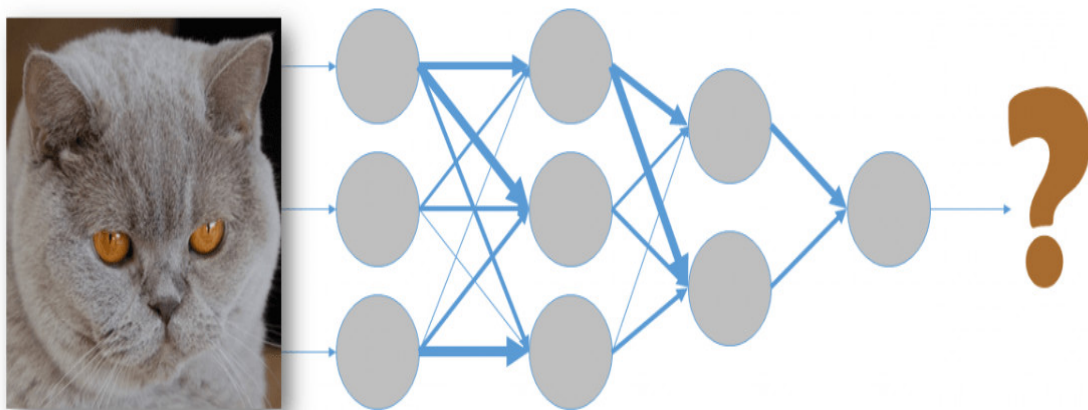


APLICACIÓN DE RED NEURONAL.



Inteligencia Artificial.
26 de Junio del 2021

Tabla Informativa.

Periodo	Mayo - Agosto 2021
Cuatrimestre	14° “A”
Asignatura	Inteligencia Artificial
Corte	3
Actividad	171048 ROQUE JIMENO_C3.A3
Fecha de asignación	2021.07.07
Fecha de entrega	2021.07.31

Matricula	Nombre
171048	Alexis Martin Roque Jimeno

Índice.

Tabla Informativa.	2
Índice de figuras.	4
Índice de imágenes.	5
Índice de fragmentos de código.	5
Índice de tablas.	5
Índice de gráficas.	5
Introducción.	6
Métodos.	7
Definición de variables.	7
Pre procesamiento de imágenes.	7
Pasos por épocas.	8
Definiendo el modelo de Red.	8
Compilar el modelo de red.	12
Entrenamiento del modelo de red.	12
Guardar modelo.	13
Implementación.	14
Librerías utilizadas.	14
Arquitectura de la red.	15
Arquitectura del dataset.	17
Entrenamiento.	18
Resultados.	24
Entrenamiento 1.	24
Entrenamiento 2.	25
Entrenamiento 3.	26
Discusiones.	27
1era. Discusión.	27
2da. Discusión.	27
3ra. Discusión.	27
4ta. Discusión.	27
5ta. Discusión.	27
Análisis y conclusiones.	28
Referencias	29

Índice de figuras.

Figura 1. Estructura de neurona biológica.	5
Figura 2.Arquitectura de la red.	14

Índice de imágenes.

Imagen 1. MaxPooling2D	8
Imagen 2. Ejemplo de imágenes dataset.	16
Imagen 3.Proceso entrenamiento 1.	17
Imagen 4. Proceso entrenamiento 2.	19
Imagen 5. Proceso entrenamiento 3.	21
Imagen 6. Resultados entrenamiento 1.	23
Imagen 7. Resultados entrenamiento 2.	24
Imagen 8. Resultados entrenamiento 3.	25

Índice de fragmentos de código.

Fragmento código 1. Capa convolutional2D.	7
Fragmento código 2. 2da y 3era capa.	9
Fragmento código 3. Capa plana.	9
Fragmento código 4. Capa Densa.	10
Fragmento código 5. Dropout.	10
Fragmento código 6. Capa de salida.	10
Fragmento código 7. Compilación del modelo de red.	11
Fragmento código 8.Entrenamiento del modelo de red.	12
Fragmento código 9. Guardar modelo.	12

Índice de tablas.

