Universidad nacional del sur

Departamento de Ingeniería Eléctrica y de Computadoras

Aprendizaje de máquina

Trabajo Final:

Introducción a redes neuronales

Profesor:

Del Rio, Claudio

Alumno:

Vettori, Kevin O.

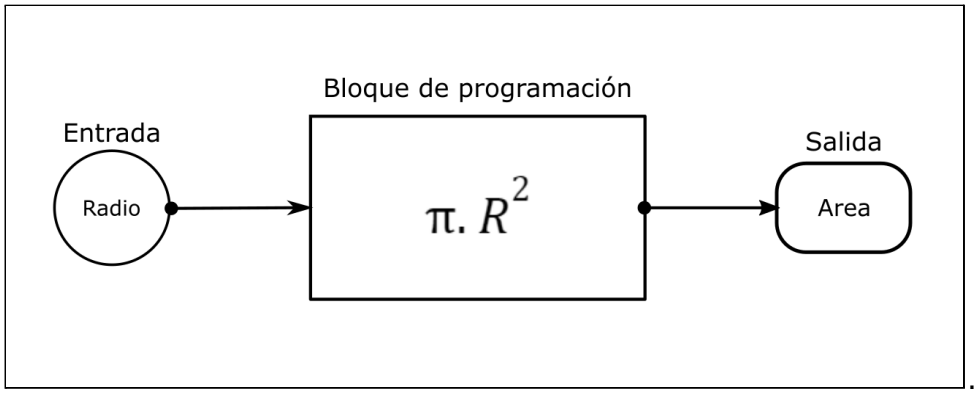
Resumen:

Este es un trabajo final de la materia de *aprendizaje de máquina* ,en el mismo se aventuró a aprender un poco más sobre este mundo del *Machine learning*. Por lo que se ha intentado trabajar aprendiendo con redes neuronales, esto se hizo en base a un dataset sacado de internet sobre imágenes de frutas llamado *Fruit 360*. Básicamente se trata de resolver un problema de clasificación de las frutas antes mencionadas con una red neuronal entrenada a través de la herramienta online *Google Colab.* Así que primero vamos a ir aprendiendo un poco sobre el tema, para al final ver los resultados obtenidos. Podemos ver el código implementado en este link de GitHub.

Redes neuronales:

Introducción a aprendizaje de máquina

En la programación normal o "regular" se suele realizar algoritmos donde se transforma una o más entradas en uno o varios resultados. Por ejemplo, si un programa calculará el área de una circunferencia, la entrada sería el radio y la salida el área. Podría verse como el siguiente diagrama de bloques.

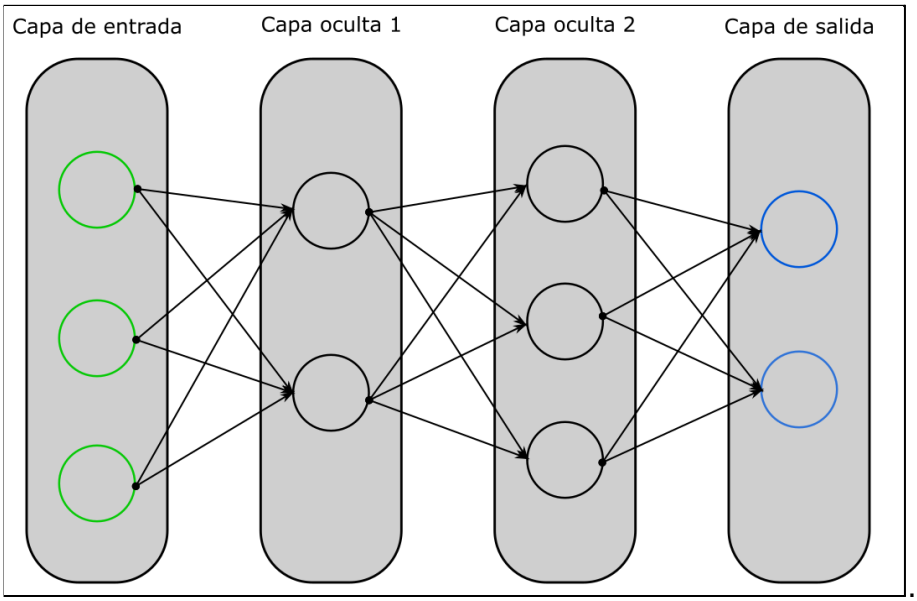


Si bien para el usuario el bloque de programación puede verse como una caja negra, estos siempre están construidos por un programador o un grupo de estos, el cual estructuró ciertos algoritmos basados en reglas y lógicas para realizar determinada tarea. En cambio, en el aprendizaje automático no se sabe necesariamente cómo se transforman las entradas en resultados.

