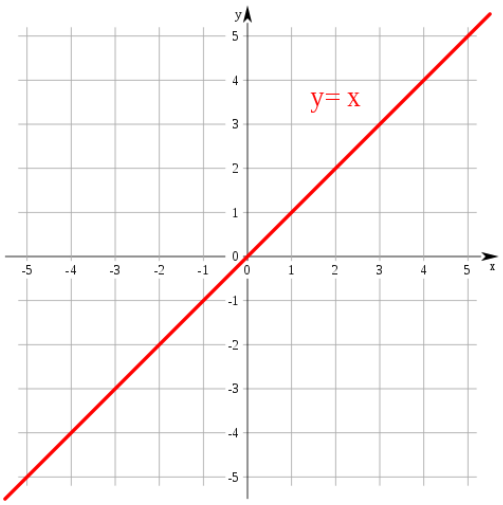


Imagen 2. 4 Gráfica de la función identidad

Función Escalón Unitario

Es la función de activación más usada por redes neuronales binarias, ya que produce un resultado cero cuando el valor de entrada es menor a cero y uno cuando el valor de entrad es mayor o igual a cero, como se muestra a continuación, además en la imagen 2.5.

$$F(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

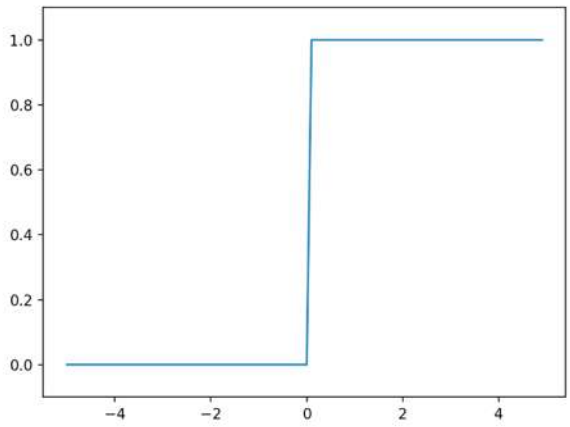


Imagen 2. 5 Gráfica de la función escalón unitario

Función Sigmoide

La función sigmoidal posee un rango comprendido entre 0 y 1. Esto, aplicado a las unidades de proceso de una red neuronal artificial significa que, sea cual sea la entrada, la salida estará comprendida entre 0 y 1, la gráfica que describe el comportamiento de esta función se muestra en la imagen 2. 6.

$$F(x) = \frac{1}{1 + e^{-x}}$$

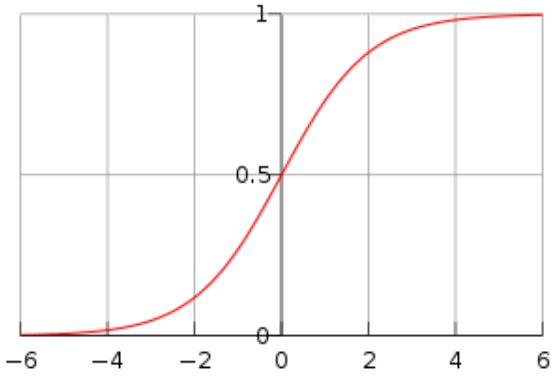


Imagen 2. 6 Gráfica de la función sigmoide

Función Tangente Hiperbólica

Esta función es utilizada por redes con salidas continuas. Un ejemplo será el perceptrón multicapa con retropropagación, ya que su algoritmo de aprendizaje necesita una función derivable y la gráfica de la imagen 2.7 describe el comportamiento de la función.

$$F(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

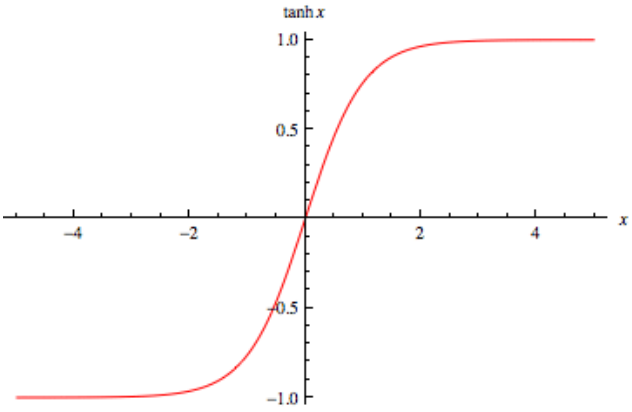
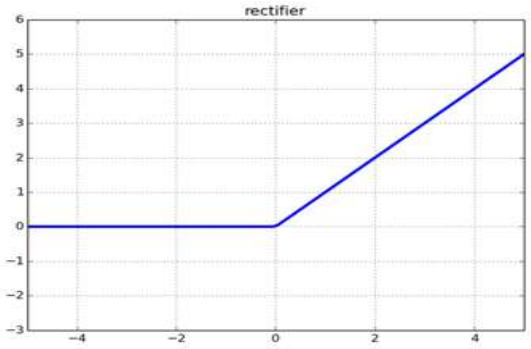


Imagen 2. 7 Función tangente hiperbólica

ReLU

La función de activación ReLU se introdujo por primera vez en el año 2000, por Hahnloser, con motivaciones biológicas y matemáticas, es muy utilizado en un tipo de redes llamado convolucionales que se describirá más a detalle en el la sección 2.6, la gráfica que describe la función se muestra en la imagen 2.8

$$F(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$



.

Imagen 2. 8 Gráfica de la función ReLU

Softplus

Es una función de aproximación suavizada de la función rectificadora ReLU y la comparación se muestra en la imagen 2.9 la línea de color verde es Softplus y azul ReLu.

$$F(x) = \ln(1 + e^x)$$

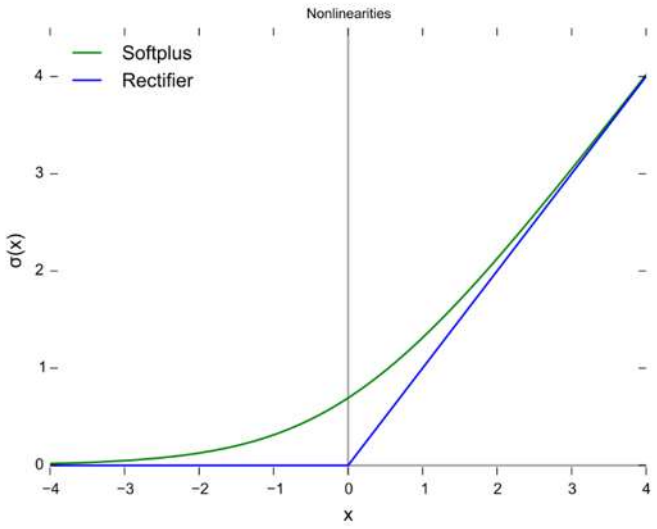


Imagen 2. 9 Gráfica de la función Softplus

2.5 Tipos de Aprendizaje

Las redes neuronales mediante el proceso de entrenamiento, ajustan los pesos de interconexión de sus neuronas con el objetivo de poder realizar una predicción lo más acertada posible. Enseguida se describen los diferentes enfoques de aprendizaje que se utilizan en la actualidad para llegar a una solución satisfactoria.

2.5.1 Aprendizaje supervisado:

La red trata de minimizar un error entre la salida que calcula y la salida deseada que es un valor conocido, de modo que la salida calculada termine siendo la deseada.

El aprendizaje supervisado se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo que determina la respuesta que debería generar la red a partir de una entrada determinada.

El supervisor controla la salida de la red y en caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada, enseguida se mencionan algunos tipos de aprendizaje.

2.5.2 Aprendizaje no supervisado o auto organizado

La red conoce un conjunto de patrones sin conocer la respuesta deseada. Debe extraer rasgos o agrupar patrones similares.

En este tipo de redes la salida representa el grado de familiaridad o similitud entre la información que se le está presentando a la entrada y las informaciones que se le han mostrado hasta entonces. Está constituido por un conjunto de reglas que dan a la red la habilidad de aprender asociaciones entre los patrones que ocurren en conjunto. Una vez que los patrones se han aprendido como asociación le permite a las redes realizar tareas útiles de reconocimiento de patrones y la habilidad de recordar.

2.6 Redes Neuronales Convolucionales

Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico.Este tipo de red es una variación de un perceptron multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes. [5]

Las redes neuronales convolucionales forman parte de un conjunto de algoritmos conocidos como DeepLerning o Aprendizaje profundo, que intentan modelar abstracciones de alto nivel a partir de datos, usando arquitecturas compuestas por múltiples transformaciones no lineales.

En los últimos años se han propuesto varias arquitecturas nuevas de redes neuronales convolucionales. Sin embargo, muchos de ellos usan los conceptos principales de LeNet.

LeNet fue una de las primeras redes neuronales convolucionales que ayudó a impulsar el campo del aprendizaje profundo. Este trabajo pionero de Yann LeCun se llamó LeNet5 después de muchas iteraciones exitosas anteriores desde el año 1998. En ese momento, la arquitectura LeNet se usaba principalmente para tareas de reconocimiento de caracteres, como leer códigos postales, dígitos, etc.

Las Redes Neuronales convolucionales están formadas por una estructura que consta de 3 Capas.

- Capa de Convolución
- Capa de Pooling
- Capa Completamente conectada.

En la imagen 2.10 se muestra un ejemplo de red convolucional. La red neuronal convolucional recibe una imagen de un barco como entrada, después a esa imagen se aplican 3 filtros de convolución y se le aplica una función de activación, se le reduce el tamaño en la capa de agrupamiento (pooling), este proceso se repite 2 veces, hasta que se introduce a una red neuronal completamente conectada de 2 capas con una salida de 4 neuronas donde se ve que la predicción es de que la probabilidad de que sea barco es de 0.96.

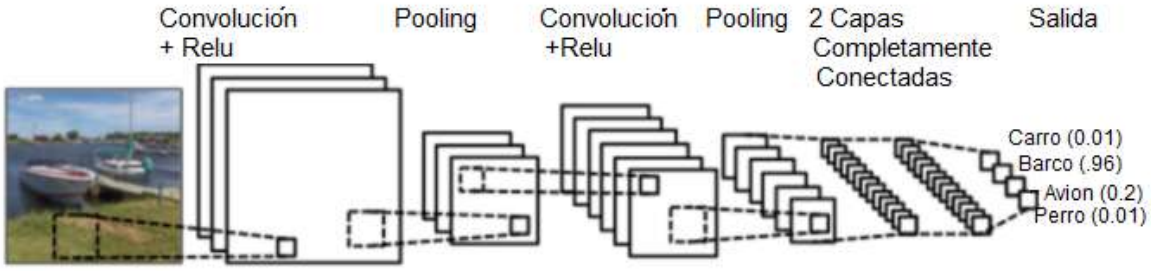


Imagen 2. 10 Arquitectura Red Convolucional

2.6.1 Capa de Convolución

El propósito principal de la Convolución en el caso de una CNN, es extraer características de la imagen de entrada. La imagen que se recibe es tratada como una matriz de datos, por lo tanto los valores pueden variar entre 0 y 1 cuando la imagen es blanco y negro o 0 y 255 para las imágenes a color.

La imagen 2.11 es una representación de una imagen de entrada vista como matriz, y está incluida para facilitar la comprensión del párrafo anterior, en esta imagen se muestra como cada pixel que representa la forma de la imagen, tiene un valor que corresponde a la intensidad de dicho color.

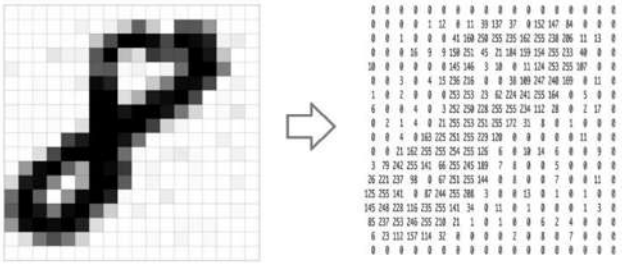


Imagen 2. 11 Matriz de valores de una imagen

La convolución es una operación de productos y sumas entre la imagen de entrada y un filtro (o kernel) que genera un mapa de características. Las características extraídas corresponden a cada posible ubicación del filtro en la imagen original. Una de las ventajas que presenta la convolución es que preserva la relación espacial entre los píxeles.

Una vez aplicado el filtro en la imagen, se genera un mapa de funciones o mapa de características que puede contener valores negativos por tal motivo se utiliza la función ReLU para sustituir valores de pixeles que tengan valores negativos por cero.

2.6.1.1 Filtro

Durante el proceso de entrenamiento, una CNN aprende los valores de los filtros por sí misma, aunque es importante mencionar que ya existen varios filtros predeterminados. Mientras más cantidad de filtros se tenga, más características de imágenes se extraerán y mejor será la red para reconocer patrones en las imágenes de entrada. A continuación se presenta la tabla 2.1 con algunos ejemplos de filtros, su nombre y su efecto.

Tabla 2. 1 Tabla de filtros y su efecto en una imagen

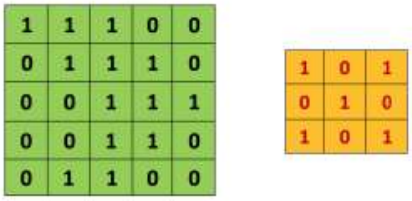
Operación	Núcleo	Resultado de la imagen
Identidad	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Detección de bordes	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Afilar	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Caja de desenfoque (normalizado)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (aproximación)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (aproximación)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Máscara de enfoque 5 × 5 Basado en desenfoque gaussiano con una cantidad como 1 y umbral como 0 (sin máscara de imagen)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Para ejemplificar el recorrido de un filtro en una imagen de entrada se utiliza la imagen 2.12, en ella se muestra una imagen binaria en forma de matriz con tamaño 5 x 5 pixeles marcado de color verde y también un filtro de tamaño 3 x 3 pixeles, el desplazamiento del filtro en la imagen es de un pixel. Los pasos que se ejemplifican en la imagen 2.12 son:

- El filtro se posiciona en la coordenada (0,0) de la imagen de entrada
- Se multiplica cada pixel de la imagen con el pixel correspondiente del filtro
- Se hace la sumatoria de los valores resultantes de la multiplicación, como se puede ver en la imagen 4 el primer valor da como resultado 4.
- El filtro se desplaza una posición a la derecha
- Una vez en la posición (1,0) el filtro multiplica los valores de los pixeles por los del filtro correspondiente.
- Se hace la sumatoria de los calores que resultan de la multiplicación dando como resultado 3
- El filtro se desplaza un pixel a la derecha quedando en la posición (2,0) y realiza la multiplicación de los pixeles.
- Se hace la sumatoria de los valores resultantes de la multiplicación y se obtiene como resultado 4.

- Al llegar al final de la imagen, el filtro se desplaza un pixel abajo e inicia nuevamente el recorrido iniciando en la posición (0,1).
- Una vez que el filtro está en la posición inicial (0,1) de la imagen, se multiplica cada pixel de la imagen con el pixel en la misma posición del filtro.
- Se hace la sumatoria del resultado de la multiplicación y se guarda el valor en la posición (0,1) de la matriz resultante
- Se realizan los mismos pasos hasta llegar al final de la imagen.



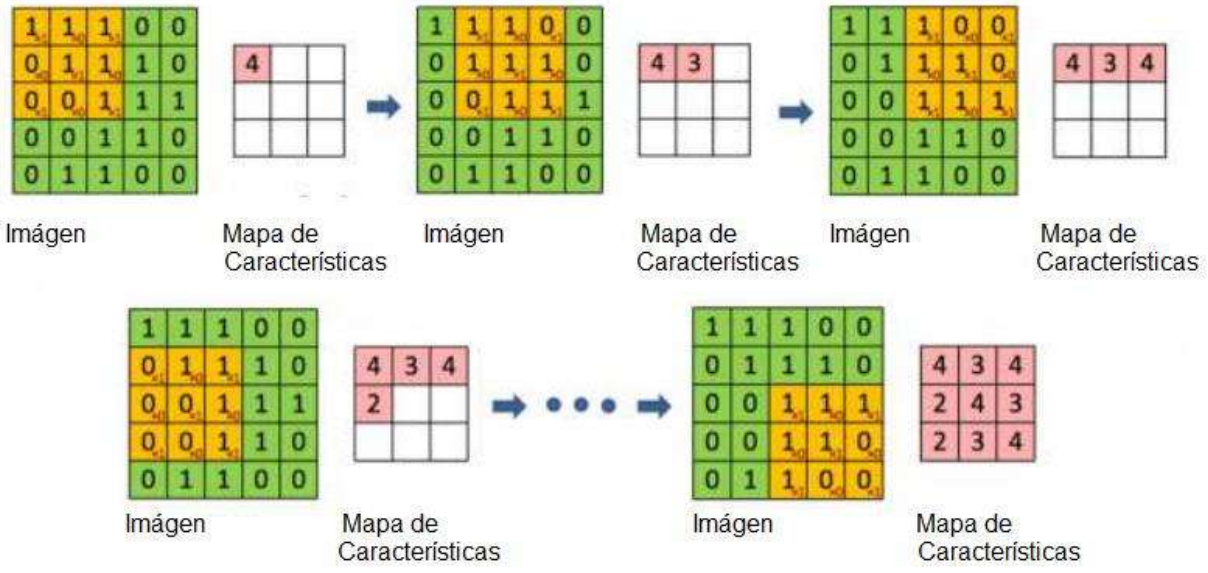


Imagen 2. 12 Ejemplo de convolución de una imagen

2.6.1.2 Mapa de Funciones

El tamaño del Mapa de funciones o Mapa de características está controlado por tres parámetros que debemos decidir antes de realizar el paso de convolución, y se describen a continuación:

- Profundidad: la profundidad corresponde a la cantidad de filtros que se usan para la operación de convolución. En la red de la imagen 3, se está realizando la convolución de la imagen de un barco con tres filtros ya que la imagen es RGB, produciendo así tres mapas de características diferentes. Puede pensarse en estos tres mapas de características como matrices 2d apiladas, por lo que la "profundidad" del mapa de características sería tres".
- Paso (Stride): Es la cantidad de píxeles con los que se desplaza la matriz de filtro sobre la matriz de entrada. Cuando el desplazamiento es 1, se mueve el filtro un pixel a la vez. Cuando el desplazamiento es 2, los filtros saltan 2 píxeles a la vez a medida que recorre la imagen.
- Relleno con ceros (Zero-padding): A veces, es conveniente rellenar la matriz de entrada con ceros alrededor del borde, de modo que se pueda aplicar el filtro a los elementos que bordean la matriz de imagen de entrada. Una buena ventaja de rellenar con ceros es que permite controlar el tamaño de los mapas de características. Cuando se aplica el relleno con ceros, el tamaño del mapa de características es el mismo que el de entrada pero, cuando no se aplica el rellenado con ceros el tamaño del mapa de características está dado por la siguiente función:

$$tamaño = n - f + 1$$

$$donde\ n\ es\ el tamaño\ de\ la\ imagen\ de\ entrada\ y\ f\ tamaño\ del\ filtro$$

2.6.3 Capa de Agrupamiento (Pooling)

La capa de agrupamiento reduce la dimensión de cada mapa de características, pero conserva la información más importante. El agrupamiento puede ser de diferentes tipos:

- Max : Se toma el valor máximo de un conjunto de datos pertenecientes a una región
- Promedio (Average) : Se calcula el promedio del conjunto de datos analizado
- Sum : Se realiza la suma del conjunto de datos de la región

Para realizar el agrupamiento se define un vecindario espacial (una ventana de tamaño n × n pixeles) y se toma el elemento más grande (en el caso de agrupamiento Max) del mapa de características rectificadas dentro de esa ventana. En lugar de tomar el elemento más grande, también se puede tomar el promedio o la suma de todos los elementos en esa ventana. Para ejemplificar el funcionamiento de pooling se muestra la imagen 2.13 en ella se utiliza:

- Maxpooling
- Mapa de características de tamaño 4 x 4 pixeles

- Una ventana de tamaño 2 x 2 pixeles
- Salto de 2 pixeles

En la imagen 2.13 la ventana o vecindario espacial, se posiciona en el inicio del mapa de características (0,0) y esa región se resalta utilizando el color rosa, después se obtiene el valor máximo de esa región que es el número 6 y se pasa a la matriz que está a la derecha, después el vecindario espacial se desplaza x + salto pixeles a la derecha, ese vecindario espacial esta sombreado de color verde, donde el máximo es el número 8, cuando se llega al final de la imagen, se continua con el recorrido en la posición y + salto como se ve en el mapa de características resaltado con la ventana amarilla que tiene un máximo 3, después se recorre la ventana con el tamaño del salto establecido que es 2 y esta sección se muestra resaltada de color azul, se extrae el máximo que es 4 y termina el proceso de pooling, dando como resultado una nueva imagen en forma de matriz de tamaño 2 x 2.