Tabla 1. Cantidad de imágenes.	16
Tabla 2. Matriz de confusión entrenamiento 1.	23
Tabla 3. Matriz de confusión entrenamiento 2.	24
Tabla 4. Matriz de confusión entrenamiento 2.	25

Índice de gráficas.

Gráfica 1. Resultados obtenidos en entrenamiento 1 .	18
Gráfica 2. Accuracy and loss entrenamiento 1 .	18
Gráfica 3. Resultados obtenidos en entrenamiento 2.	20
Gráfica 4. Accuracy and loss entrenamiento 2.	20
Gráfica 5. Accuracy and loss entrenamiento 3.	22
Gráfica 6. Resultados obtenidos en entrenamiento 3.	22

Introducción.

Las redes neuronales son una forma de emular ciertas características y comportamientos propios de los seres humanos, tales como la capacidad de memorizar y de asociar hechos.

Una red neuronal es un sistema para el tratamiento de información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona.

En la estructura neuronal biológica las neuronas constituyen procesadores de la información sencillos. Poseen un canal de entrada de información que son las dendritas, un órgano de cómputo, que es el cuerpo celular o soma, y un canal de salida que es el axón.

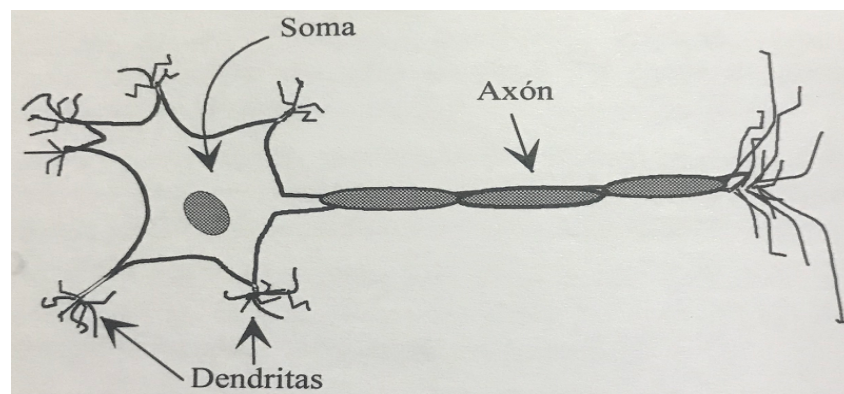


Figura 1. Estructura de neurona biológica.

En un modelo de red neuronal, como se mencionó anteriormente, está inspirado en el sistema biológico humano, ya que incorpora un conjunto de entradas y proporcionar cierta respuesta, la cual se propaga para conformar una salida o entrada de otras neuronas, constituyendo así una red de neuronas las cuales colaboran entre sí, para un objetivo común.

Métodos.

1. Definición de variables.

Para comenzar, hay que definir algunos parámetros. Primero, se comienza por definir las variables que almacenarán la ruta de nuestro dataset.

Para estas variables se necesitarán 2, ya que una almacenará la ruta de las imágenes para entrenamiento, y otra para las validaciones.

Lo siguiente es definir los parámetros de nuestras imágenes. Es importante definir el alto y ancho de la imagen correspondiente al tamaño de nuestras imágenes en el dataset. Y definir el número de canales (RGB).

2. Pre procesamiento de imágenes.

Antes de realizar el análisis de las imágenes, es importante prepararlas, para ello se hace uso del módulo de *ImageDataGenerator*, el cual proviene de Keras.

Utilizando este módulo, le pedimos que para las imágenes de entrenamiento le realice una serie de modificaciones tales como:

1. *rescale*: Esto hará que todas las imágenes tengan un reescalado de entre 0 y 1 pixel para tener más eficiencia.
2. *shear_range*: Esto tiene como función inclinar un poco las imágenes.
3. *zoom_range*: Aplicará un zoom sobre la imagen.
4. *horizontal_flip*: Invertirá la imagen.