Para esto se necesita entrenar un modelo, brindando entradas y resultados para que el modelo elija la forma conveniente de llegar a estos últimos. En este escrito veremos el modelado basado en redes neuronales.

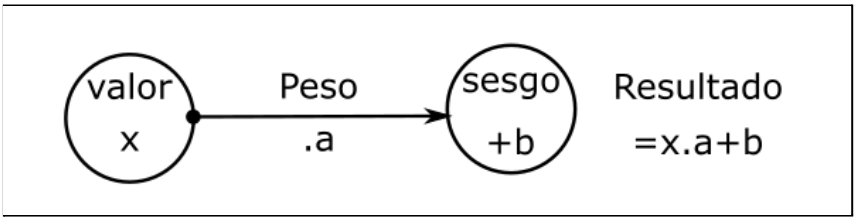
¿Qué es una red neuronal?

Una red neuronal consiste en un conjunto de unidades llamadas nodos, perceptrones o neuronas interconectadas para transferir datos entre sí. Estas redes se separan en capas, cada capa puede tener una o más neuronas. Mínimamente constan de dos capas, una de entrada donde se reciben los datos de entrada, y una de salida donde se obtiene el resultado buscado. Las capas intermedias suelen llamarse capas ocultas, pero entraremos en esos detalles más adelante. Para acompañar esta idea presentamos el siguiente diagrama:



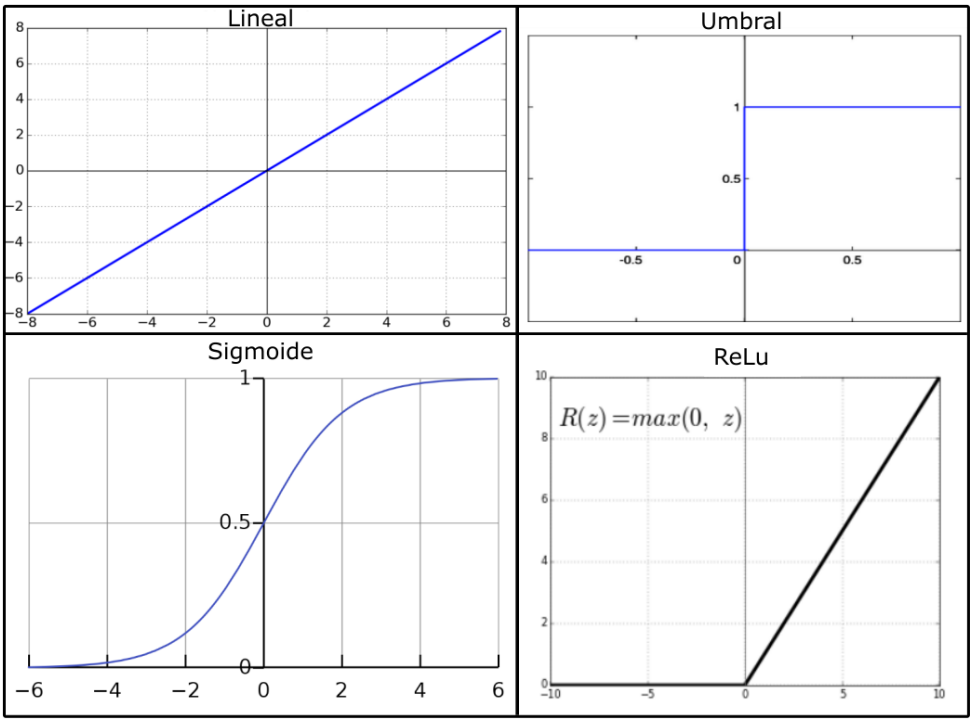
¿Cómo actúa una red neuronal?

Teniendo ahora una idea de como se ve una red neuronal, nos queda entender como ésta es capaz de resolver nuestro problema y llegar a un resultado. Para empezar cada conexión entre nodos tiene un valor numérico llamado *peso*, este se podría ver como la importancia de la conexión, este peso se le multiplica al valor que envía el "nodo emisor". Luego se le suma el *sesgo*, que es un valor propio del "nodo receptor". Para fortalecer la explicación vemos la siguiente imagen, que presenta lo dicho con sólo dos nodos.