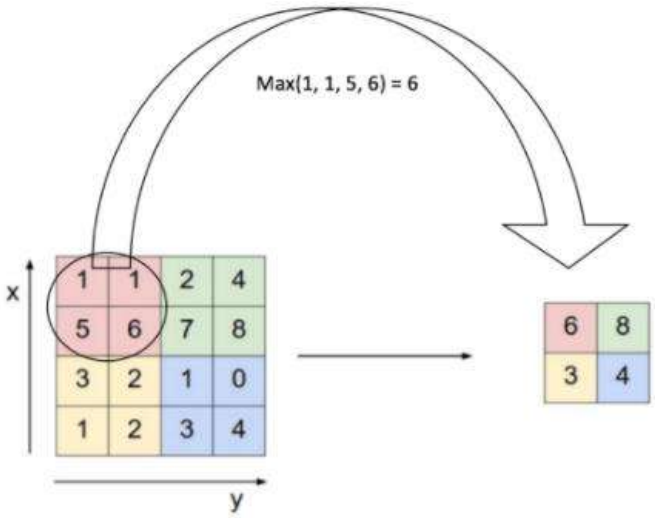


Imagen 2. 13 Ejemplo de agrupamiento

La función de Pooling es reducir progresivamente el tamaño espacial de la representación de entrada y algunas ventajas son:

- Hace que las representaciones de entrada sean más pequeñas y manejables.
- Reduce la cantidad de parámetros y cálculos en la red, por lo tanto, controla el sobreajuste.
- Hace que la red sea invariante para pequeñas transformaciones o distorsiones en la imagen de entrada ya que una pequeña distorsión en la entrada no cambiará la salida de Pooling, por que se toma el valor máximo, promedio o suma, en un vecindario local.
- Ayuda a llegar a una representación equivalente a escala de la imagen de entrada.

2.6.4 Capa Completamente Conectada (Fully Connected layer)

La capa Completamente conectada (Fully Connected) es un perceptrón de múltiples capas que utiliza una función de activación en la capa de salida. El término "Totalmente Conectado" implica que cada neurona en la capa previa está conectada a cada neurona en la siguiente capa.

La salida de las capas convolucionales y de agrupamiento representa características de alto nivel de la imagen de entrada. El objetivo de la capa completamente conectada es utilizar estas características para clasificar la imagen de entrada en varias clases, en función del conjunto de datos de capacitación.

La suma de las probabilidades de salida de la capa Totalmente Conectada es 1. Esto se garantiza mediante el uso de una función de activación llamado Softmax en la capa de salida de la Capa Completamente Conectada. La función de Softmax toma un vector de puntajes de valores reales arbitrarios y lo aplasta a un vector de valores entre cero y uno que suma a uno.

Las redes neuronales convolucionales proporcionan un resultado combinando todos los procesos explicados anteriormente, las capas Convolución + las capas de Pooling actúan como extractores de características de la imagen de entrada, mientras que la capa Totalmente Conectado actúa como un clasificador.

Para ejemplificar una salida en la capa completamente conectada se tiene la imagen 2.14 donde se tiene la imagen de un barco, donde se le aplica la convolucion, para detectar características y se aplica la rectificación mediante la función ReLU, después se le aplica pulling para reducir las dimensiones del mapa de características, ese proceso se repite una vez más, teniendo imágenes más pequeñas las cuales entran a la red completamente conectada, de 2 capas y una última capa con función de activación softmax que proporciona una respuesta al problema, y la suma de los resultados de la última capa es 1.

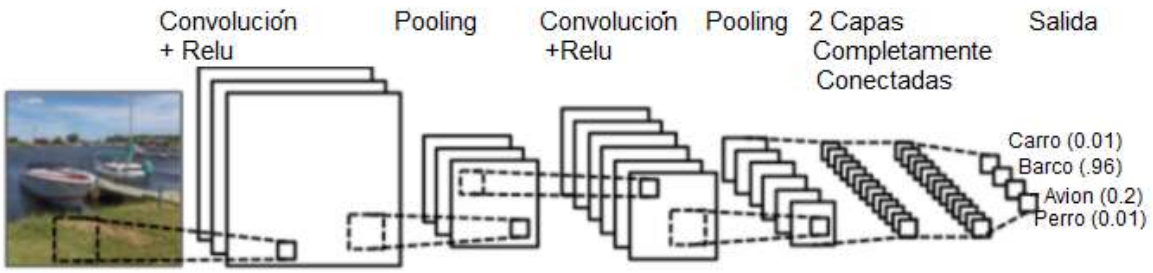


Imagen 2. 14 Representación de una Red Convolucional

2.7 Ventajas y Desventajas

Algunas ventajas de las RNA frente a otros sistemas de procesamiento de información son:

- Las Redes Neuronales pueden sintetizar algoritmos a través de un proceso de aprendizaje.
- Para utilizar la tecnología neural no es necesario conocer los detalles matemáticos. Sólo se requiere estar familiarizado con los datos del trabajo.
- La solución de problemas no lineales es uno de los fuertes de las RNA.
- Puenden fallar algunos elementos de procesamiento pero la red continúa trabajando.

Las desventajas de las redes neuronales son:

- Las Redes Neuronales se deben entrenar para cada problema. Además, es necesario realizar múltiples pruebas para determinar la arquitectura adecuada. El entrenamiento es largo y puede consumir varias horas de procesamiento de la computadora (CPU).
- Debido a que las redes se entrenan en lugar de programarlas, éstas necesitan muchos datos.
- Las Redes Neuronales representan un aspecto complejo para un observador externo que desee realizar cambios. Para añadir nuevo conocimiento es necesario cambiar las iteraciones entre muchas unidades para que su efecto unificado sintetice este conocimiento.

Algunas aplicaciones de las Redes Neuronales

• Automóviles: Sistemas de piloto automático. Detección de fallas por reconocimiento externo de vibraciones.

• Bancos: Lectura de cheques y otros documentos. Evaluación de aplicaciones de créditos.

• Electrónica: Predicción de secuencia de códigos. Control de procesos. Análisis de fallas. Visión artificial. Reconocimiento de voz.

• Finanzas: Tasación real de los bienes. Asesoría de préstamos. Previsión en la evolución de precios. Seguimiento de hipotecas. Análisis de uso de línea de crédito. Evaluación del riesgo en créditos. Identificación de falsificaciones. Interpretación y reconocimiento de firmas.

• Manufactura: Control de la producción y del proceso. Análisis y diseño de productos. Diagnóstico de fallas en el proceso y maquinarias. Identificación de partículas en tiempo real. Inspección de calidad mediante sistemas visuales. Análisis de mantenimiento de máquinas.

• Medicina: Análisis de células portadoras de cáncer. Análisis de electroencefalograma y de electrocardiograma. Diseño de prótesis. Optimización en tiempos de trasplante. Reducción de gastos hospitalarios.

• Robótica: Control dinámico de trayectoria. Robots elevadores. Controladores. Sistemas ópticos.

• Seguridad: Códigos de seguridad adaptivos. Criptografía. Reconocimiento de huellas digitales.

• Telecomunicaciones: Compresión de datos e imágenes. Automatización de servicios de información. Tradución en tiempo real de lenguaje hablado.

• Transporte: Diagnóstico de frenos en camiones, sistemas de ruteo y seguimiento de flotas.

Capítulo 3
Diseño e implementación

Este capítulo corresponde al diseño e implementación de una Red Neuronal Convolucional, la red tiene como propósito 2 cosas, reconocer emociones mediante imágenes de expresiones faciales y reconocer individuos, aunque la implementación de la red no está sujeta al set de datos, para las pruebas se utilizan dos set de datos distintos, uno con caras de personas mostrando una emoción y otro con personajes de "Los Simpson".

Por tal motivo se describirán las herramientas utilizadas, la configuración inicial del entorno de desarrollo paso a paso y la arquitectura.

3.1 Herramientas utilizadas

Para la implementación de la red neuronal convolucional se utilizaron herramientas que facilitaron la implementación del código como un IDE (Entorno de Desarrollo Integrado), un editor de texto, un lenguaje de programación, así también como librerías. Las herramientas son:

- Leguaje de programación Python
- IDE anaconda navigator
- Spyder
- Tensorflow
- Keras

Cada una de las herramientas mencionadas se describen a continuación.

3.1.1 Lenguaje de programación Python

Python es un lenguaje de programación de propósito general muy fácil de aprender, con una sintaxis característica que hace que los programas escritos en él sean muy legibles. Es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. El intérprete de Python y la extensa biblioteca estándar están a libre disposición en el sitio web de Python, [7] , y puede distribuirse libremente. El mismo sitio contiene distribuciones, enlaces de módulos libres de Python, programas, herramientas y documentación.

Python permite separar los programas en módulos que pueden reusarse en otros programas en Python. Viene con una gran colección de módulos estándar que se pueden usar como base, o

como ejemplos para empezar a aprender a programar. Este lenguaje de programación permite escribir programas compactos y legibles debido a que:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una menor cantidad de líneas
- La agrupación de instrucciones se hace por sangría en vez de llaves de apertura y cierre
- No es necesario declarar variables ni argumentos.

3.1.2 Anaconda Navigator

Anaconda es una interfaz de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la ciencia de datos con Python. En líneas generales Anaconda es una distribución de Python que funciona como un gestor de entorno, donde cada entorno puede tener las características y librerías necesarias para el trabajo a realizar.

Esta interfaz para la ciencia de datos con Python cuenta con una gran cantidad de características entre las que resaltan las siguientes:

- Libre, de código abierto, con una documentación bastante detallada.
- Multiplataforma (Linux, macOS y Windows).
- Permite instalar y administrar paquetes, dependencias y entornos para la ciencia de datos con Python de una manera muy sencilla.
- Ayuda a desarrollar proyectos de ciencia de datos utilizando diversos IDE como Jupyter, JupyterLab, Spyder y RStudio.
- Cuenta con herramientas como Dask, numpy, pandas y Numba para analizar Datos.
- Anaconda Navigator es una interfaz gráfica de usuario GUI bastante sencilla pero con un potencial enorme.
- Puede gestionar de manera avanzada paquetes relacionados a la ciencia de datos con Python desde la terminal.
- Elimina problemas de dependencia de paquetes y control de versiones.
- Está equipado de herramientas que permiten crear y compartir documentos que contienen código con compilación, ecuaciones, descripciones y anotaciones.
- Permite compilar Python en código de máquina para una ejecución rápida.
- Facilita la escritura de complejos algoritmos paralelos para la ejecución de tareas.
- Cuenta con soporte para computación de alto rendimiento.
- Los proyectos son portables, lo que permite compartir proyectos con otros y ejecutar proyectos en diferentes plataformas.
- Simplifica de manera acelerada la implementación de proyectos de ciencia de datos.

A continuación se tiene la imagen 3.1, en ella se muestra el entorno de desarrollo Anaconda, esta imagen es una captura de pantalla del aspecto inicial, y se pueden notar varios complementos con los que viene integrado desde la instalación.

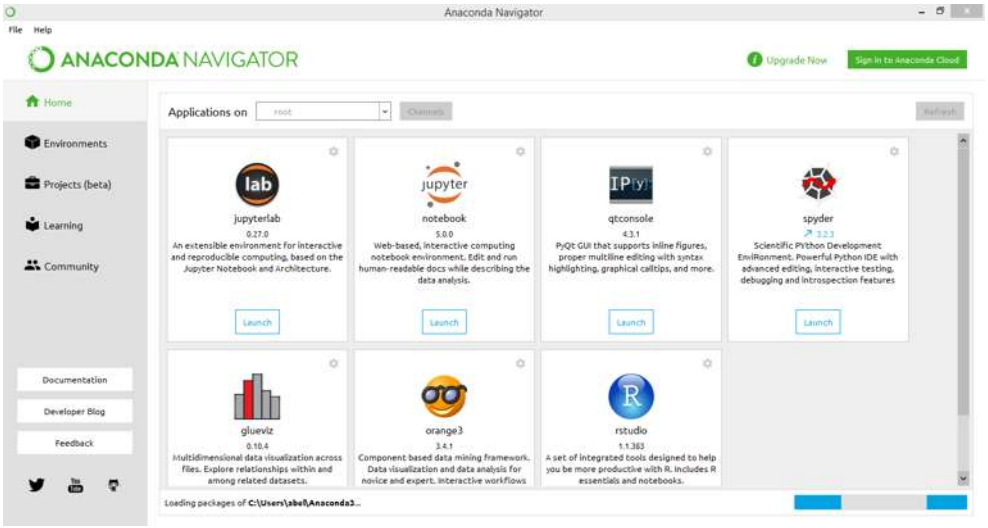


Imagen 3. 1 Inicio del IDE Anaconda Navigator

Una vez iniciado, se nota un apartado llamado Environments que significa “ambiente”, en ese apartado se selecciona el ambiente de trabajo configurado con las características necesarias para el desarrollo de la aplicación deseada, y al seleccionarlo proporciona la opción de agregar un entorno, que mejor se adapte a las necesidades del problema, dando la opción de importar librerías, versión de Python, editor de texto, incluso un nombre para identificar entre entornos creados, todo lo mencionado se puede ejemplificar en la imagen 3.2 en ella se tiene una captura de pantalla del entorno llamado base (root), ya con las especificaciones por defecto establecidas, como algunas librerías y la versión de Python.

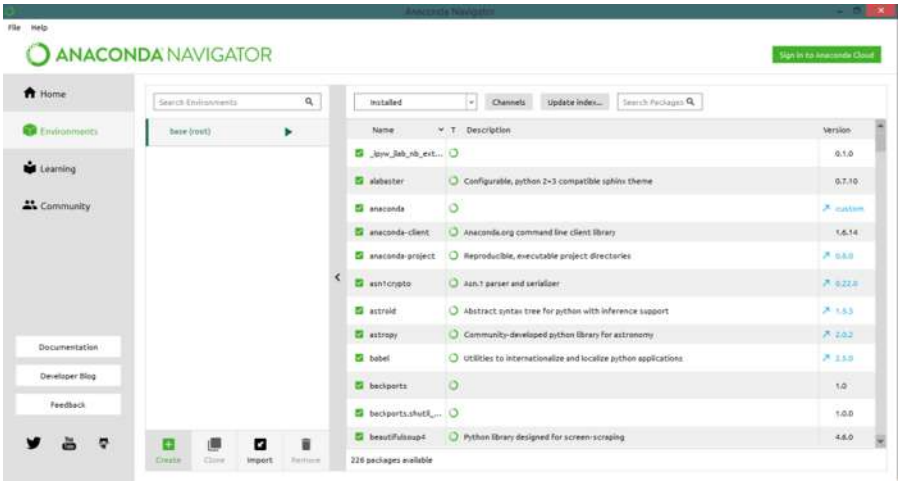


Imagen 3. 2 Elementos del ambiente de desarrollo root en anaconda

La imagen 3.3 muestra cómo se crea un nuevo ambiente de desarrollo, primero se presiona el botón con el símbolo de + que se ecuentra en la parte inferior con la leyenda “créate”, al realizar esa acción se despliega la opción que se esta visualidando en la imagen, donde se especifica el nombre del ambiente de trabajo y la versión de Python con la que se desea trabajar, por ultimo se presiona el botón de créate (crear) y asi se tiene el nuevo ambiente de trabajo.

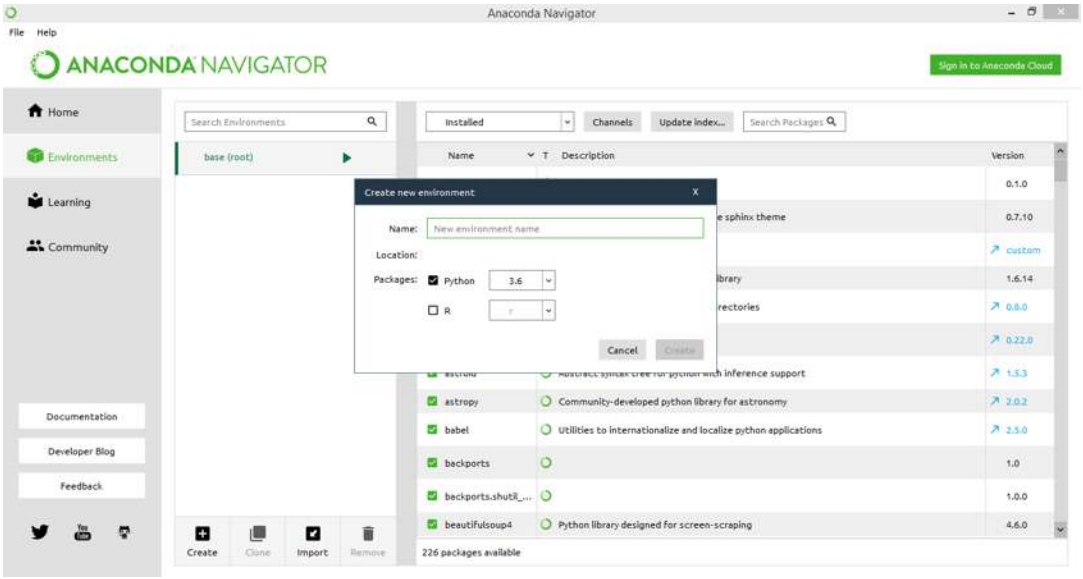


Imagen 3. 3 Ejemplo de la creación de un ambiente de desarrollo

3.1.3 IDE Spyder

Spyder es un Entorno de Desarrollo Integrado (IDE), multiplataforma de código abierto para programación científica en el lenguaje Python. Posee funciones avanzadas de edición, pruebas interactivas y depuración. A continuación se enlistan algunas ventajas:

- El editor que integra este IDE es multilenguaje.
- Consola interactiva.
- Poseé un visor de documentación. El programa puede mostrar documentación para cualquier llamada de clase o función realizada en el Editor o en una consola.
- Exploración de variables creadas durante la ejecución de un archivo.
- Posibilidad de buscar en archivos.
- Explorador de archivos para una mayor comodidad.
- Registro del historial de comandos.

La imagen 3.4 es una captura de pantalla del IDE Spyder donde se muestran algunos de los complementos que contiene por defecto al instalar anaconda.

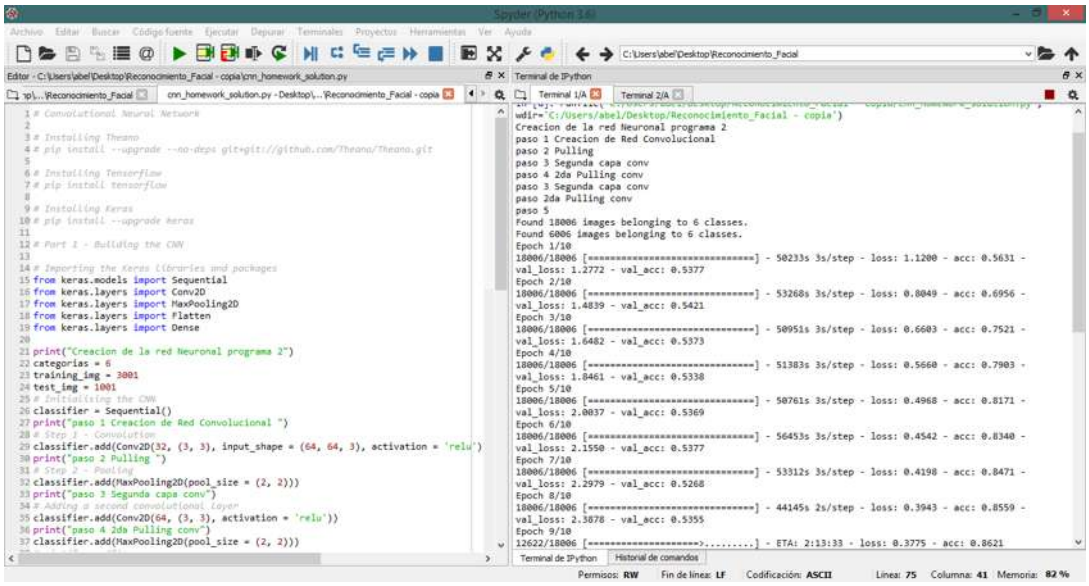


Imagen 3. 4 Captura de pantalla del IDE Spyder

3.1.4 TensorFlow

TensorFlow es una biblioteca de software de código abierto para el cálculo numérico de alto rendimiento. Su arquitectura flexible permite una fácil implementación de computación en una variedad de plataformas (CPU, GPU) y desde escritorios hasta clústeres de servidores, dispositivos móviles y periféricos.

Desarrollado originalmente por investigadores e ingenieros del equipo Google Brain dentro de la organización IA de Google, cuenta con un sólido respaldo para el aprendizaje automático y el aprendizaje en profundidad, y el núcleo de computación numérica flexible se utiliza en muchos otros dominios científicos.

3.1.5 Keras

Keras es una API de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano.

Fue desarrollado con un enfoque en permitir la experimentación rápida. Poder pasar de la idea al resultado con la menor demora posible es la clave para hacer una buena investigación.

Algunas ventajas:

- Ofrece API consistentes y simples, minimiza el número de acciones de usuario requeridas para casos de uso común y proporciona comentarios claros y procesables ante el error del usuario.

•Modularidad. Un modelo se entiende como una secuencia o un gráfico de módulos independientes y completamente configurables que se pueden conectar con la menor cantidad de restricciones posible. En particular, las capas neuronales, las funciones de costos, los optimizadores, los esquemas de inicialización, las funciones de activación y los esquemas de regularización son todos módulos independientes que puede combinar para crear nuevos modelos.

•Facilita la extensibilidad. Los nuevos módulos son simples de agregar (como nuevas clases y funciones), y los módulos existentes brindan amplios ejemplos. Para poder crear fácilmente nuevos módulos permite una total expresividad, lo que hace que Keras sea adecuado para la investigación avanzada.

•Trabaja con Python. No hay archivos de configuración de modelos por separado en un formato declarativo. Los modelos se describen en el código de Python, que es compacto, más fácil de depurar y permite facilidad de extensión.

3.1.6 Caracteristicas del equipo utilizado

El equipo utilizado para la implementación y las pruebas es una PC con 2 nucleos, una memoria ram de 4 Gigas, un sistema operativo Windows 8.1 de 64 bits y un procesador Intel Celeron con una velocidad de 2.16GHz.

3.2 Configuraciones iniciales

Para iniciar la implementación de la red neuronal es necesario configurar el entorno de desarrollo, el cual se describe paso a paso en esta sección.

3.2.1 Configuración de Anaconda Navigator

Para la realizarla instalación, es necesario ingresar a la página oficial de Anaconda Navigator [8], la imagen 3.5 muestra la página de inicio que se ve al ingresar a la dirección, también se nota que hay una parte resaltada de color verde que dice descargar Anaconda (Download Anaconda) esa opción es para iniciar la descarga.

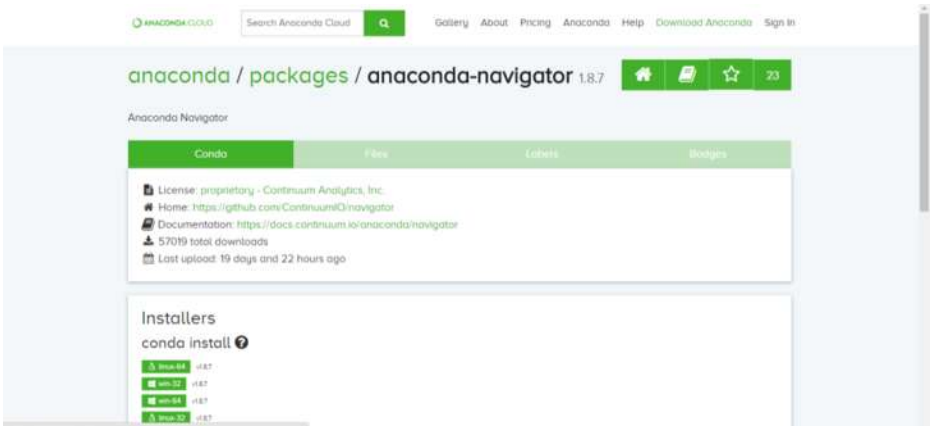


Imagen 3. 5 Página oficial de Anaconda Navigator

Se selecciona el sistema operativo en el que se va a instalar, la versión de Python con la que se desea trabajar y si el sistema operativo si es 32 o 64 bits, una vez descargado el instalador se inicia con el proceso de instalación, se aceptan términos y condiciones como se muestra en la imagen 3.6.

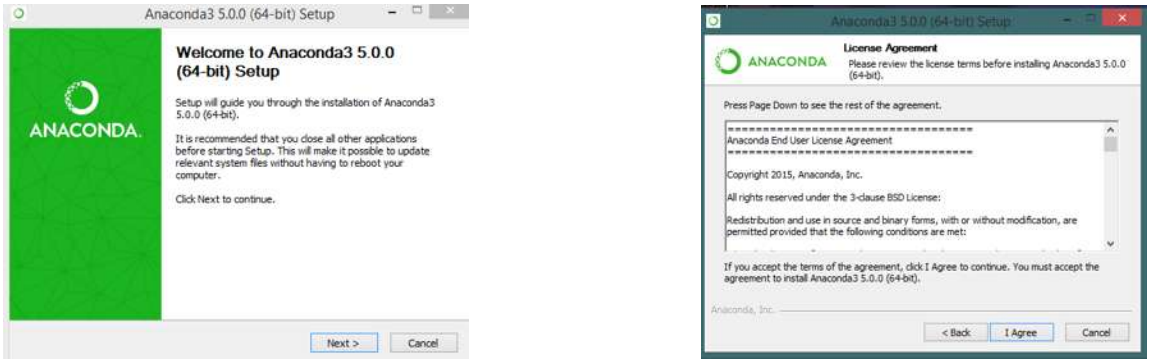


Imagen 3. 6 Instalación de Anaconda Navigator

Continuando con la instalación se selecciona la opción recomendada, y por último la carpeta de instalación como se muestra en la imagen 3.7.

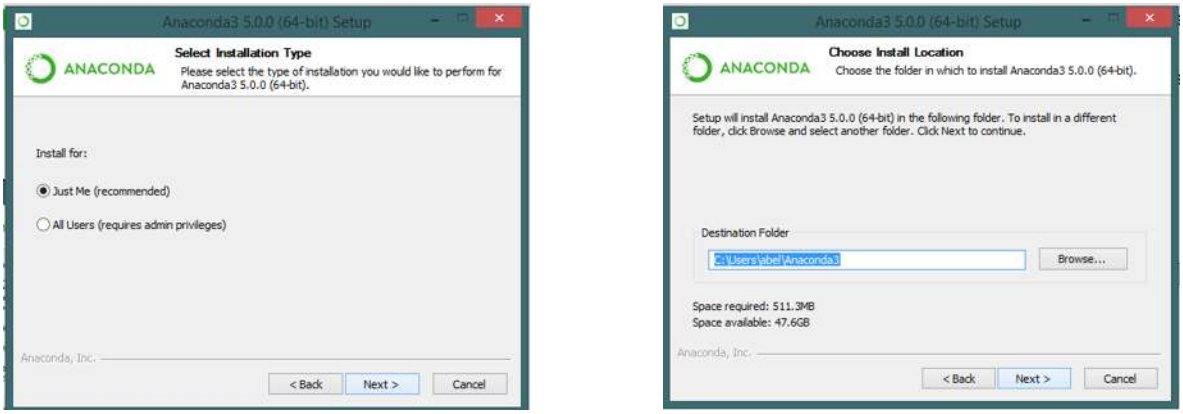


Imagen 3. 7 Paso final de instalación de Anaconda Navigator

3.2.2 Descarga de librerías

Para la descarga de librerías en Anaconda Navigator se pueden hacer desde el centro de comandos de Anaconda (Anaconda Prompt) o por la interface de usuario.

Desde el interface de anaconda, primero se ingresa al apartado Ambientes (environments), se despliega y se selecciona la opción de búsqueda, se selecciona la opción de no instalado (not installed) se escribe el nombre de la librería y por último importar (import) como se muestra en la imagen 3.8.

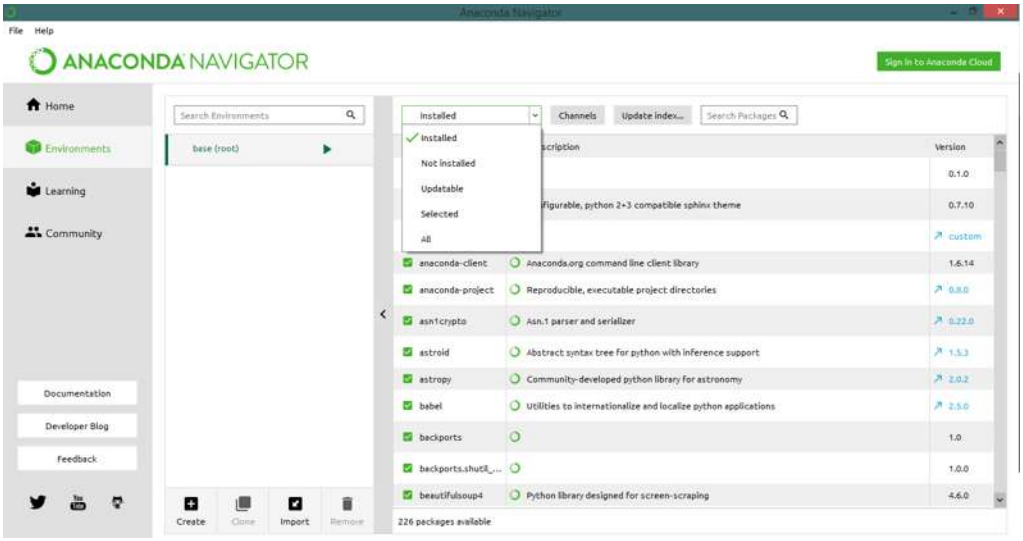


Imagen 3. 8 Descarga de librerías Anaconda Navigator

Para instalar la librería de Tensorflow con el centro de comandos de anaconda Anaconda Prompt o desde terminal se ingresa el siguiente comando:

conda create -n nombre pip python = versión_de_python

En el comando anterior se especifica el nombre del entorno de trabajo y se especifica la versión de Python.

Se activa el entorno creado con el siguiente comando

activate nombre

Despues de estar activo el entorno se inserta el siguiente comando que permite instalar la versión para CPU de Tensorflow.

(nombre)C:> pip install --ignore-installed --upgrade tensorflow

Para la Instalación de Keras en Anaconda es necesario introducir el siguiente comando dentro del entorno creado.

(nombre) C:> pip install Keras

Otra librería a instalar es la de numpy para el manejo de imágenes y se hace con el siguiente comando

(nombre) C:> conda install numpy

3.2.3 Preparación del conjunto de imágenes

Para importar los lotes de imágenes, es necesario que éstos se encuentren en una carpeta con el nombre “dataset”, también es necesario que la carpeta “dataset” se encuentre en la misma carpeta que el script de Python, como se muestra en la imagen 3.9.

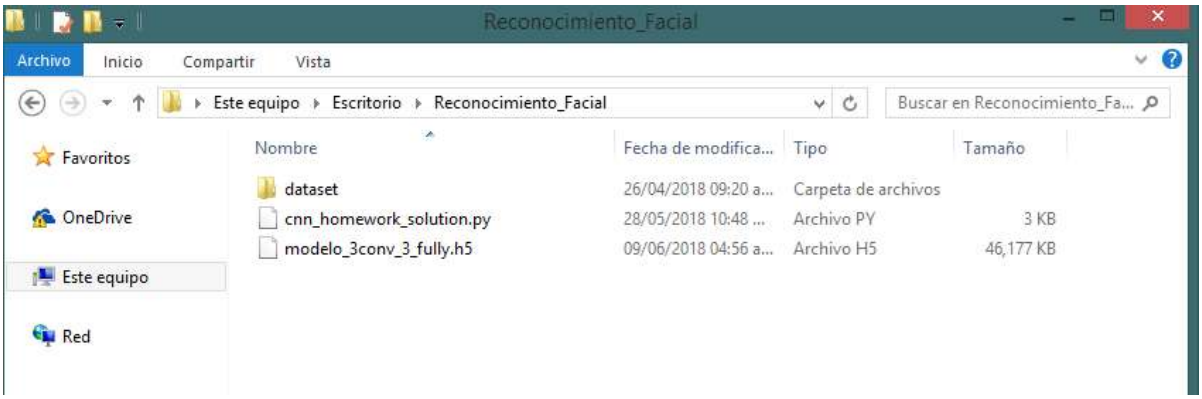


Imagen 3. 9 Carpeta dataset

Dentro de la carpeta “dataset”, es importante tener 2 carpetas, una llamada “training_set” y otra llamada “test_set”. Como se muestra en la imagen 3.10.

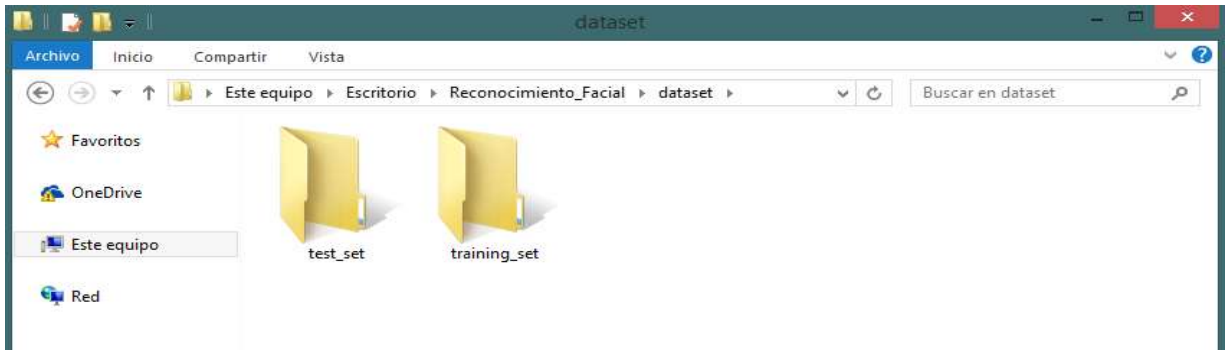


Imagen 3. 10 Carpetas dentro de dataset

La carpeta “training_set” debe contener las imágenes con las cuales la red neuronal va a entrenar, es importante crear subcarpetas por cada clase como se muestra en la imagen 3.11.

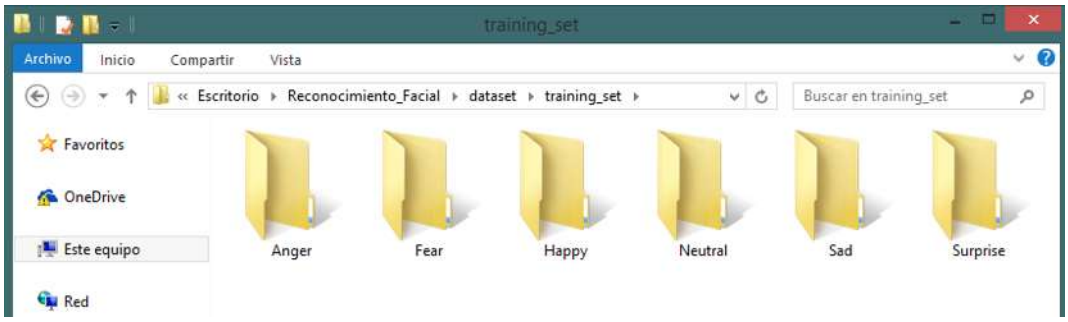


Imagen 3. 11 Subcarpetas dentro de la carpeta training_set

En la carpeta test_set, se debe crear el mismo número de carpetas que representan las clases como se muestra en la imagen 3.12.

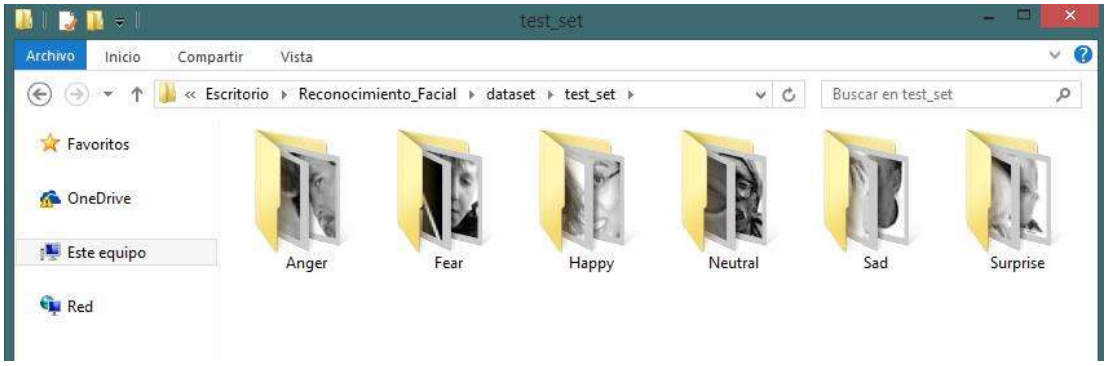


Imagen 3. 12 Contenido de la carpeta test_set

Las imágenes contenidas en las carpetas training_set y test_set deben ser diferentes, ya que en el proceso de prueba, Keras realiza mediciones de la precisión de la red neuronal utilizando las imágenes en la carpeta test_set, y lo almacena en la variable precisión (accuracy). La imagen 3.13 muestra como el contenido de las carpetas de test_set y training_set son diferentes para 2 set de datos diferentes.

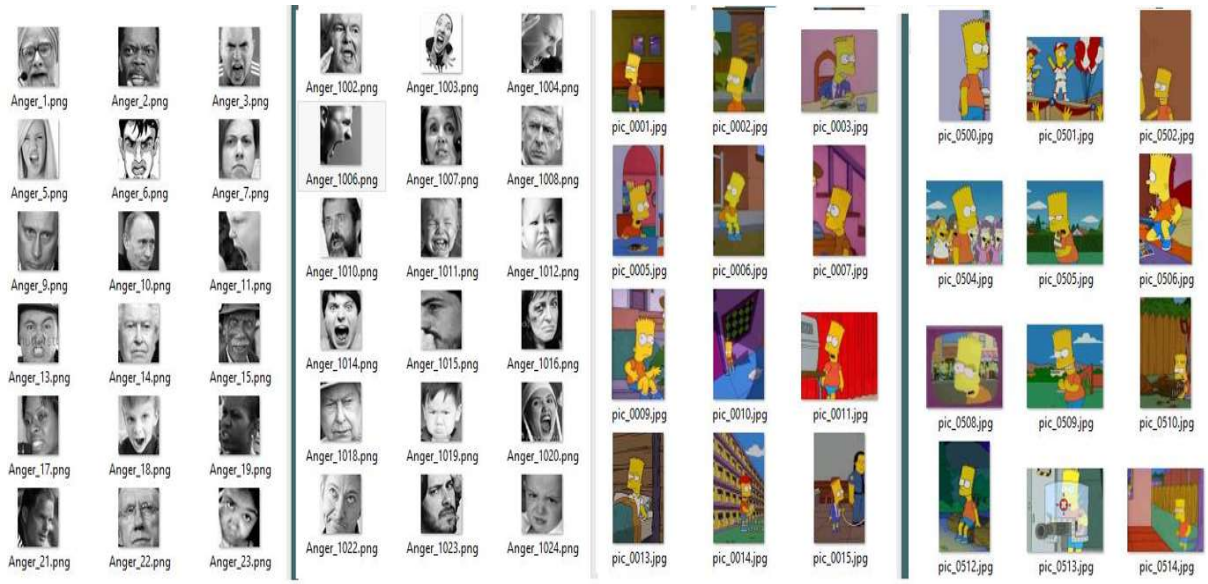


Imagen 3. 13 Ejemplo de imágenes dentro de carpeta Anger y la carpeta bart en test _set y training_set

Antes de introducir las imágenes a la red convolcional, es importante redimensionar el tamaño de cada uno, ya que dicha red neuronal recibe imágenes de 64x64 pixeles.