Estos valores inician aleatoriamente y se van modificando a medida que la red se va entrenando. Ya sabemos que necesitamos valores de entrada y salida para que nuestra red se pueda entrenar, ¿Cuantos? Eso es un tema de estudio por sí mismo, depende mucho del problema que queramos resolver.

Volviendo a la capacidad de nuestra red neuronal, vemos que con un *sesgo* y *peso* podemos resolver ecuaciones lineales. Entonces, ¿Qué pasa si queremos resolver algo más complejo? Ahí es donde entran las antes mencionadas capas ocultas y algo llamado *función de activación.* Esta *función de activación* devuelve una salida que será generada por la neurona dada una entrada o conjunto de entradas. Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir. Existen varios tipos de funciones de activación, entre estas pueden ser las lineales, las cuales no modificarían nuestro ejemplo anterior. También pueden ser no lineales, como las de tipo umbral, siendo 1 o 0, si el valor del nodo supera el valor umbral o no. Existen varios tipos de funciones de activación, entre las más usadas están las tipo ReLu y las Sigmoide. Siguiendo el patrón de ayuda con diagramas presentamos las 4 funciones mencionadas recientemente.



Estas funciones no lineales, sumadas a la cantidad de capas y nodos que se pueden agregar a una red neuronal, nos permite resolver infinidad de casos ya sean complejos o no. Ahora con una idea sobre lo que es y cómo funciona una red neuronal, podemos adentrarnos en este trabajo.

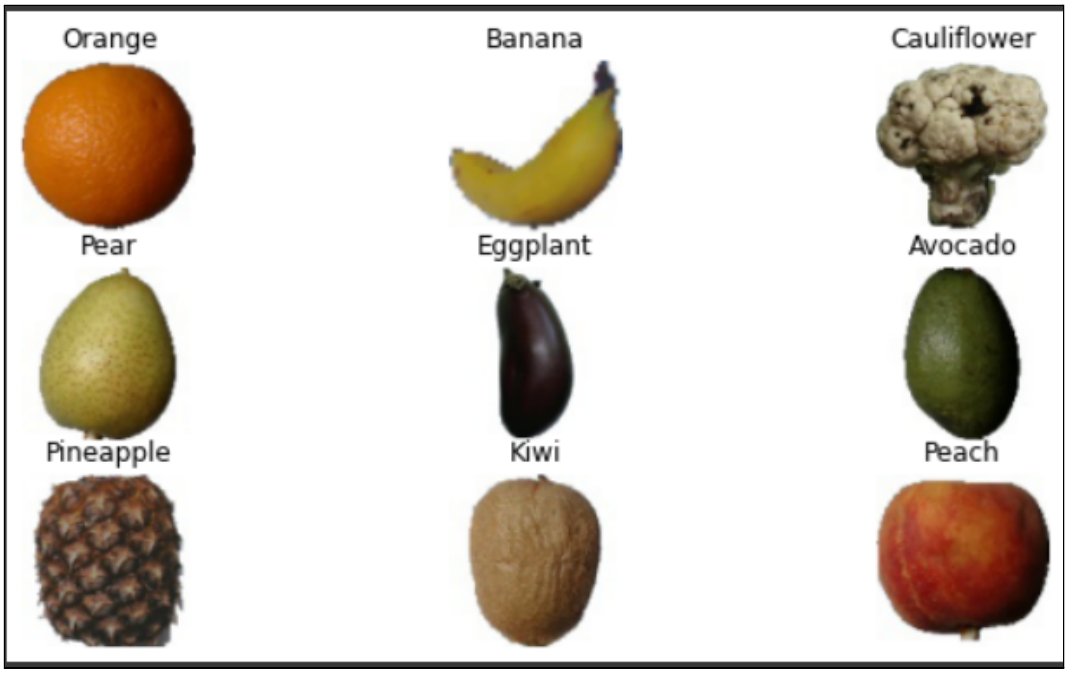
Implementación:

Motivación

Este trabajo se realizó con el fin de ser el trabajo final del cursado de la materia aprendizaje de máquina. A priori se eligió trabajar sobre redes neuronales, porque era avanzar un poco más en los contenidos de la materia y entrar un poco más en el mundo del "deep learning". Entonces restaba elegir en qué aplicación se podría optar el uso de redes neuronales. Como nativo de una ciudad que depende fuertemente de la fruticultura, se me ocurrió ver si podía lograr algo semejante a clasificador del estado de fruta en un galpón de empaque. Por ejemplo, separar manzanas entre las de descarte o las comercialización. Pero crear un dataset, era una tarea extensiva para lo que se refiere al alcance de una materia cuatrimestral. Entonces empecé con la búsqueda de algún dataset similar, de esta forma encontré el dataset *Fruit 360* en Kaggle que veremos con mayor detalle a continuación.