El siguiente código ejemplifica la forma de preparar el lote de imágenes para el proceso de entrenamiento.

```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

El código redimensiona las imágenes con la instrucción *target_size* dejando la imagen en 64 x 64 pixeles *batch_size* indica el número de imágenes que se toman por lote para el entrenamiento, *class_mode* define el tipo de clasificación si son 2 categorías se utiliza ‘binary’, pero como son 6 categorías se usa *'categorical'* y automáticamente reconoce las carpetas dentro de training_set.

3.3 Diseño de la Red Convolucional

La Red Neuronal Convolucional está implementada en Python y compone de 3 tipos de capas y una función de activación en la capa convolucional.

- Capa Convolucional + ReLU
- Capa Pooling
- Capa Completamente Conectada

3.3.1 Capa Convolucional

En esta capa se aplica la operación llamada convolución, donde se recibe la imagen de entrada y sobre ella se aplican filtros que regresan mapas de características.

Se utiliza una función después de la convolución llamada ReLU que es una operación de elemento inteligente (aplicada por píxel) y reemplaza todos los valores negativos de píxeles en el mapa de características por cero.

La imagen muestra una imagen de características, antes y después de aplicar la función ReLU.

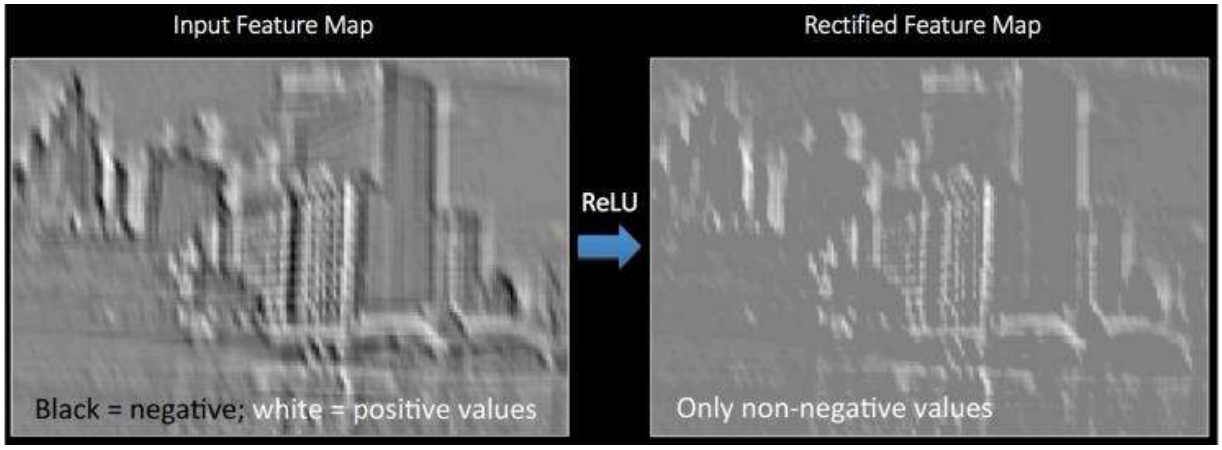


Imagen 3. 14 Aplicación de ReLU en mapa de características de una imagen

La Red Neuronal Convolucional aprende los valores de estos filtros por sí misma durante el proceso de entrenamiento, inicialmente aplica filtros aleatorios cercanos a cero y con el entrenamiento se ban ajustando los pesos para dar mejores resultados de predicción, aunque también se pueden utilizar los filtros preestablecidos mencionados en la sección 2.6.1.1 y la imagen 3.15 es una representación de este proceso, mostrando algunos puntos del recorrido del filtro en la imagen, en la primera parte se muestra como se recorre sin encontrar líneas, en la

segunda, se detecta el primer borde de la imagen, la tercera una esquina y el ultimo un borde inferior, todo este recorrido genera un mapa de características como se ve en la imagen debajo.

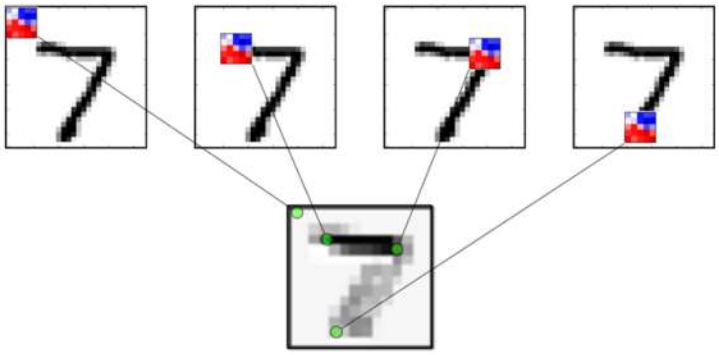


Imagen 3. 15 Ejemplo del proceso de convolución

En el siguiente fragmento de código se muestra como se genera una capa convolucional 2D, primero se importan las librerías necesarias, se definen el número de categorías en la variable *categorias*, el número de imágenes de entrenamiento con la variable *training_img*, el número de imágenes de prueba con la variable *test_img*, se crea el modelo que es la red neuronal convolucional y se asigna a la variable *classifier = Sequential()*, después se le agrega la primer capa convolucional al modelo con la instrucción *Conv2D,* para el código de ejemplo es de 32 filtros con tamaño de 3 x 3 pixeles, se especifica los datos de entrada, que son imágenes de tamaño 64 x 64 pixeles con la instrucción *input_shape*, el valor 3 es debido a que la imagen de entrada es RGB y por último se tiene la función de activación ReLU con la instrucción *activation*.

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
categorias = 6
training_img = 3001
test_img = 1001
```

```
classifier = Sequential()
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
```

3.3.2 Capa de Agrupamiento (Pooling)

En esta capa se reduce la dimensión de cada mapa de características, pero conserva la información más importante, para agregar una capa de Pooling en Keras se presenta a continuación el código, en el cual se utiliza maxpoopling de 2 Dimensiones con la instrucción *.add(MaxPooling2D())*, con un tamaño de 2 x 2 pixeles con la instrucción *pool_size = (2, 2)*.

```
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Establecidos los parámetros de la capa de pooling se muestra la imagen 3.16 que ejemplifica el proceso, la imagen de entrada es un mapa de características con tamaño 4 x 4 pixeles, la parte rosa, amarilla, azul y verde, son las regiones que recorre el vecindario establecido.

El barrido de la imagen de características inicia en la zona de color rosa, se aplica maxpooling y se obtiene como resultado el número 20, el filtro se recorre a la derecha como se ve en la parte de color amarillo, también se busca el mayor que es 30, al llegar al final de la imagen el filtro se desplaza hacia abajo, para cubrir las zonas faltantes e inicia nuevamente el recorrido de izquierda a derecha, como se puede observar la parte azul es la zona que se le extrae el valor máximo dando como resultado el número 112,el filtro se recorre hasta llegar a la zona de color verde y repite el procedimiento para entregar un máximo de 37, todos estos pasos generan una nueva imagen de características de menor dimensión.

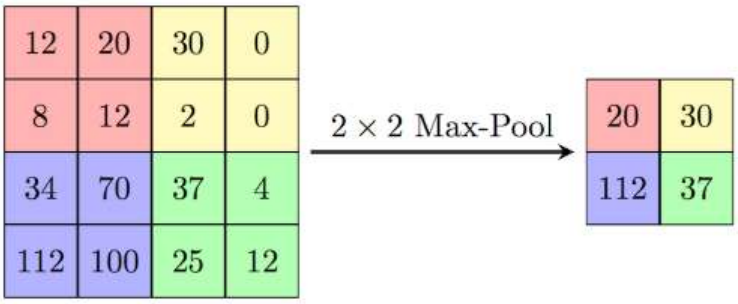


Imagen 3. 16 Ejemplo Max poolling 2x2

3.3.3 Capa Completamente Conectada (Fully Connected Layer)

Las capas completamente conectadas en Keras se agregan como *.add(Dense()*, en el siguiente fragmento de código se muestra una red neuronal ya declarada llamada *classifier* al que se le agrega una capa completamente conectada, pero cuando se están trabajando con matrices es importante convertir estos datos de entrada en vectores, por eso se utiliza la instrucción *.add(Flatten())* su trabajo es aplanar las matrices de características antes de enviarlas a la red completamente conectada, y después de eso es necesario especificar el número de neuronas, como en el código de ejemplo la red tiene 256 neuronas completamente conectadas se generan con la instrucción *units = 256*, y una función de activación que en la red de ejemplo es ReLU con la siguiente línea de código *activation= 'relu'*.

Adicional al número de capas completamente conectadas también se agrega la capa de salida a la red neuronal, como se muestra en la porción de código, el número de neuronas es igual al número de categorías o clases que se tengan, y la función de activación es softmax debido a que la suma de la salida esperada es 1.

```
classifier.add(Flatten())
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

3.3.4 Definición de parámetros para compilación

Para poder compilar una red neuronal en Keras, es importante definir un optimizador, una función de perdida, y una medida de precisión. A continuación se presentan algunos optimizadores:

- SDG (Stochastic Gradient Descent):

  Optimizador de descenso de gradiente estocástico. En el descenso de gradientes, un lote es la cantidad total de ejemplos que usas para calcular la gradiente en una sola iteración. Un lote muy grande puede causar que incluso una sola iteración tome un tiempo muy prolongado para calcularse. Al elegir ejemplos al azar de nuestro conjunto de datos, podríamos estimar (si bien de manera inconsistente) un promedio grande de otro mucho más pequeño. El término "estocástico" indica que el ejemplo único que compone cada lote se elige al azar.

- Adam :

Es un algoritmo de optimización que se puede usar en lugar del clásico procedimiento de descenso de gradiente estocástico para actualizar las ponderaciones de red de forma iterativa en función de los datos de entrenamiento. El nombre Adam se deriva de la estimación del momento adaptativo [2].

El método Adam calcula las tasas de aprendizaje adaptativo individuales para diferentes parámetros a partir de las estimaciones de los momentos primero y segundo de los gradientes.

- AdaGrad (Adaptive Gradient Algorithm)

  Gradiente adaptativo, permite que la velocidad de aprendizaje se adapte según los parámetros [2]. Realiza actualizaciones más grandes para parámetros infrecuentes y actualizaciones más pequeñas para los frecuentes. Debido a esto, es adecuado para datos dispersos (NLP o reconocimiento de imágenes). Otra ventaja es que básicamente elimina la necesidad de ajustar la velocidad de aprendizaje. Cada parámetro tiene su propia tasa de aprendizaje y, debido a las peculiaridades del algoritmo, la tasa de aprendizaje disminuye monótonamente. Esto causa el mayor problema: en algún punto del tiempo, la tasa de aprendizaje es tan pequeña que el sistema deja de aprender.

El valor que proporciona la función de pérdida (Loss) en Keras, es la diferencia entre el resultado de la red y el resultado correcto, una función de pérdida es la binaria (binary_crossentropy) la cual se utiliza cuando se desea identificar solo 2 clases, y se calcula con la siguiente función:

$$Error = -(y\log(p) + (1-y)\log(1-p))$$

Al tener más de 2 clases a identificar, se utiliza la función multiclase (categorical_crossentropy), el valor del error se ajusta al número de categorías creadas, y se calcula con la siguiente función.

$$Error = \sum_{C=1}^{M} ylog(p)$$

M: $Número\ de\ clases$
Y: $Resultado\ correcto$
P: $Predicción$

El otro dato que se define antes de ejecutar el modelo es la precisión (accuracy), se denomina accuracy al porcentaje de observaciones correctamente clasificadas. Esta métrica es válida tanto para problemas de clasificación binarios como multiclase.

El siguiente fragmento de código ejemplifica la forma de especificar los parámetros de compilación con la instrucción *.compile(),* el optimizador se especifica con la instruccion *optimizer* =, perdida cuando no es binaria la clasificación se utiliza *loss = categorical_crossentropy,* y la precisión accuracy en Keras con la instrucción *metrics = ['accuracy']).*

```
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

3.3.5 Preparación de datos de entrenamiento y pruebas

Una vez configurado el método de compilación, también es importante generar los datos de entrenamiento. Eso se puede realizar importando primero una librería llamada ImageDataGenerator, después se crea una variable "*train_datagen*" donde se configuran los lotes de imágenes para el entrenamiento, primero se ajustan los valores de los pixeles de la imagen a color para que los valores se encuentren en valores de 0 a 1 con la instrucción *rescale*, también se hace un recorte a la imagen con *shear_range*, después una ampliación con *zoom_range* y un movimiento suave con la instrucción *horizontal_flip* para tener más características.

El lote de imagenes para prueba de este ejemplo simplemente se le ajustan los valores de los pixeles y se asigna a la variable "*test_datagen*" como se muestra en el siguiente código.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

Ya creadas las configuraciones para entrenamiento y prueba, se crean las variables donde se guardaran los lotes de imágenes, primero se crea la de entrenamiento, la variable se llama "*training_set*" que recibe las imágenes con las configuraciones guardadas en la variable "train_datagen", esas configuraciones se les aplica a todas las imágenes en el directorio especificado, además de que se le redimensiona a 64 x 64 pixeles con la instrucción *target_size*,

se configura lotes de tamaño 32 con la instrucción *batch_size* y el tipo de clasificación multiclase con *class_mode*.

Los mismos procedimientos del entreamiento se realizan para las pruebas y se gradan los lotes en la variable "*test_set*" el ejemplo del código en Keras se muestra a continuación.

```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

3.3.6 Ejecución del modelo

Para poder ejecutar el modelo, se configuran los parámetros de entrenamiento con la función *fit_generator()* donde se especifica que, del lote de entrenamiento *training_set,* se tomaran todas las carpetas que representan una categoría, multiplicada por el número de imágenes que contiene cada carpeta, las validaciones se hacen mediante el lote de prueba "*test_set*" el cual se repite el número de clases para prueba multiplicado por el número de imágenes que contiene, ese proceso se repetirá 10 veces , para implementar lo descrito se muestra el siguiente código de Keras y el código completo se encuentra en el Anexo A.

```
classifier.fit_generator(training_set,
                         steps_per_epoch = training_img * categorias,
                         epochs = 10,
                         validation_data = test_set,
                         validation_steps = test_img * categorias)
```

Al iniciar la ejecución del modelo el IDE Spyder comenzara a mostrar información del entrenamiento como se purde observar en laparte derecha de la imagen 3.17.

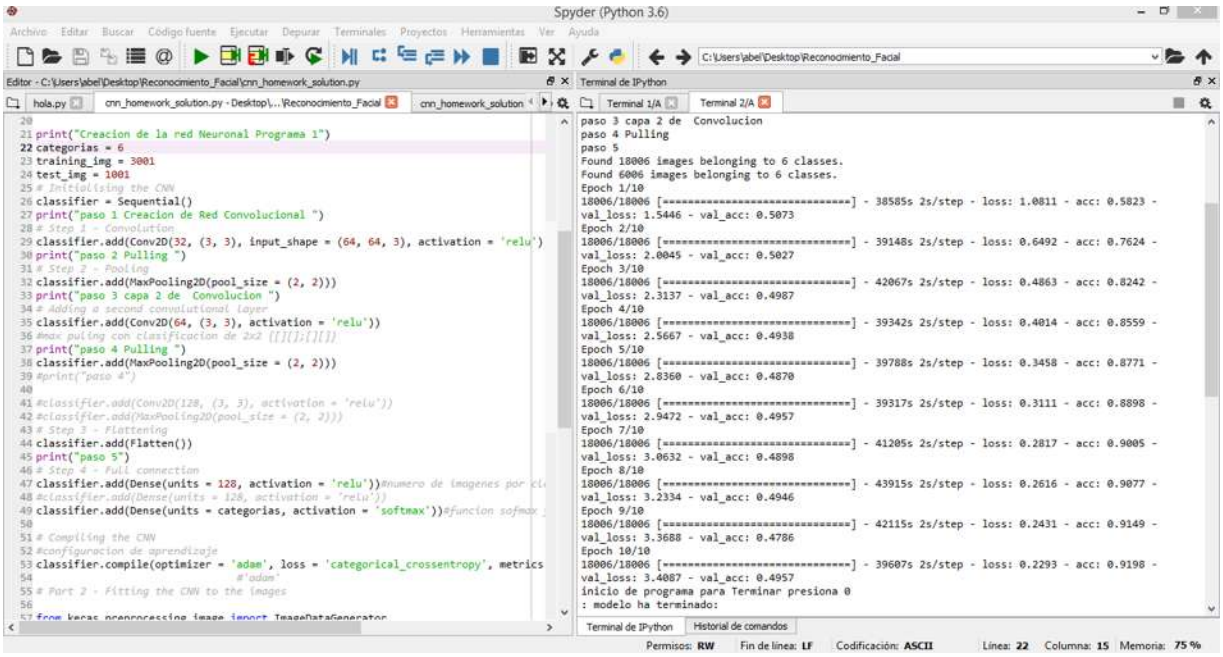


Imagen 3. 17 Ejemplo del corrimiento de un modelo en Spyder

Una vez implementadas las configuraciones para el funcionamiento de la red neuronal ya se pueden realizar las pruebas utilizando un set de datos representativo del problema que se desea resolver, para este caso se utilizan imágenes de personas mostrando diferentes expresiones faciales que representan una emoción y la tarea de la red neuronal convolucional es reconocer que emoción pertenece, También se utiliza un segundo set de datos con imágenes de personajes de “Los Simpson” para la tarea de reconocimiento de individuos, los resultados obtenidos se muestran en el capítulo 4 mediante gráficas y tablas.

Capítulo 4
Experimentos y resultados

En este capítulo se muestran las pruebas y resultados obtenidos con la variación del número de capas aplicadas a la red neuronal convolucional, esta misma Red Neuronal Convolucional se utiliza para realizar dos tareas, detectar emociones mediante imágenes de expresiones faciales y reconocimiento de individuos mediante imágenes de personajes animados.

4.1 Detección de expresiones faciales

Esta sección muestra los resultados obtenidos con un dataset de expresiones faciales para determinar la emoción que representa cada imagen de entrada. Para estos experimentos se consideraron 6 emociones distintas: enojo, tristeza, felicidad, sorpresa, miedo y neutral. Para el entrenamiento se cuenta con 3001 imágenes de cada emoción, para las pruebas se tienen 1001 imágenes por cada emoción. Este set de datos contiene imágenes de personas con diferentes características por ejemplo, algunas imágenes son de mujeres, hombres, niños, niñas, bebes, dibujos animados, además de que presentan diferentes nacionalidades, expresando una emoción, las imágenes se pueden descargar de la página [10].

4.1.1 Red Neuronal Convolucional con 1 capa de convolución

En esta sección se muestran los resultados obtenidos con 1 capa de convolución variando el número de capas completamente conectadas.

La configuración del modelo con 1 capa de convolución y 1 capa completamente conectada consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de Keras donde se muestra esta implementación se encuentra en el Anexo A

La parte que genera 1 capa de convolución y 1 capa completamente conectada se muestra en la siguiente fracción de código.

```
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

Los resultados obtenidos de esta configuración se resumen en la imagen 4.1 que es una gráfica donde se puede observar que el mejor resultado es de 0.4667 que es después de la primera iteración de entrenamiento, si se continúan con las iteraciones la precisión de la red con datos que no conoce decrementa.

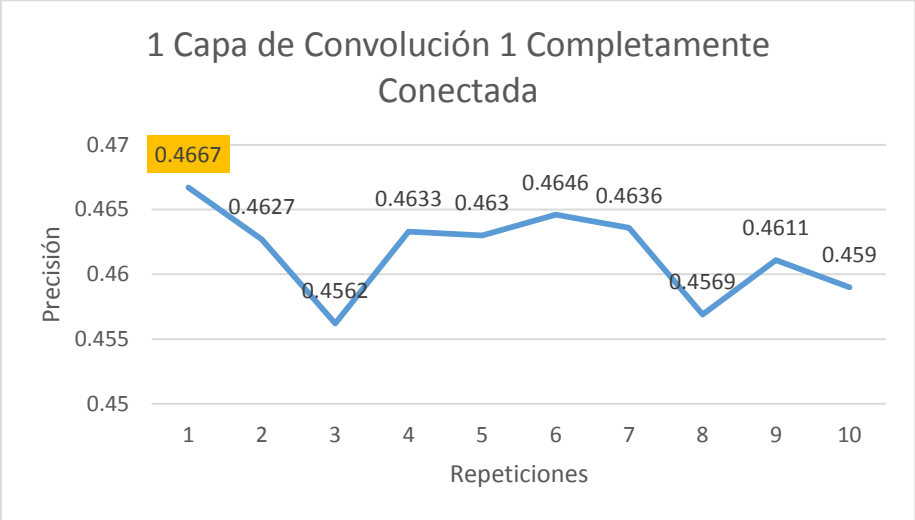


Imagen 4. 1 Gráfica de 1 capa de convolución 1 capa completamente conectada

La configuración del modelo con 1 capa de convolucion y 2 capas completamente conectadas consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas

- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

Para agregar una segunda capa se utiliza el mismo código del anexo A, simplemente se agrega capa completamente conectada de 64 neuronas como se muestra en la siguiente porción de código.

```
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 64, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

La gráfica de la imagen 4.2 describe los resultados obtenidos después de ejecutar el modelo con las configuraciones mencionadas, en esta gráfica se puede observar que la mejor precisión se obtiene en la primera iteración.

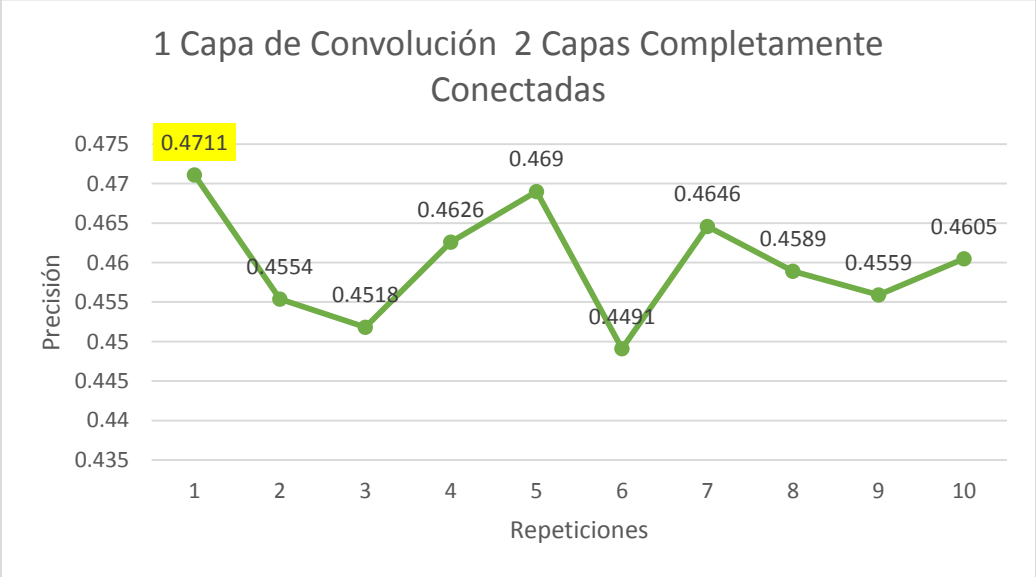


Imagen 4. 2 Gráfica de 1 capa convolucional y 2 capas completamente conectadas

La configuración del modelo con 1 capa de convolucion y 3 capas completamente conectadas consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 256 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código del anexo A se le agrega una red de 256 neuronas, otra de 128 y una de 64 además de la capa de salida que depende del número de categorías que se tiene, el siguiente fragmento de código muestra la implementación:

```
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 64, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

La imagen 4.3 contiene una gráfica muestra los resultados obtenidos con 1 capa convolucional y 3 capas completamente conectadas, se observa que la mejor presicion del modelo se tiene en la iteración 7 donde alcanza un valor de 0.489.

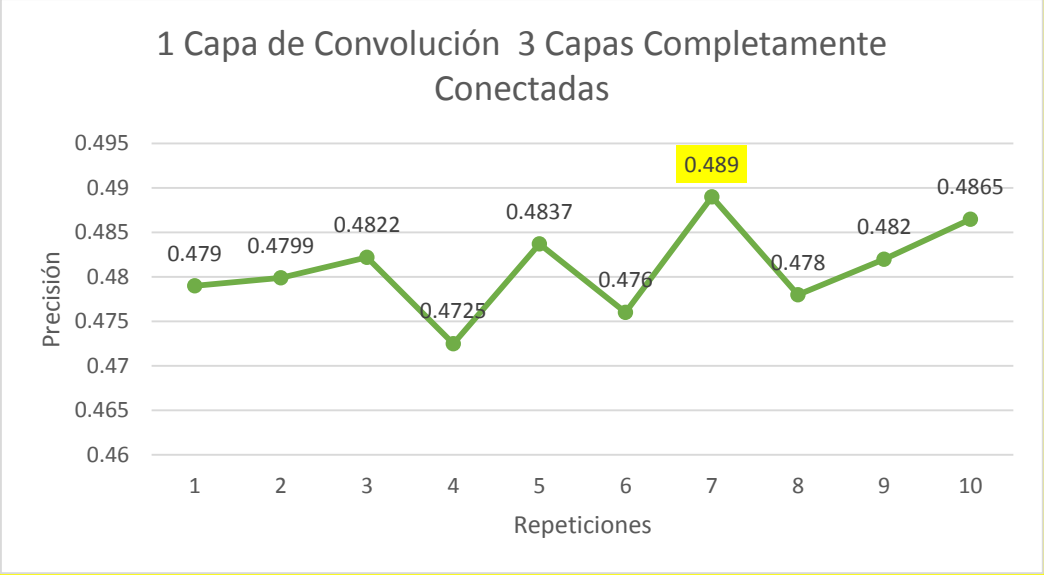


Imagen 4. 3 Gráfica 1 capa de convolución 3 completamente conectadas

En base a los resultados obtenidos con 1 capa convolucional se puede mostrar la siguiente tabla que contiene los datos de cada una de las ejecuciones, con las configuraciones mencionadas resaltando los mejores resultados de color amarillo.

Tabla 4. 1 Resultados del modelo con 1 capa convolucional

General		Imagenes		Capa Convolucional			Fully conected					Salida			
Epoch	Clases	Training	Test	NoFiltros	T. Filtros	Funcion	Capa1	Capa2	Capa3	Salida	F. Activación	Optimizador	Loss	Accuracy	Val_Accuracy
1	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	1.2478	0.5115	0.4667
2	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.9023	0.6615	0.4627
3	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.7148	0.7371	0.4562
4	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.5943	0.7835	0.4633
5	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.5101	0.8157	0.463
6	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.4485	0.8393	0.4646
7	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.3995	0.8576	0.4636
8	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.365	0.8709	0.4569
9	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.3339	0.8818	0.4611
10	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.3102	0.891	0.459
1	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	1.2545	0.5069	0.4711
2	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.9446	0.6431	0.4554
3	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.7756	0.7124	0.4518
4	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.6618	0.7569	0.4626
5	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.581	0.7886	0.469
6	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.5213	0.8113	0.4491
7	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.4722	0.8294	0.4646
8	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.4352	0.8496	0.4589
9	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.4044	0.8554	0.4559
10	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.3793	0.8647	0.4605
1	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	1.1881	0.5353	0.479
2	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.7871	0.7079	0.4799
3	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.5881	0.787	0.4822
4	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.4665	0.8339	0.4725
5	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.3827	0.8651	0.4837
6	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.3266	0.8859	0.476
7	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.2843	0.9019	0.489
8	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.2537	0.9131	0.478
9	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.2286	0.922	0.482
10	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.2066	0.933	0.4865

Enseguida en la imagen 4.4 se muestra una gráfica de la tabla 4.1 con los resultados obtenidos, en ella se puede observar el impacto que tiene el agregar una capa completamente conectada en el resultado de una red convolucional, como se puede observar una tercera capa, mejora considerablemente la precisión.

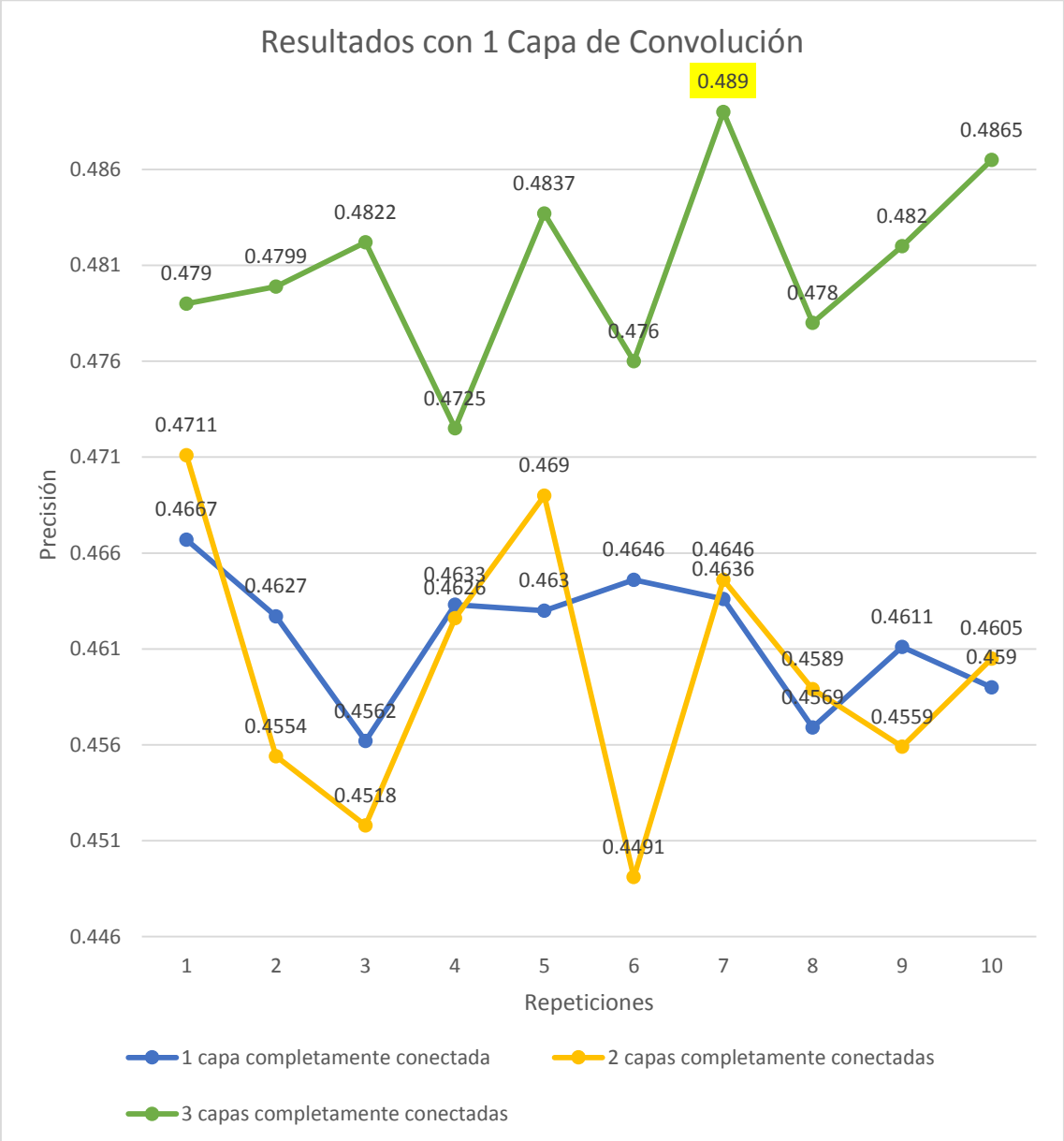


Imagen 4. 4 Resultados 1 capa convolucional

4.1.2 Red Neuronal Convolucional con 2 capas de convolución

En esta sección se muestran los resultados de una Red Neuronal Convolucional con 2 capas de convolucion y variaciones en las capas completamente conectadas.

La configuración del modelo con 2 capas de convolucion y 1 capa completamente conectada consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de esta red neuronal se encuentra en el anexo B, la parte donde se encuentra la implementación de la segunda capa convolucional y la capa completamente conectada se muestra en el siguiente código.

```
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(64, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

La capa convolucional implementada genera la gráfica que se muestra en la imagen 4.5 la mejor precisión mostrada está en la primer iteración alcanzando una precisión de 0.5073.

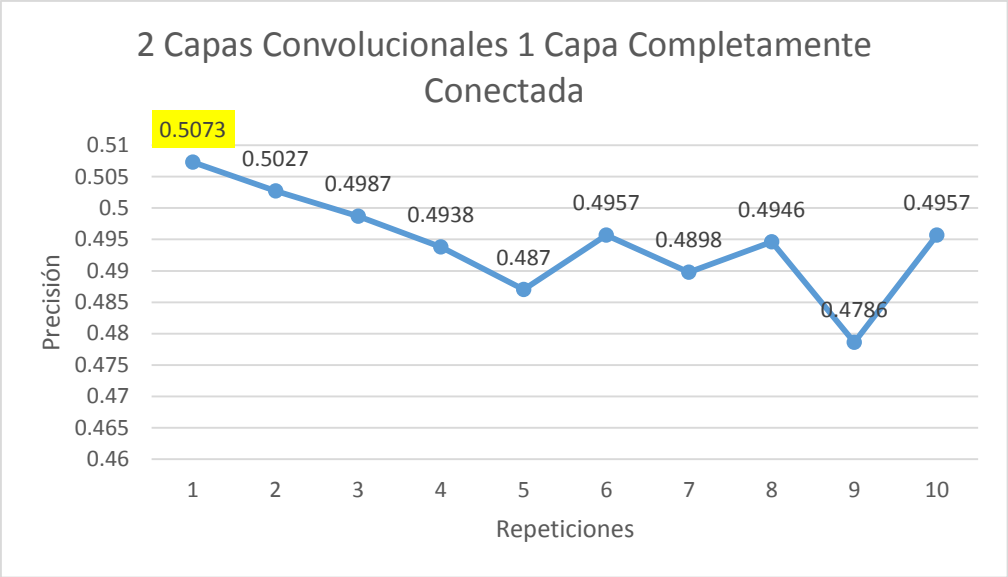


Imagen 4. 5 Gráfica de 2 capas convolucionales 1 capa completamente conectada

La configuración de la red convolucional de 2 capas de convolucion y 2 capa completamente conectada consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

La siguiente fracción de código muestra la forma de implementar la configuración especificada en Keras haciendo la modificacion en el Anexo B.

```
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 64, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

En la imagen 4.6 se muestra la gráfica de los resultados obtenidos con esta configuración resaltando de color amarillo el mejor resultado encontrado en la quinta iteración.

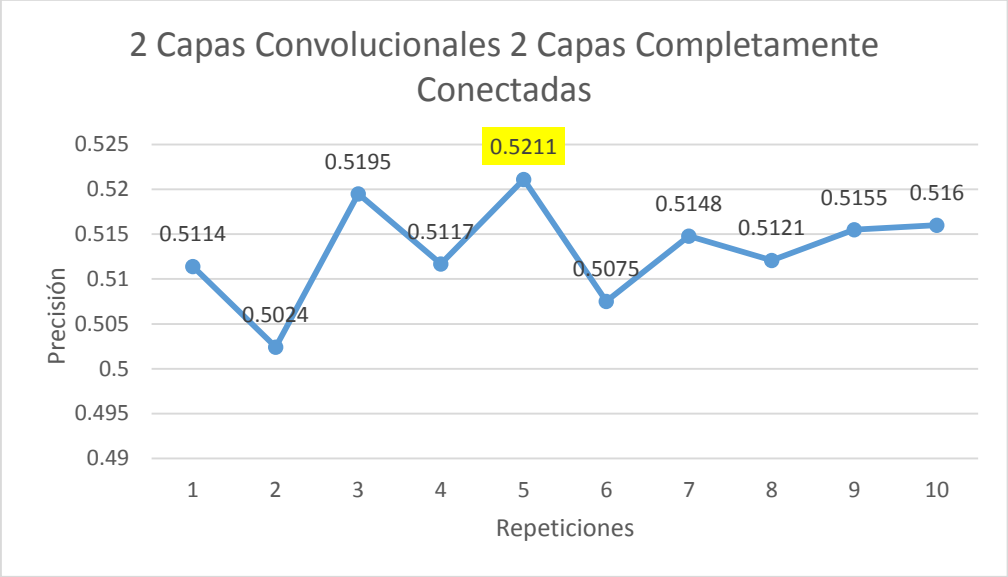


Imagen 4. 6 Gráfica con 2 capas de convolución 2 capas completamente conectadas

La configuración del modelo con 2 capas de convolucion y 3 capas completamente conectadas consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 256 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.

- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

La siguiente porción de código muestra lo que se debe agregar al código del Anexo B para poder tener la configuración que se especificó en el párrafo anterior.

```
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 64, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

Esta configuración genera los resultados que se muestran en la gráfica de la imagen 4.7, donde la mejor precisión se tiene en la iteración 8 con un valor de 0.5292.

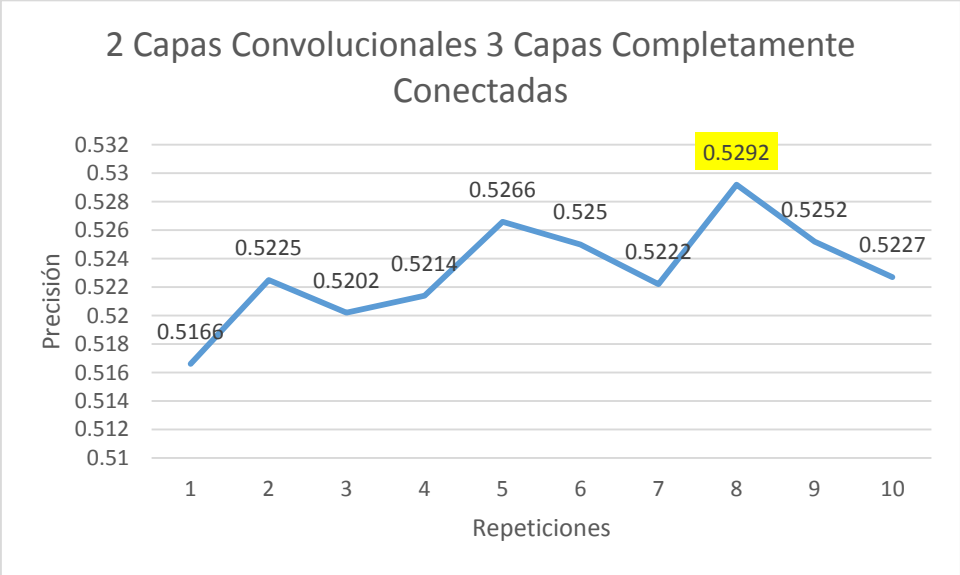


Imagen 4. 7 Gráfica de 2 capas convolucionales y 3 capas completamente conectadas

La siguiente tabla de resultados 4.2 muestra los valores a los que llega la precisión con las diferentes configuraciones de capas completamente conectadas de la red neuronal de 2 capas convolucionales, se resalta de color amarillo el valor de precisión más alto.

Tabla 4. 2 Resultados de la red neuronal con 2 capas convolucionales

General		Imagenes		Capa Convolucional					Fully conected					Salida			
Epoch	Clases	Training	Test	NoFiltros	T. Filtros	NoFiltros	T. Filtros	Funcion	Capa1	Capa2	Capa3	Salida	F. Activación	Optimizador	Loss	Accuracy	Val_Accuracy
1	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	1.0811	0.5823	0.5073
2	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.6492	0.7624	0.5027
3	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.4863	0.8242	0.4987
4	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.4014	0.8559	0.4938
5	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.3458	0.8771	0.487
6	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.3111	0.8898	0.4957
7	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.2817	0.9005	0.4898
8	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.2616	0.9077	0.4946
9	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.2431	0.9149	0.4786
10	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.2293	0.9198	0.4957
1	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	1.117	0.5678	0.5114
2	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.7226	0.7316	0.5024
3	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.5426	0.8019	0.5195
4	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.4379	0.8419	0.5117
5	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.3668	0.8696	0.5211
6	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.3184	0.8875	0.5075
7	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.2803	0.9016	0.5148
8	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.2562	0.9102	0.5121
9	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.235	0.9178	0.5155
10	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.2177	0.9243	0.516
1	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.9851	0.6212	0.5166
2	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.4937	0.8225	0.5225
3	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.3249	0.8861	0.5202
4	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.2425	0.916	0.5214
5	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.1962	0.933	0.5266
6	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.167	0.9434	0.525
7	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.1465	0.9509	0.5222
8	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.13	0.9564	0.5292
9	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.1186	0.9607	0.5252
10	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.1098	0.9636	0.5227

Para hacer más fácil la comparación de los cambios que genera agregar o quitar capas completamente conectadas a la red convolucional se muestra la imagen 4.8. Que está basado en la tabla anterior 4.2. Dicha gráfica muestra una comparación de todas las capas completamente conectadas agregadas a la red neuronal convolucional de 2 capas, donde la mejor precisión se obtiene en la iteración número 8 con 3 capas completamente conectadas.