Sobre el dataset "Fruit 360"

Este dataset hecho por Horea Muresan y Mihai Oltean, en el cual se cuenta con un aproximado de 100 frutas con más de 500 fotos de cada una de ellas, divididas ya entre entrenamiento y testeo. Estas imágenes fueron realizadas fruta por fruta, atravesadas por una mecha haciéndolas rotar a 3 rpm. Este dataset está disponible tanto en Kaggle como en Github. Para más información sobre este, se puede acceder libremente a él. Damos un vistazo rápido de las imágenes que podemos encontrar en el dataset.

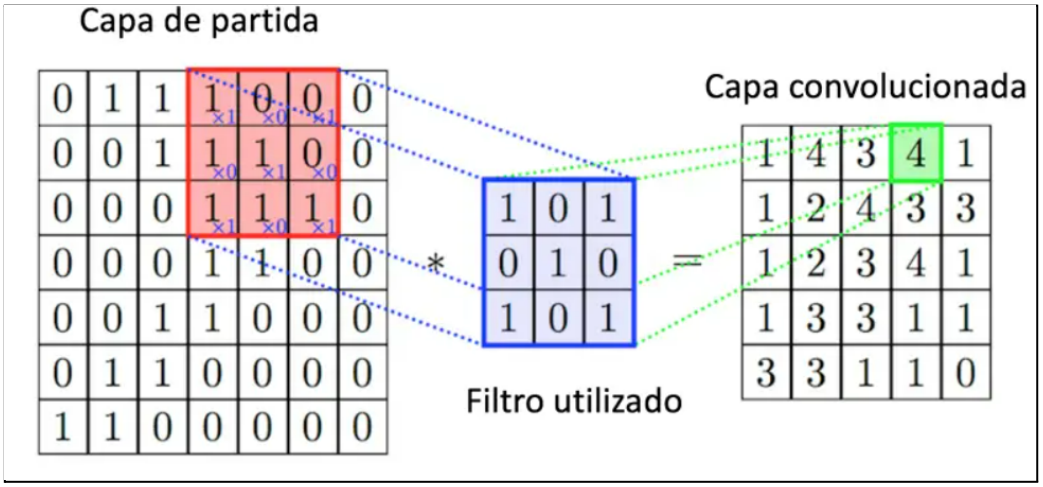


Trabajar con imágenes

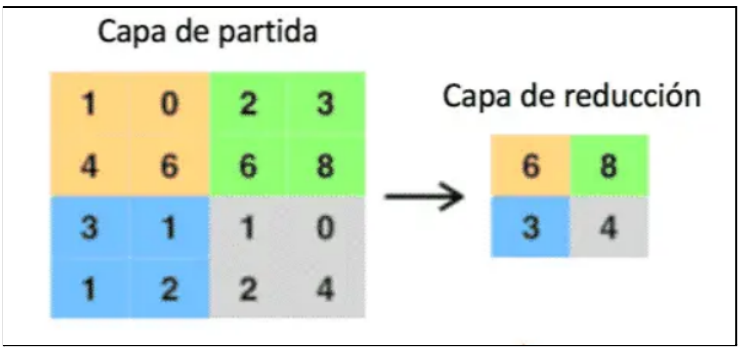
Conociendo el dataset con el que se va a trabajar, está claro que se va a tener que trabajar sobre imágenes. El procesamiento de imágenes es otro campo de estudio por sí mismo, por lo que se intentará abordar sobre esto muy poco ya que no es nuestro punto de enfoque, aún así hay varias cosas a tener en cuenta. Una de estas, es cómo usar una imagen como entrada en nuestra red neuronal. Para este caso tenemos la suerte de que el dataset utilizado nos brinda las imágenes en 100x100 píxeles, lo que nos simplifica el trabajo. Dándonos la posibilidad de que cada píxel de la imagen, sea un nodo de entrada de nuestra red neuronal. Una practica muy usada tambien es la de llevar la imagen a blanco y negro para disminuir la complejidad de la misma.

Otra práctica muy recurrente cuando se trabaja con algoritmos de entrenamiento es expandir virtualmente los datos de entrenamiento, por ejemplo girando algunas imágenes para obtener más datos y estar mejor entrenada para otros tipos de circunstancias. Esto es algo que no se aplicó a este trabajo debido a que por las características del dataset y la finalidad del proyecto no fue necesario.