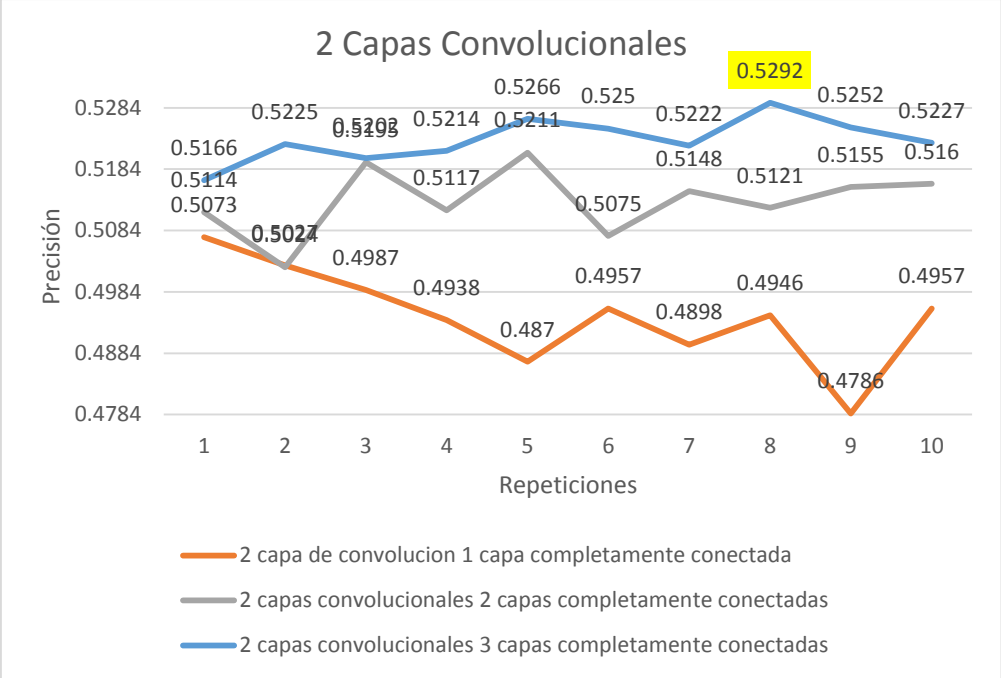


Imagen 4. 8 Gráfica de 2 capas convolucionales

4.1.3 Red Neuronal Convolucional con 3 capas de convolución

La configuración de la red convolucional de 3 capas de convolución y 1 capa completamente conectada consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código para la implementación de la red neuronal convolucional de 3 capas se muestra en el anexo C, y las instrucciones que agregan la capa completamente conectada se muestran en el siguiente fragmento de código.

```
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(64, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(128, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

Al ejecutar el modelo con las especificaciones, genera la información que se muestra en la gráfica de la imagen 4.9, donde se puede apreciar que en las primeras 2 iteraciones la red neuronal convolucional mejora la precisión, pero en iteraciones posteriores empieza a decrementar.

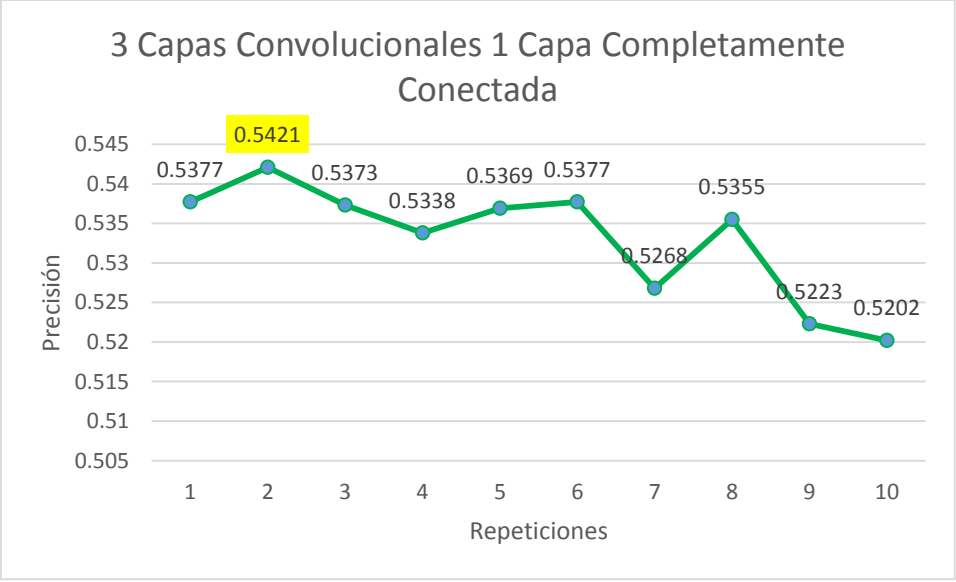


Imagen 4. 9 Gráfica 3 capas convolucionales 1 completamente conectadas

La configuración de la red convolucional de 3 capas de convolucion y 2 capa completamente conectada consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código del anexo C, es modificado para implementar las especificaciones de esta red neuronal convolucional, simplemente agregando las siguientes líneas de código

```
classifier.add(Dense(units = 128, activation = 'relu'))

classifier.add(Dense(units = 64, activation = 'relu'))

classifier.add(Dense(units = categorias, activation = 'softmax'))
```

Después de ejecutar el programa, genera los resultados que se pueden ver en la imagen 4.10 la cual muestra que inicia mejorando la precisión de la red neuronal, sin embargo al llegar a la iteración 6 donde se tiene el mejor resultado, éste empieza a decrementar.

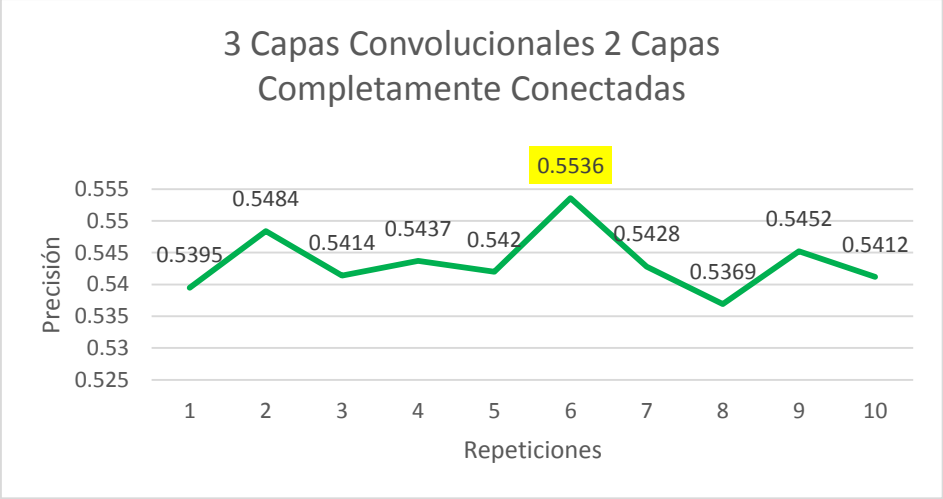


Imagen 4. 10 Gráfica de 3 capas convolucionales y 2 completamente conectadas

La configuración de la red convolucional con 3 capas de convolucion y 3 capas completamente conectadas consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 128 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 256 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

Para obtener la configuración mencionada, al código del anexo C se le inserta la siguiente porción de código.