Algo muy importante en las redes neuronales que trabajan con imágenes, son las *redes convolucionales*, estas permiten a la red extrapolar los datos que brindan los píxeles individualmente a características propias de la imagen. Esto lo logra viendo no solo el valor numérico del píxel asignado, si no su entorno, este le permite asignar un valor tanto por las características del mismo como por lo que está rodeado. Podría decirse que hacen uno del refrán “dime con quien anda y te diré quien eres” similar a la imagen a continuación.



Algo sumado a esta *capa de convolución* es una llamada *capa de agrupación*, el trabajo de esta capa se puede resumir en que toma una matriz de píxeles y elige el más importante entre estos. Esto da como resultado a una imagen más pequeña pero que conserva los parámetros más importantes. Como vemos en la siguiente imagen.



Estructurado de Red neuronal

El armado de una red neuronal puede darse de muchas formas, estas no solo dependen de la librería que se utilice o cual sea su entrada. Así mismo no significa que una red compleja sea mejor que una simple, incluso la misma red entrenada de otra forma puede cambiar drásticamente su funcionamiento. Incluso existen redes pre-armadas o pre-entrenadas para facilitar algunos trabajos. En este caso se utilizara una red relativamente simple, para esto se usará la librería de *Keras*, y se eligió el modelo *secuencial* que es de los más simples que hay dentro de lo que nos brinda la librería. A continuación se muestra una captura del código sobre el sencillo armado de la red neuronal.

```
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = array_image.shape))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(32,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(numberOfClass)) # output
model.add(Activation("softmax"))
```

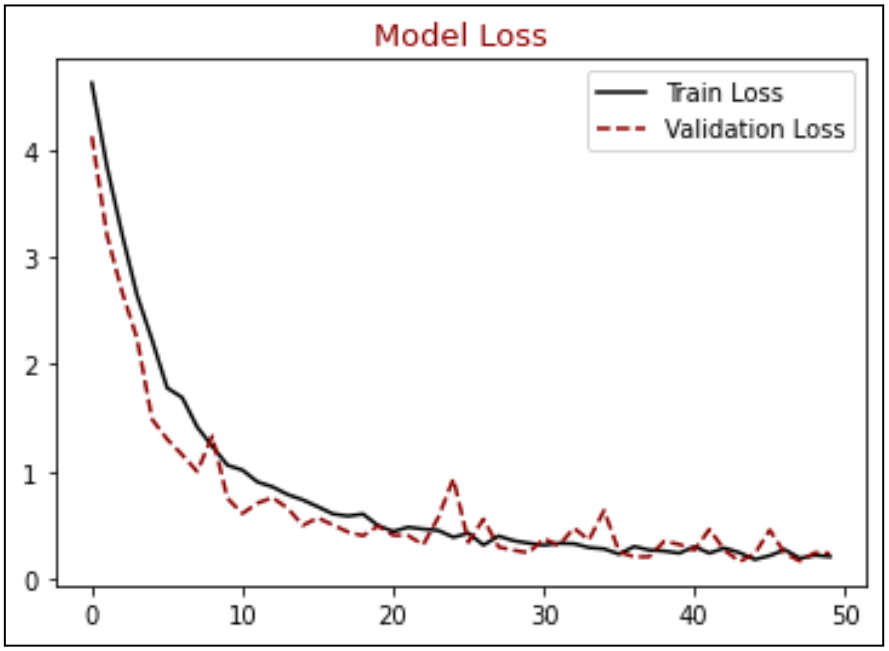
Como vimos anteriormente, para casos con imágenes siempre se suele usar capas convolucionales con otra capa de agrupación. Hemos resuelto esta red con 3 pares de capas de convolución (Conv2D) y agrupación (MaxPooling2D). Las dos primeras son