```
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 64, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
```

La imagen 4.11 muestra los resultados del corrimiento del código mostrado en el anexo C, con la modificación que se muestra en el código anterior.

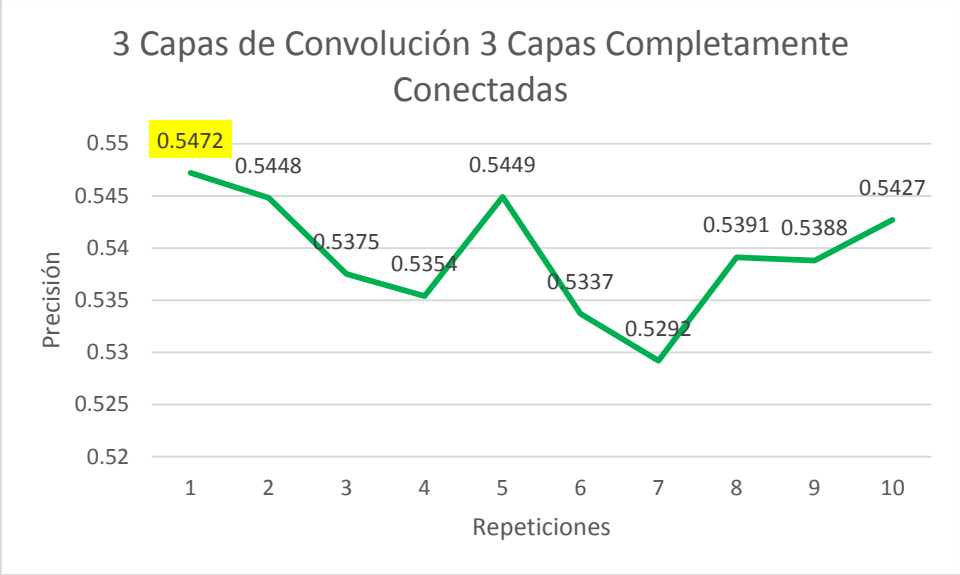


Imagen 4. 11 Gráficas con 3 capas de convolucion y 3 capas completamente conectadas

La tabla 4.3 muestra todos los valores que se obtuvieron después de ejecutar la Red Neuronal Convolucional variando el número de capas completamente conectadas mostrando también las configuraciones mencionadas. Se puede observar que la red convolucional con 2 capas completamente conectadas genera un mejor resultado en la sexta iteración.

Tabla 4. 3 Resultados con 3 capas convolucionales

General		Imagenes		Capa Convolucional							Fully conected					Salida			
Epoch	Clases	Training	Test	NoFiltros	T. Filtros	NoFiltros	T. Filtros	NoFiltros	T. Filtros	Funcion	Capa1	Capa2	Capa3	Salida	F. Activación	Optimizador	Loss	Accuracy	Val_Accuracy
1	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	1.12	0.5631	0.5377
2	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.8049	0.6956	0.5421
3	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.6603	0.7521	0.5373
4	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.566	0.7903	0.5338
5	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.4968	0.8171	0.5369
6	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.4542	0.834	0.5377
7	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.4198	0.8471	0.5268
8	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.3943	0.859	0.5355
9	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.374	0.8638	0.5223
10	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.3573	0.8705	0.5202
1	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	1.055	0.591	0.5395
2	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.6977	0.7372	0.5484
3	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.5516	0.7953	0.5414
4	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.4652	0.8286	0.5437
5	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.4089	0.8509	0.542
6	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.3669	0.8672	0.5536
7	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.3342	0.8799	0.5428
8	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.3088	0.8896	0.5369
9	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.2878	0.8978	0.5452
10	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.2739	0.9032	0.5412
1	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	1.0597	0.5852	0.5472
2	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.6451	0.7592	0.5448
3	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.4692	0.829	0.5375
4	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.3737	0.8654	0.5354
5	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.3166	0.8879	0.5449
6	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.277	0.9096	0.5337
7	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.2483	0.9237	0.5292
8	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.2259	0.9225	0.5391
9	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.2082	0.929	0.5388
10	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.1924	0.9347	0.5427

De la tabla anterior se genera la gráfica de la imagen 4.12, donde se realiza una comparación de los resultados obtenidos y se puede observar que una segunda capa completamente conectada proporciono mejores resultados que incluso teniendo 3 capas completamente conectadas.

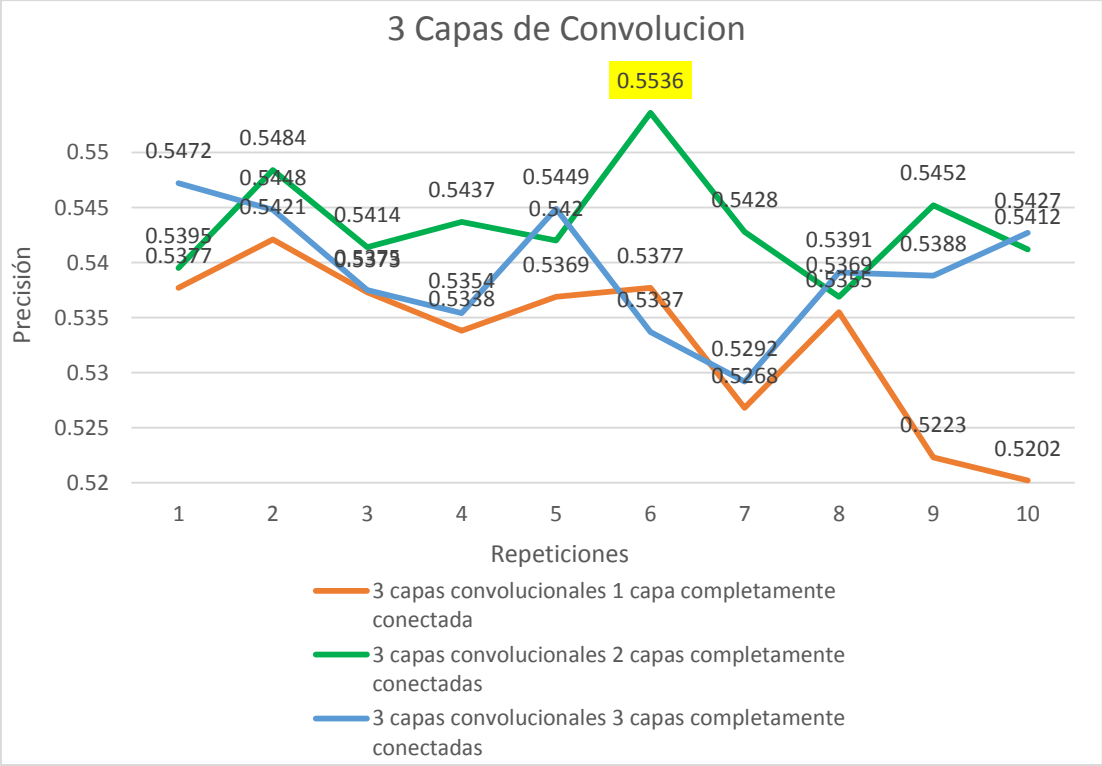


Imagen 4. 12 Gráfica de la red neuronal con 3 capas de convolución

4.1.4 Comparación de precisión para expresiones faciales

En esta sección se presenta una gráfica que concentra un resumen de todos los resultados obtenidos durante las diferentes pruebas realizadas con un conjunto de datos de rostros con expresiones faciales que demuestran emociones en personas.

La imagen 4.13 contiene una comparación de las gráficas obtenidas de todas las pruebas realizadas. Si bien como se puede observar los resultados con mejor precisión los proporciona la red convoluciónal de 3 capas y 3 capas completamente conectadas, esa configuración la precisión de la red llega a un máximo de 0.5536.

En su Gran parte de la precisión a la que la red puede tener depende del conjunto de datos de entrenamiento, como se menciona en la sección 4.1 anterior, este set de datos es muy extenso ya que toma muchas consideraciones y para una mejor precisión es necesario más entrenamiento y a agregar más capas convoluciónales sin embargo, para eso es necesario equipos más potentes como GPUs debido a que con una CPU este proceso puede tardar demasiado. Eso quiere decir que la red tiene una ma la precisión, ya que se está tratando de identificar 6 emociones, 0.5536 que está por arriba de un sexto.

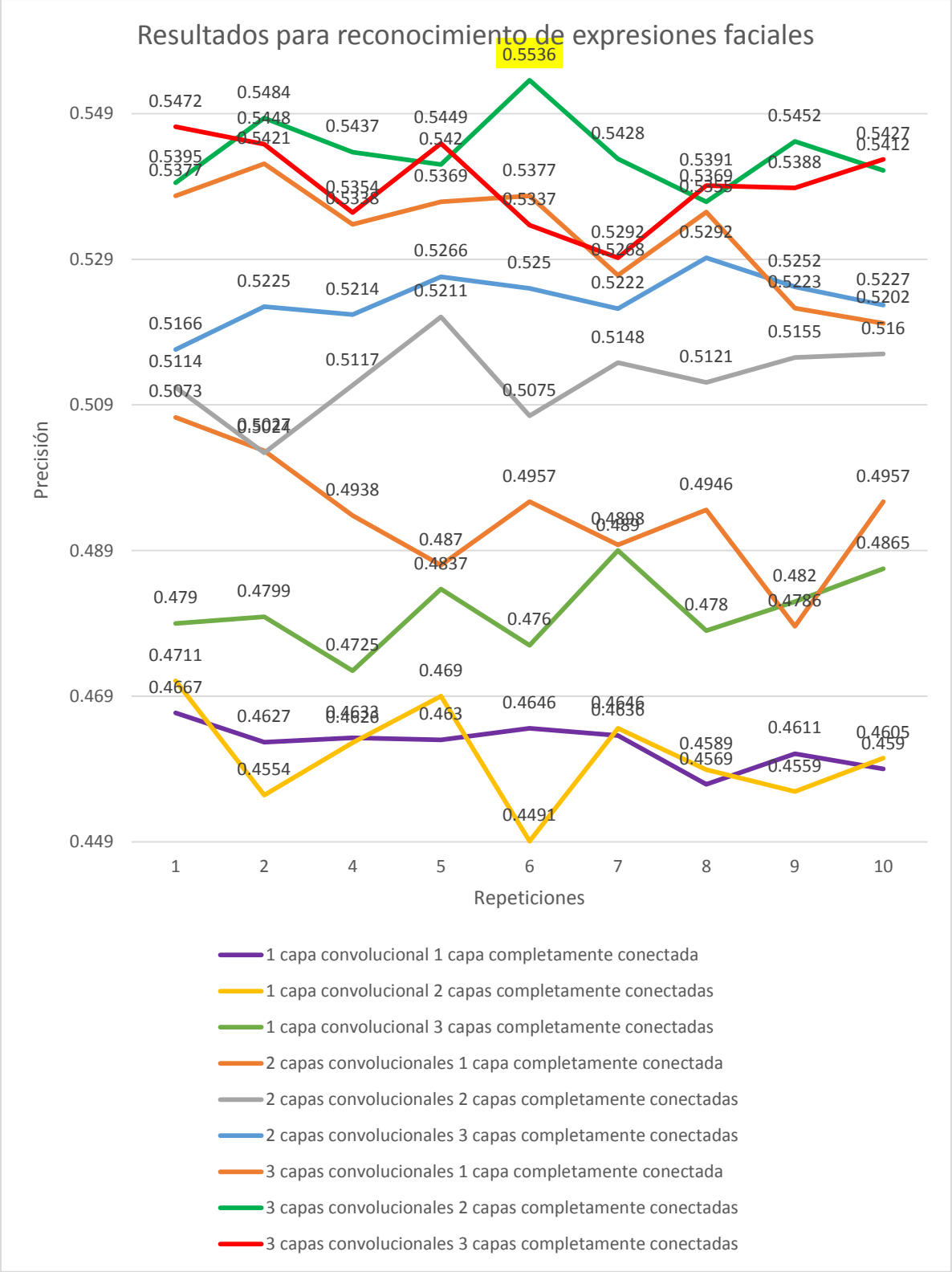


Imagen 4. 13 Gráfica de resultados para la tarea de reconocimiento de emociones

4.2 Reconocimiento de individuos

Esta sección muestra los resultados obtenidos para la tarea de reconocimiento de individuos. Se utilizó un dataset de personajes animados de la serie "Los Simpsons" con seis individuos distintos en diferentes posiciones y con distintas expresiones faciales cada uno. El dataset cuenta 791 imágenes por cada clase para el entrenamiento, y con 500 imágenes por clase para pruebas y se puede descargar de la pagina [11].

4.2.1 Red Neuronal Convolucional con 1 capa de convolución

En este apartado se muestran los resultados obtenidos con 1 capa de convolución variando el número de capas completamente conectadas.

La configuración del modelo es:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de Keras es el mismo de la sección 4.1.1 que se encuentra en el Anexo A. Los resultados obtenidos de esta configuración se resumen en la imagen 4.14 que es una gráfica donde se puede observar que el mejor resultado es de 0.731.

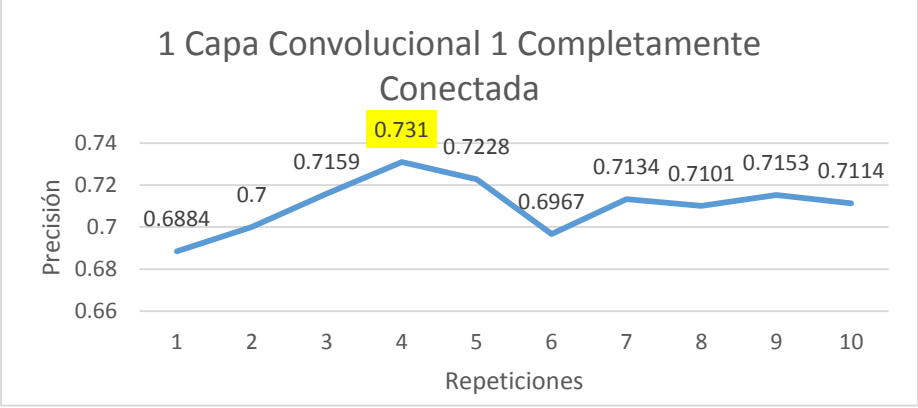


Imagen 4. 14 Gráfica con 1 capa de convolución 1 capa completamente conectada

La configuración del modelo con 1 capa de convolucion y 2 capas completamente conectadas consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

Para agregar una segunda capa se utiliza el código de la sección 4.1.2.

La gráfica en la imagen 4.15 describe los resultados obtenidos después de ejecutar el modelo con las configuraciones mencionadas. En dicha gráfica se puede observar que la mejor precisión se obtiene en la novena iteración.

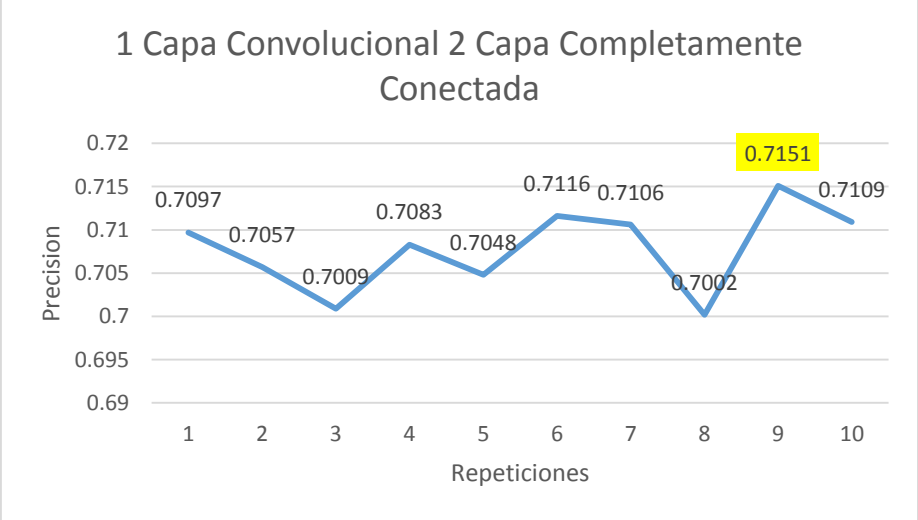


Imagen 4. 15 Gráfica de 1 capa convolucional y 2 capas completamente conectadas

La configuración del modelo con 1 capa de convolución y 3 capas completamente conectadas consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles

- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 256 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de Keras es el mismo que la sección 4.1.2

La siguiente imagen 4.16 muestra una gráfica con los resultados obtenidos con 1 capa convolucional y 3 capas completamente conectadas, donde el mejor valor de precisión está en la iteración 9.

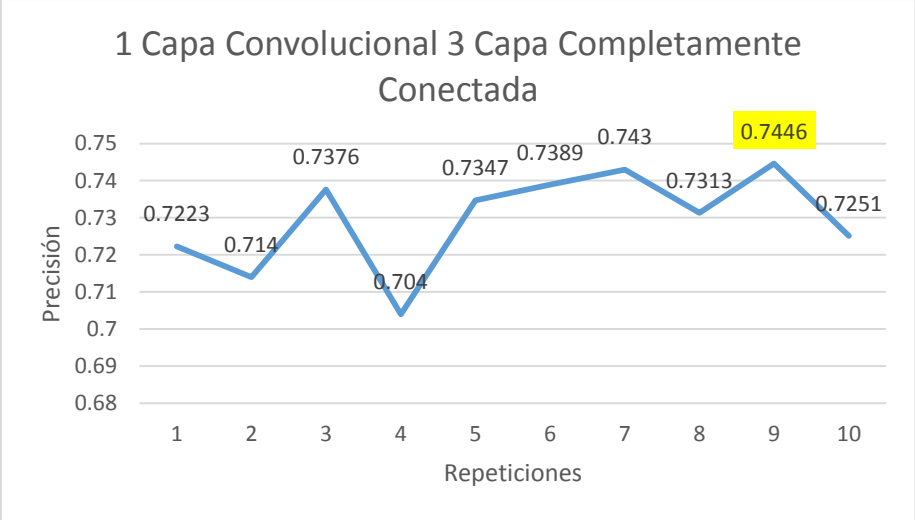


Imagen 4. 16 Gráfica con 1 capa de Convolucion 3 completamente conectadas

Con los resultados obtenidos del modelo con 1 capa convolucional se puede mostrarla siguiente tabla 4.4 que contiene los datos de los corrimientos por cada iteración.

Tabla 4. 4 Resultados con 1 capa Convolucional

General		Imagenes		Capa Convolucional			Fully conected					Salida			
Epoch	Clases	Training	Test	NoFiltros	T. Filtros	Funcion	Capa1	Capa2	Capa3	Salida	F. Activación	Optimizador	Loss	Accuracy	Val_Accuracy
1	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.5583	0.8054	0.6884
2	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.1512	0.9505	0.7
3	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.085	0.9737	0.7159
4	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.0574	0.982	0.731
5	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.046	0.9859	0.7228
6	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.0385	0.9886	0.6967
7	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.036	0.9896	0.7134
8	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.0303	0.9912	0.7101
9	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.0294	0.9917	0.7153
10	6	18006	6006	32	3x3	Relu	128			6	softmax	adam	0.0251	0.9931	0.7114
1	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.4701	0.8319	0.7097
2	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.1119	0.9624	0.7057
3	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.067	0.9782	0.7009
4	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.0497	0.9839	0.7083
5	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.0405	0.9876	0.7048
6	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.0373	0.9886	0.7116
7	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.0317	0.9902	0.7106
8	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.0286	0.9914	0.7002
9	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.025	0.9926	0.7151
10	6	18006	6006	32	3x3	Relu	128	64		6	softmax	adam	0.0248	0.993	0.7109
1	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.3973	0.8578	0.7223
2	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0765	0.9751	0.714
3	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0463	0.9853	0.7376
4	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0351	0.9889	0.704
5	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0293	0.9912	0.7347
6	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.025	0.9927	0.7389
7	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0206	0.9937	0.743
8	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.017	0.9953	0.7313
9	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0162	0.9953	0.7446
10	6	18006	6006	32	3x3	Relu	256	128	64	6	softmax	adam	0.0151	0.9957	0.7251

En la imagen 4.17 se muestra una gráfica con los resultados obtenidos, y se puede notar la mejoría en la precisión al incorporar más capas completamente conectadas a la red.

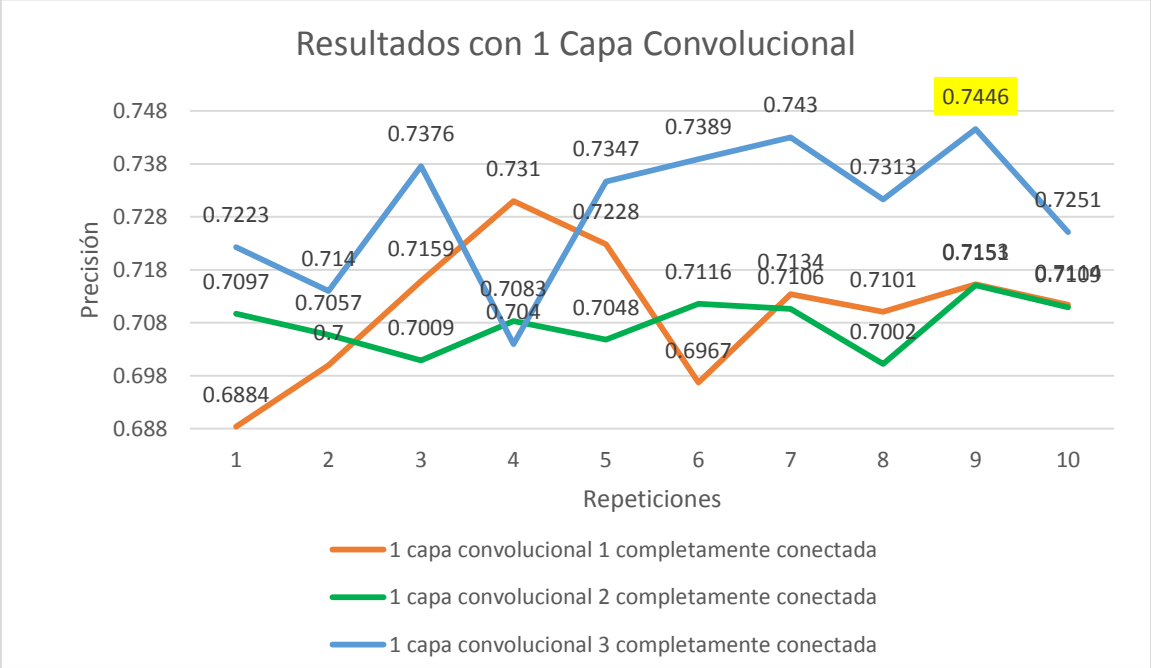


Imagen 4. 17 Resultados 1 capa convolucional

4.2.2 Red Neuronal Convolucional con 2 capas de convolución

En esta sección se muestran los resultados una red neuronal con 2 capas convolucionales, con variaciones en las capas completamente conectadas.

La configuración de la red con 2 capas de convolución y 1 capa completamente conectada consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLuU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de esta red neuronal se encuentra en el anexo B

La capa convolucional implementada genera la gráfica que se muestra a continuación en la imagen 4.18, la mejor precisión al que se llega es al .8277.

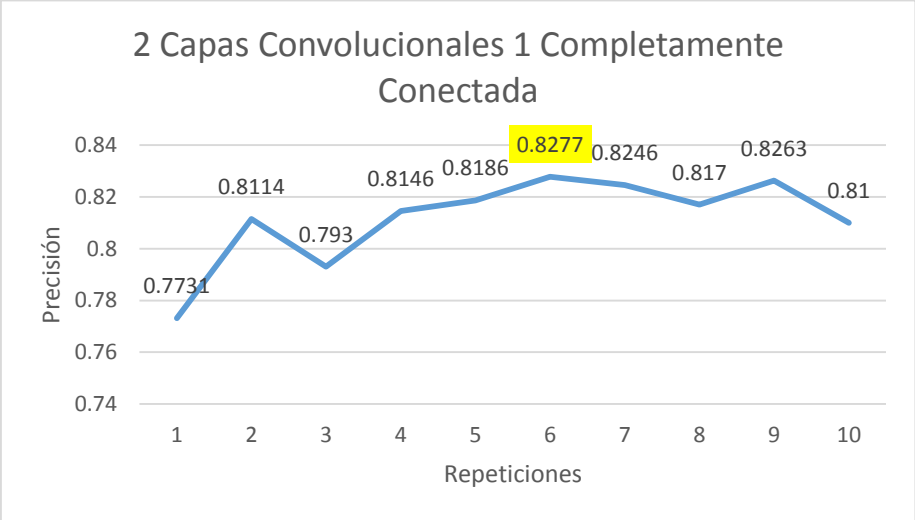


Imagen 4. 18 Gráfica de 2 capas convolucionales 1 capa completamente conectada

La configuración de la red convolucional de 2 capas de convolucion y 2 capa completamente conectada consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código que muestra la forma de implementar la configuración especificada en Keras es la de la sección 4.1.2.

La imagen 4.19 es una gráfica muestra las mejoras que proporciona crear otra capa completamente conectada a la red convolucional.

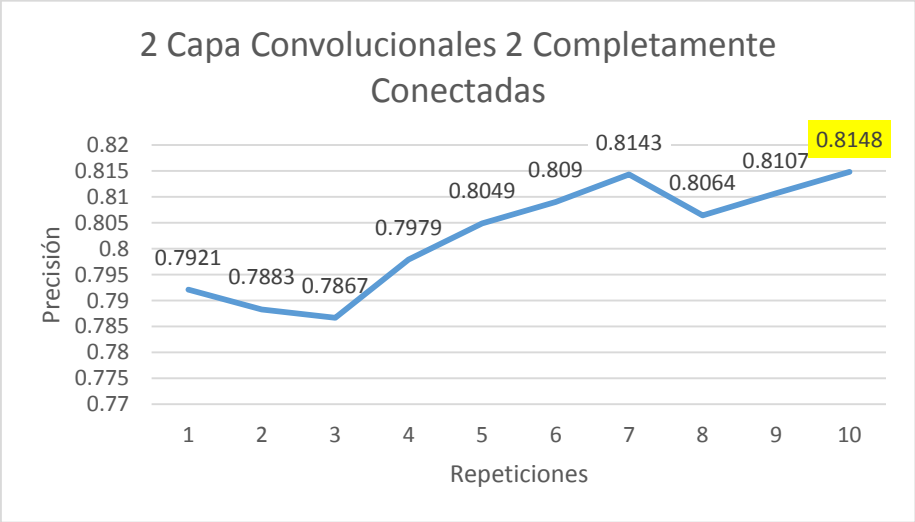


Imagen 4. 19 Gráfica con 2 capas de convolución y 2 completamente conectadas

La configuración de la red convolucional con 2 capas de convolucion y 3 capas completamente conectadas consta de:

- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 256 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de la sección 4.1.2 con 3 capas completamente conectadas, es el que se utiliza para esta sección y los resultados obtenidos con el set de datos de los Simpson se pueden observar en la imagen 4.20.

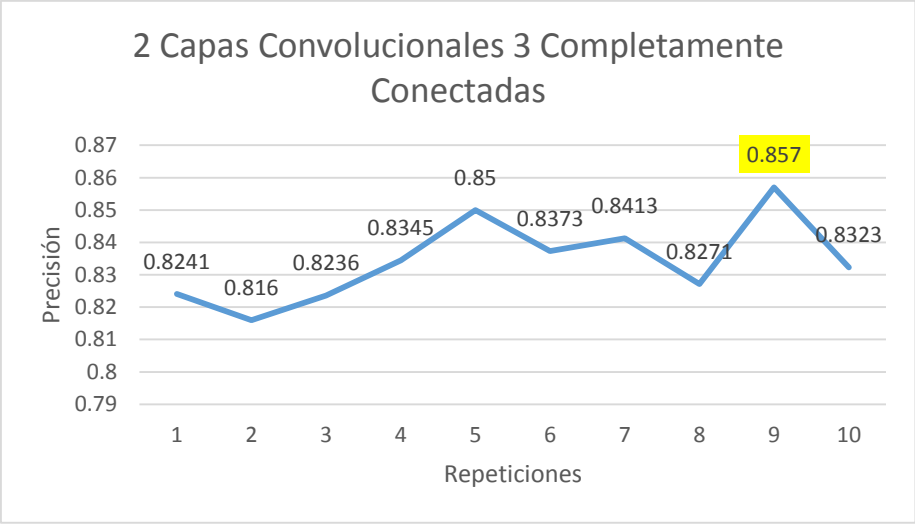


Imagen 4. 20 Gráfica con 2 capas convolucionales y 3 completamente conectadas

La siguiente tabla 4.5 muestra todos los resultados obtenidos de la Red Neuronal con 2 capas convolucionales resaltando de color amarillo la mejor precisión en cada ejecución.

Tabla 4. 5 Resultados de la red neuronal con 2 capas convolucionales

General		Imagenes		Capa Convolucional					Fully conected					Salida			
Epoch	Clases	Training	Test	NoFiltros	T. Filtros	NoFiltros	T. Filtros	Funcion	Capa1	Capa2	Capa3	Salida	F. Activación	Optimizador	Loss	Accuracy	Val_Accuracy
1	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.3366	0.8808	0.7731
2	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0502	0.9845	0.8114
3	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0309	0.9905	0.793
4	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.023	0.9931	0.8146
5	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0195	0.9943	0.8186
6	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0184	0.995	0.8277
7	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0158	0.9958	0.8246
8	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0127	0.9966	0.817
9	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0147	0.9964	0.8263
10	6	18006	6006	32	3x3	64	3x3	Relu	128			6	softmax	adam	0.0136	0.9968	0.81
1	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.3658	0.8689	0.7921
2	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0536	0.9823	0.7883
3	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0345	0.989	0.7867
4	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0267	0.9917	0.7979
5	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0217	0.9934	0.8049
6	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0193	0.9943	0.809
7	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0172	0.995	0.8143
8	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0151	0.9957	0.8064
9	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0136	0.9961	0.8107
10	6	18006	6006	32	3x3	64	3x3	Relu	128	64		6	softmax	adam	0.0138	0.9964	0.8148
1	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.288	0.8965	0.8241
2	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0397	0.9871	0.816
3	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0247	0.9923	0.8236
4	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0189	0.9943	0.8345
5	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0164	0.9951	0.85
6	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0141	0.9961	0.8373
7	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0129	0.9966	0.8413
8	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.01	0.9973	0.8271
9	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0114	0.9973	0.857
10	6	18006	6006	32	3x3	64	3x3	Relu	256	128	64	6	softmax	adam	0.0112	0.9971	0.8323

En la imagen 4.21 se tienen 3 gráficas que muestran una comparación de los resultados obtenidos de agregar capas completamente conectadas a la red de 2 capas convolucionales mostrando que la mejor precisión con es con 3 capas completamente conectadas.

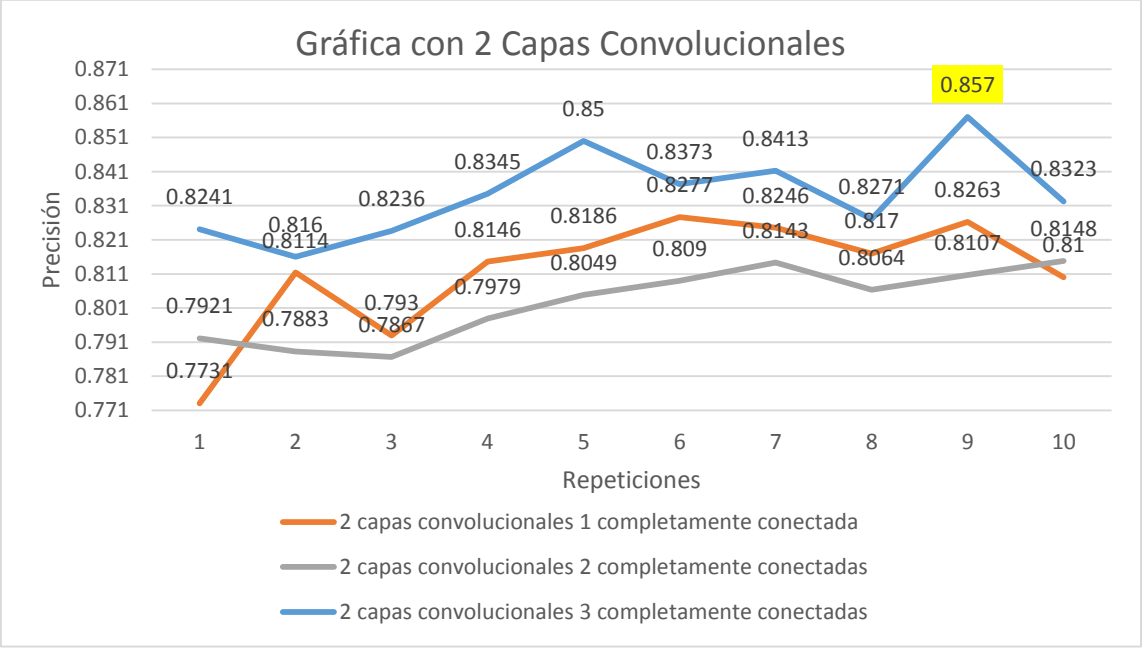


Imagen 4. 21 Gráfica de 2 capas convolucionales

4.2.3 Red Neuronal Convolucional con 3 capas de convolución

En este apartado se muestran los resultados obtenidos agregar capas completamente conectadas a una red convolucional de 3 capas.

La configuración de la red convolucional de 3 capas de convolucion y 1 capa completamente conectada consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 128 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de Keras para la implementación de la red neuronal convolucional de 3 capas se muestra en el anexo C.

Al ejecutar el modelo, se tienen los resultados de la imagen 4.22 donde se puede observar que la precisión llega a 0.8877

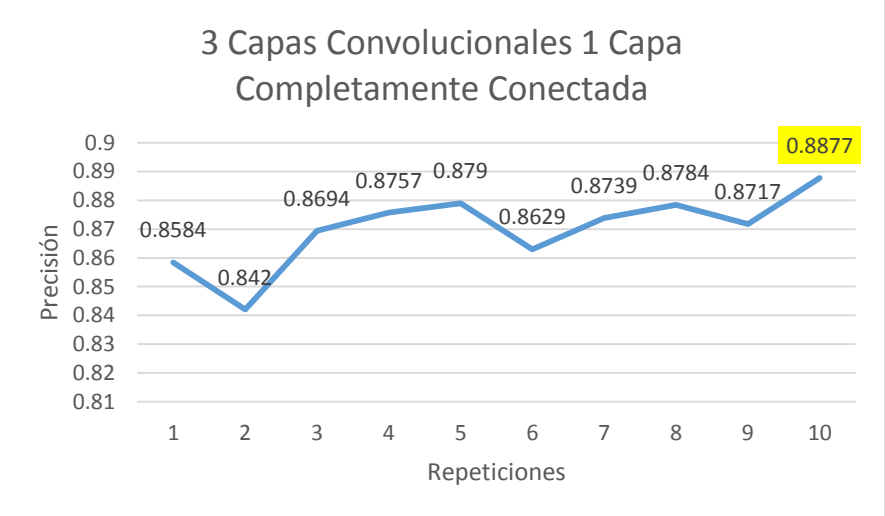


Imagen 4. 22 Gráfica de 3 capas convolucionales 1 completamente conectada

La configuración de la red convolucional de 3 capas de convolucion y 2 capa completamente conectada consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 128 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de la sección 4.1.3 con 3 capas de convolucion es el mismo que se utiliza en esta sección lo que cambia es el data set, y después de ejecutar el programa se tienen los resultados de la imagen 4.23.

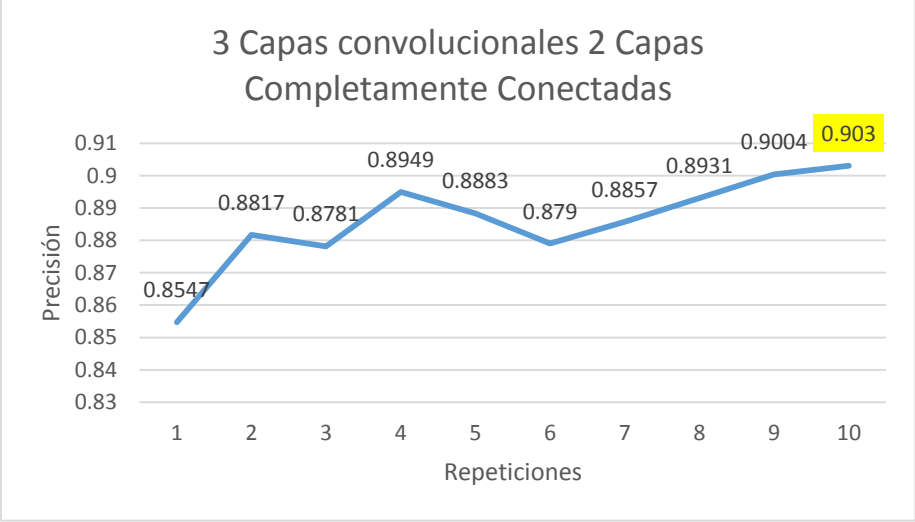


Imagen 4. 23 Gráfica de 3 capas convolucionales y 2 completamente conectadas

La configuración de la red convolucional con 3 capas de convolucion y 3 capas completamente conectadas consta de:

- Una capa convolucional de 32 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 64 filtros de 3x3 pixeles
- Activación ReLU
- Una capa convolucional de 128 filtros de 3x3 pixeles
- Activación ReLU
- Maxpooling de tamaño 2x2 pixeles
- Capa Completamente conectada de 256 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 128 neuronas internas
- Función de activación ReLU
- Capa Completamente conectada de 64 neuronas internas
- Función de activación ReLU
- Capa de salida de 6 neuronas.
- Función de activación Softmax
- Optimizador ADAM
- Repeticiones 10

El código de Keras con las especificaciones se encuentra en la sección 4.1.3, y los resultados obtenidos con el conjunto de datos de los Simpson se muestran en la imagen 4.24 donde se llega a un valor máximo de 0.9132

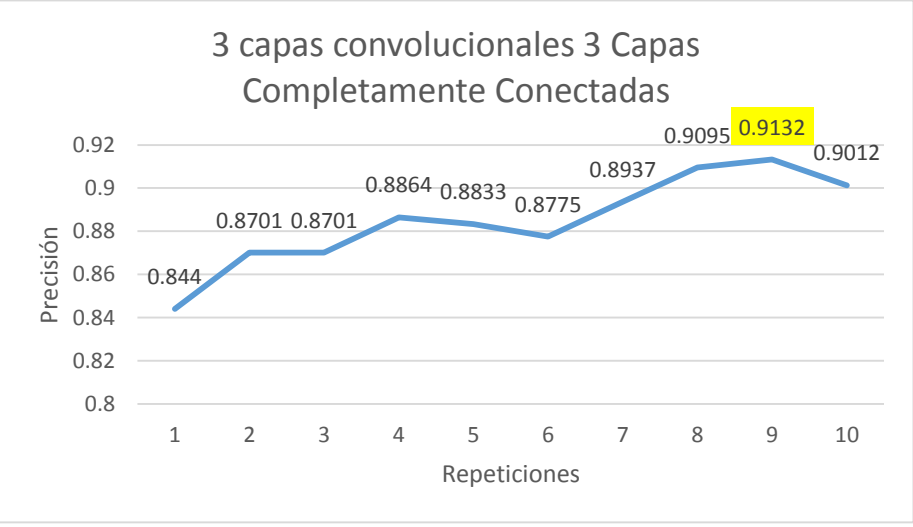


Imagen 4. 24 Gráficas con 3 capas de convolución y 3 capas completamente conectadas

La tabla 4.6 muestra todos los valores obtenidos para el modelo con 3 capas de convolución resaltando los mejores resultados en color amarillo.

Tabla 4. 6 Resultados del modelo con 3 capas de convolución

General		Imagenes		Capa Convolucional							Fully conected					Salida			
Epoch	Clases	Training	Test	NoFiltros	T. Filtros	NoFiltros	T. Filtros	NoFiltros	T. Filtros	Funcion	Capa1	Capa2	Capa3	Salida	F. Activación	Optimizador	Loss	Accuracy	Val_Accuracy
1	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.2711	0.9042	0.8584
2	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0399	0.9874	0.842
3	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0277	0.9917	0.8694
4	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0204	0.9943	0.8757
5	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0229	0.9941	0.879
6	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.021	0.9949	0.8629
7	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0196	0.9955	0.8739
8	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0187	0.996	0.8784
9	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0172	0.9965	0.8717
10	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128			6	softmax	adam	0.0235	0.9961	0.8877
1	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.3011	0.8909	0.8547
2	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.039	0.9874	0.8817
3	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0253	0.9923	0.8781
4	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0202	0.994	0.8949
5	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0179	0.9949	0.8883
6	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.015	0.9959	0.879
7	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0161	0.9959	0.8857
8	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0128	0.9967	0.8931
9	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0114	0.997	0.9004
10	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	128	64		6	softmax	adam	0.0126	0.9969	0.903
1	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.2662	0.9044	0.844
2	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0396	0.9878	0.8701
3	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0249	0.9927	0.8701
4	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0181	0.9949	0.8864
5	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0164	0.9954	0.8833
6	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0155	0.996	0.8775
7	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0141	0.9964	0.8937
8	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0125	0.9971	0.9095
9	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.0148	0.9968	0.9132
10	6	18006	6006	32	3x3	64	3x3	128	3x3	Relu	256	128	64	6	softmax	adam	0.012	0.9973	0.9012

De los datos de la tabla 7 se muestra la siguiente gráfica comparativa, conde se puede notar que con 3 capas completamente conectadas el valor de la precisión mejora.

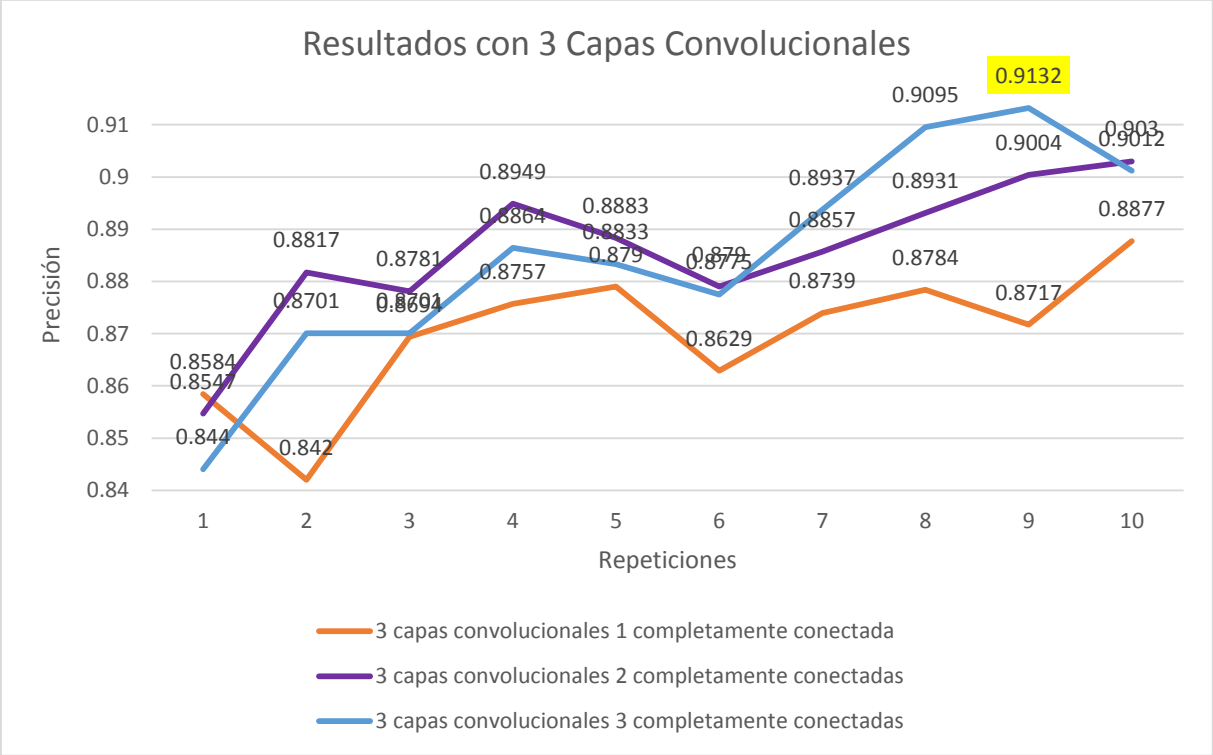


Imagen 4. 25 Gráfica de la red neuronal con 3 capas de convolución

4.2.4 Comparación de precisión para Reconocimiento de individuos

En este apartado se muestra una imagen con todas las gráficas de resultados obtenidos tras el corrimiento de las diferentes configuraciones de la red neuronal. La imagen 4.26 muestra las mejorías de la red con las configuraciones aplicadas, se puede ver como la CNN va mejorando al agregar más capas convoluciónales en combinación con capas completamente conectadas.

Es importante mencionar también que en la precisión de la red, influye el conjunto de datos de entrenamiento y la tarea que se realice ya que, como se mencionaba en el inicio de este capítulo, la red neuronal que se está usando es la misma pero el set de datos que se está utilizando para este capítulo es de personajes animados donde las características de cada uno de los personajes es muy marcado en comparación con los demás personajes de la serie. Es por ello que al momento de reconocer un personaje las características extraídas en el entrenamiento es suficiente para dar una excelente predicción.

Como se puede observar en la gráfica 4.26, la Red Neuronal llega hasta una precisión de 0.9 con 3 capas convoluciónales y 3 capas completamente conectadas a pesar de que el número de imágenes para entrenamiento y de prueba por cada clase son menores que en el dataset utilizado para el reconocimiento de expresiones faciales.

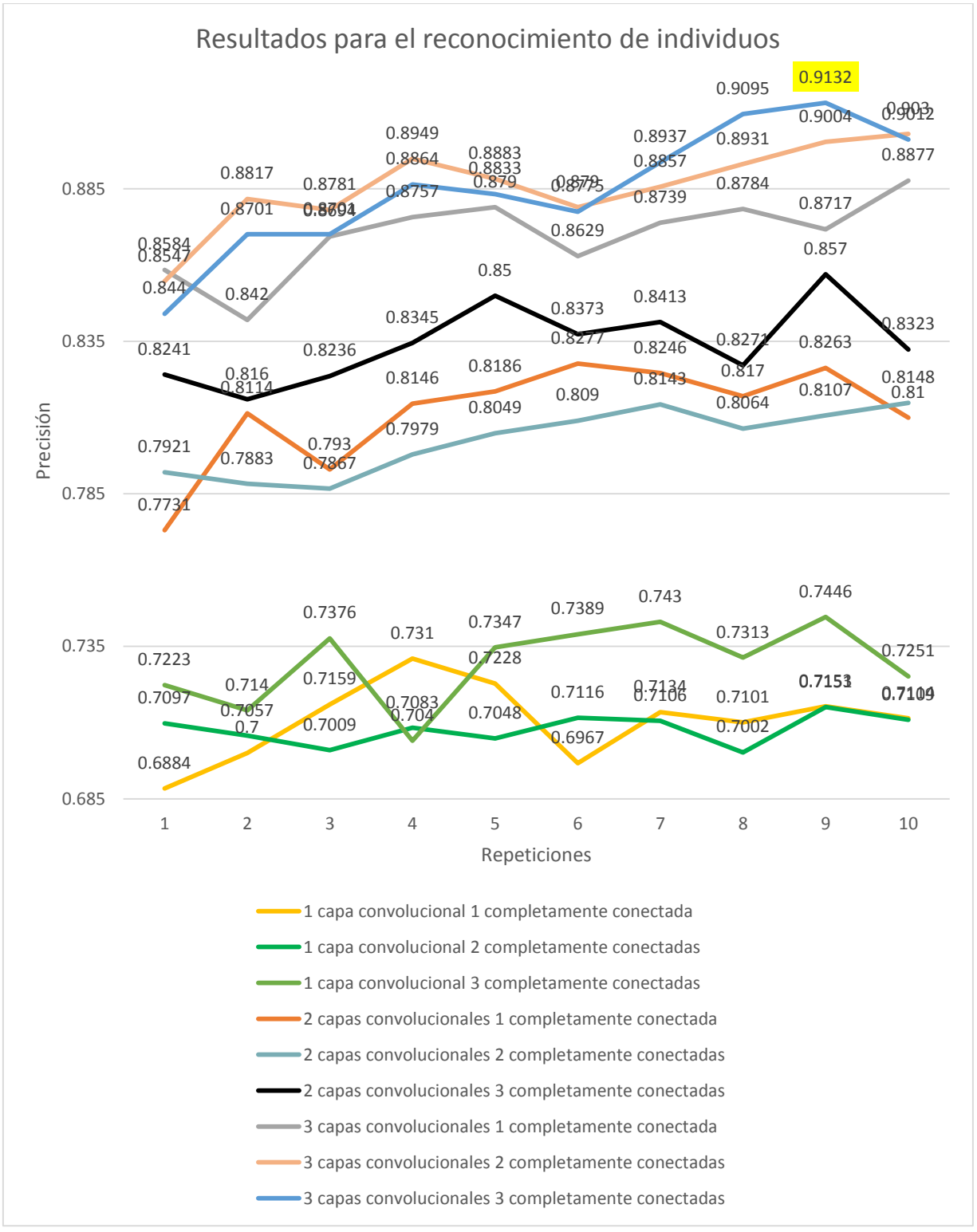


Imagen 4. 26 Gráfica de Resultados

Capítulo 5
Conclusiones y trabajos futuros

En base a los objetivos propuestos inicialmente, en este trabajo, se implementó una Red Neuronal Convolucional que clasifica imágenes para efectuar dos tareas de visión computacional: el reconocimiento de expresiones faciales y el reconocimiento de individuos.

Se diseñó e implemento una red neuronal estándar, la cual no está sujeto a un tipo de específico de imágenes y si el problema requiere clasificar imágenes diferentes a los propuestos en esta tesis, basta con cambiar el set de datos.

En las pruebas realizadas se utilizaron 2 set de datos diferentes, uno de expresiones faciales y otro con imágenes de personajes animados. A pesar de que se utiliza la misma configuración de la red en ambas tareas, los resultados de la precisión son diferentes. Esta diferencia se debe a que la complejidad de la tarea también es diferente, cuando se desea reconocer personajes de los Simpson, el entrenamiento se hace mediante imágenes del personaje a reconocer y, como son personajes animados, las diferencias son muy marcadas. Por otro lado, para la tarea de reconocer expresiones faciales, los datos de entrenamiento contienen imágenes de personas diferentes, hombres, mujeres, niños, niñas, caricaturas, personas de diferentes nacionalidades y edades haciendo así más complejo este proceso, lo que demuestra que para el problema a tratar es importante tener un conjunto de datos de entrenamiento representativo del problema.

Por otra parte se encuentran las limitaciones computacionales, la necesidad de uso de GPUs para hacer procesamientos más rápidos, ya que con una CPU el tiempo de entrenamiento varía dependiendo de los recursos que se tengan disponibles.

Es importante mencionar que si la red neuronal no proporciona una precisión del 100%, sobrepasa la línea base que marca el azar correspondiente a 1/6 [7], con lo cual se puede decir que el modelo está extrayendo las características de las imágenes para poder proporcionar un resultado. En cuanto a la tarea de reconocimiento de personajes animados “los Simpson” la precisión alcanza el 90% al aumentar el número de individuos a reconocer este porcentaje puede llegar a variar requiriendo más tiempo de entrenamiento. En cuanto al proceso de reconocimiento de expresiones faciales se muestra que se requieren más capas convolucionales para mejorar la precisión. Esto requiere de más tiempo de entrenamiento para una computadora convencional.

5.1 Trabajos futuros

Debido a la gran variedad de redes neuronales existentes al día de hoy, existen muchas líneas de investigación que podrían servir como continuación a este proyecto. A continuación, se presentan algunas:

- Creación de una red neuronal que, en vez de reconocer imágenes, sea capaz de reconocer patrones de sonido para, por ejemplo, reconocer cual es la persona que está hablando por una grabación.
- Creación de una CNN cuyo lote de imágenes sea de unas dimensiones más grandes, de manera que esto fuerce el uso de más capas en la red. Esto se podría complementar con, por ejemplo, tomar las imágenes a mano mediante una cámara (de señales de tráfico, por ejemplo) y que un determinado sistema actúe en función de lo que detecte la red.
- Usar diferentes metodologías para entrenar a la red, ya que en este proyecto siempre se usó el aprendizaje supervisado. Sin embargo, existen otros tipos de aprendizaje como el aprendizaje no supervisado.
- Reconocimiento de caracteres, uno de los trabajos futuros seria el reconocimiento de caracteres en una imagen, mediante la librería Keras de Python no sería tan complicada y puede ser utilizado una red convolucional para el reconocimiento de estos caracteres.
- Implementar las pruebas en una GPU. Para comparar la eficiencia utilizando más capas convolucionales.

Bibliografía

[1] WIKIPEDIA, «Inteligencia artificial,» Wikipedia, 23 Julio 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Inteligencia_artificial. [Último acceso: 24 Julio 2018].

[2] F. Mihaich, «Aplicación de redes neuronales en la clasificación de imágenes,» 21 julio 2014. [En línea]. Disponible: http://www2.famaf.unc.edu.ar/institucional/biblioteca/trabajos/638/17067.pdf. [Último acceso: 2017 octubre 12].

[3] M. A. R. FLORES, «DESARROLLO DE UN SISTEMA BASADO EN UN,» Abril 2010. [En línea]. Disponible: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/938/Tesis.pdf?sequence=1. [Último acceso: 12 Octubre 2017].

[4] E. C. Pardos, «Técnicas de reconocimiento facial mediante redes neuronales,» Abril 2004. [En línea]. Available: file:///C:/Users/abel/Desktop/10200404.pdf. [Último acceso: 12 Octubre 2017].

[5] F. J. N. Sánchez-Agustino, «Diseño de un sistema de reconocimiento,» 06 2016. [En línea]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/52222/7/fnunezsTFM0616memoria.pdf. [Último acceso: 12 06 2018].

[6] C. A. M. Zabaleta, «Interpretación del lenguaje,» 2017. [En línea]. Available: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/13736/Tesis.pdf?sequence=1. [Último acceso: 17 06 2018].

[7] P. P. Cruz, Inteligencia artifi cial con aplicaciones a la ingeniería, Mexico: Alfaomega Grupo Editor, S.A. de C.V., México, 2010.

[8] S. J. R. y. P. Norving, Inteligencia Artificial Un Enfoque Moderno, Madrid: Pearson, 2014.

[9] P. I. V. y. I. M. León, Redes Neuronales Artificiales Un Enfoque Practico, Madrid: Pearson Educación, 2004.

[10] WikipediA, «Red neuronal artificial,» 2 Octubre 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_artificial. [Último acceso: 05 10 2017].

[11] WIKIPEDIA, «Redes neuronales convolucionales,» WIKIPEDIA, 20 07 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales. [Último acceso: 01 08 2018].

[12] P. S. Foundation, «python,» Python Software Foundation, 2001. [En línea]. Available: http://www.python.org/. [Último acceso: 07 08 2018].

[13] About Anaconda Cloud, «About Anaconda Cloud,» About Anaconda Cloud, 2018. [En línea]. Available: https://anaconda.org/. [Último acceso: 07 08 2018].

[14] J. Brownlee, «Introducción suave al algoritmo de optimización de Adam para el aprendizaje profundo,» Machine Learning Mastery, 3 julio 2017. [En línea]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Último acceso: 11 mayo 2018].

[15] J. Villanueva-Valle, «Facial Expression of Emotion,» kaggle, 2018. [En línea]. Available: https://www.kaggle.com/sivlemx/facial-expression-of-emotion. [Último acceso: 07 08 2018].

[16] alexattia, «The Simpsons Characters Data,» kaggle, 2018. [En línea]. Available: https://www.kaggle.com/alexattia/the-simpsons-characters-dataset#simpsons_dataset.zip. [Último acceso: 07 08 2018].

[17] J. L. Devore, «Probabilidad y Estadística para,» de *Probabilidad y Estadística para*, Cengage Learning Editores,, 2008.

[18] Wikipedia, «Imagen Digital,» 07 Febrero 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Canal_(imagen_digital). [Último acceso: 05 10 2017].

[19] wikipedia, «TensorFlow,» WikipediA la enciclopedia libre, 1 11 2017. [En línea]. Available: https://es.wikipedia.org/wiki/TensorFlow. [Último acceso: 04 11 2017].

Anexo A

Código de una Red Neuronal Convolucional de 1 capa de convolución y 1 capa completamente conectada en Keras.

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
categorias = 6
training_img = 3001
test_img = 1001
classifier = Sequential()
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())

classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'categorical')
classifier.fit_generator(training_set,
                         steps_per_epoch = training_img * categorias,
                         epochs = 10,
                         validation_data = test_set,
                         validation_steps = test_img * categorias)
print("inicio de programa para Terminar presiona 0")

classifier.save('modelo_1conv_1fully.h5')
print(": modelo Guardado ")
```

Anexo B

Código Red Neuronal Convolucional con 2 capas de convolución

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dens
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
),
```

```
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'categorical')
classifier.fit_generator(training_set,
                         steps_per_epoch = training_img * categorias,
                         epochs = 10,
                         validation_data = test_set,
                         validation_steps = test_img * categorias)
classifier.save('modelo_2conv_1full.h5')
print(": Modelo Guardado : ")
```

Anexo C

Código de una Red Neuronal Convolucional de 3 capas de convolución y 1 capa completamente conectada en Keras.

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dens
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(128, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = categorias, activation = 'softmax'))
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory('dataset/training_set',
```

```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'categorical')
classifier.fit_generator(training_set,
                         steps_per_epoch = training_img * categorias,
                         epochs = 10,
                         validation_data = test_set,
                         validation_steps = test_img * categorias)
classifier.save('modelo_3conv_1full.h5')
print(": Modelo Guardado : ")
```