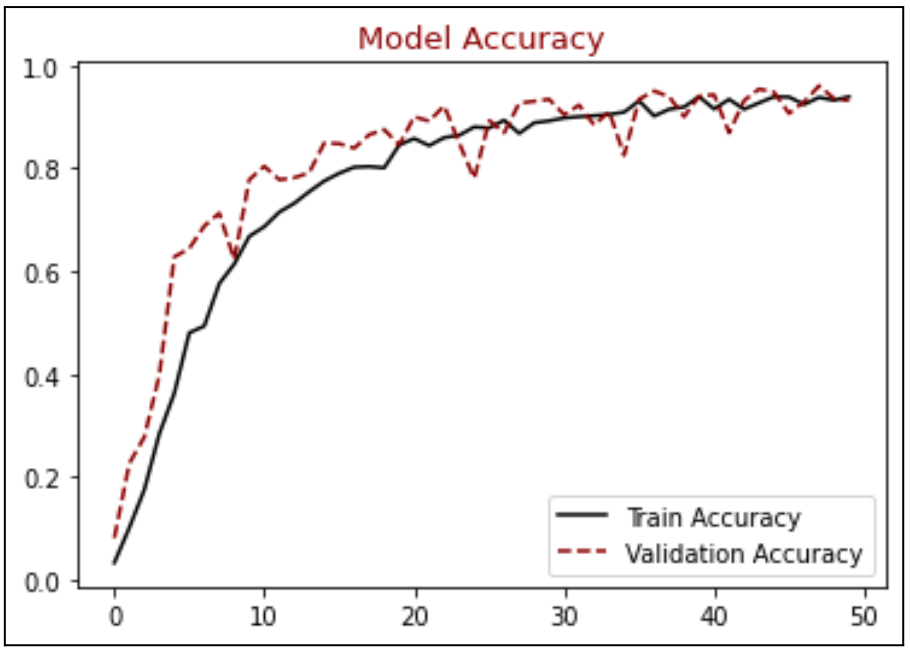
igualmente densas, mientras la última capa de convolución es el doble de densa que las primeras teniendo 64 núcleos en lugar de 32.

Luego una capa de aplanado (Flatten), que se podría ver como que pasa de un campo multidimensional a un arreglo lineal. Posteriormente una capa *densa* (dense), que sea densa significa que todas las neuronas de esta capa están conectadas con todas las de la capa siguiente. Vemos que siempre optamos por una activación *ReLu* la cual se explicó anteriormente. Luego vemos que hay una línea que dice *Dropout*, esto es una técnica de regularización para reducir el "sobreajuste" de la red. Esto lo logra mediante el "abandonar" u omitir aleatoriamente neuronas (tanto ocultas como visibles) durante el proceso de entrenamiento de la red neuronal. Finalmente la función *softmax* es una generalización de la regresión logística que puede ser aplicada a datos continuos. Soporta sistemas de clasificación multinomial, por lo que se convierte en el recurso principal utilizado en las capas de salida de un clasificador.

Resultados:

Luego de entrenar nuestra red neuronal durante unas cuantas épocas, podemos graficar como las pérdidas y los aciertos fueron evolucionando a través de las distintas épocas. Vemos ambos gráficos a continuación:





Vemos como en las primeras 10 épocas nuestro algoritmo mejora rotundamente, mientras en las más cercanas a 50 el modelo va hilando más fino. Podríamos decir que en la época 20 ya era un modelo utilizable acercándose al 80% de precisión.

Conclusiones:

Vemos que si bien armar una red neuronal desde cero mediante programación podría ser una tarea abismal. Pero gracias a la facilidad y libre accesos de librerías (en este caso para python), así como a su bibliografía, uno con acceso internet y suficiente tiempo puede alcanzar esta tecnología de vanguardia. Por ejemplo este mismo proyecto fue hecho en la plataforma online *Google Colab* de forma totalmente gratuita. Si bien a causa de esto se ha perdido la intención de comparar distintas redes neuronales debido a la continua desconexión o superando el tiempo de conexión y pérdidas de muchas horas compilando, lo cual con una plataforma no online o con el pago de una membresía a colab no hubiera ocurrido. También podría haberse evitado si estuviera más versado en el tema o tal vez si hubiera hecho algoritmos más eficientes o rápidos. Aun así esto es un detalle menor, porque debido a este trabajo realizado se ha aprendido lo compleja o sencilla que puede ser la implementación de una red neuronal. La cantidad de información y algoritmos pre hechos que ha generado la comunidad para un entorno más sencillo.

Referencias

[1]. Haykin, Simon (1998). *Neural Networks: A Comprehensive Foundation* (2 edición).

[2]. Ml4a. Neural networks.(https://ml4a.github.io/ml4a/neural_networks/).

[3]. Andrey Kurenkov. A Brief History of Neural Nets and Deep Learning.
(https://www.skynettoday.com/overviews/neural-net-history)

[4]. Mihai Oltean. Fruits 360 dataset: A dataset of images containing fruits and vegetables
(https://www.kaggle.com/datasets/moltean/fruits?select=fruits-360_dataset)