También debemos procesar las imágenes de validación, pero éstas únicamente se les aplicará el reescalado.

Con esto definido, lo siguiente es pasar al modelo de entrenamiento, mediante la operación *flow_from_directory*.

En esta función es importante definir algunos parámetros:

1. El directorio especificado.
2. Resolución de las imágenes.
3. El *batch_size*, o el total de imágenes a entrenar por cada bloque.
4. *Class_mode*, o las etiquetas que tendrá.

Esta configuración debe realizarse tanto para las imágenes de entrenamiento, como las de validación.

3. Pasos por épocas.

Es importante definir 2 cosas. La primera es cuántas épocas tendrá, y la siguiente es el `batch_size`, o el total de imágenes que pasarán por cada bloque.

Una vez que se tienen estos parámetros, se puede definir el total de pasos por cada época, o mejor conocido como: `steps_per_epoch`.

Esto define el total de pasos (lotes de muestras) que pasan durante una época. Cuando este valor no es definido, por default, adquiere el valor de `None`. Este valor es equivalente al total de muestras que se tienen.

Para definir este valor, basta con realizar una operación. La cual consiste en dividir el total de muestras, entre el `batch_size`.

P.e. Asuma que tiene un total de muestras de 1500, y un `batch_size` de 32. Esta división da como resultado 46. Lo cual indica, que por cada época, estará pasando un lote de 46 muestras.

De esta misma manera, es necesario definir tanto para los datos de entrenamiento, así como para los de validaciones.

4. Definiendo el modelo de Red.

Una vez realizada todas las configuraciones y el preprocesamiento de las imágenes, se puede comenzar por definir el modelo de la red neuronal.

En este caso, el modelo de red definido fue el secuencial. Dicho modelo, permite que las capas que se van añadiendo, se agreguen en secuencia.

Definido el modelo, podemos ir añadiendo las capas.

La primera capa en agregar será la siguiente:

```
model.add(Convolution2D(32, (3, 3), padding='same', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Fragmento código 1. Capa convolutional2D.

Esto indica, que la primera capa que se agrega es una convolucional 2D. El primer parámetro que define esta función es el total de filtros que utiliza (32), el siguiente indica el tamaño del kernel (3, 3). El padding lo que hace, es agregar el tamaño adecuado a todas las imágenes para que no varíen.

El input_shape determina la configuración de la imagen. Estos parámetros fueron definidos anteriormente. IMAGE_WIDTH y IMAGE_HEIGHT corresponden al ancho y alto de la imagen. Al tratarse de una imagen a color, el IMAGE_CHANNELS corresponde a los colores RGB.

Y por último, la función de activación. En este caso se utilizó la función 'relu', esta es utilizada por que únicamente toma valores positivos, al tratarse de un análisis de imágenes, únicamente nos interesan los valores positivos que pasarán a la siguiente capa de convolución.

La siguiente capa que se agrega es la de MaxPooling2D. Esta tiene como objetivo reducir las dimensiones de la imagen.

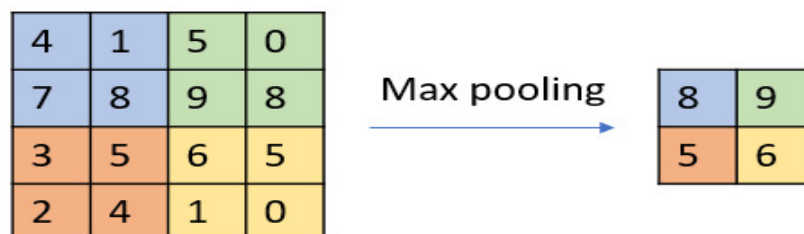



Imagen 1. MaxPooling2D

En esta imagen se observa el resultado de aplicar un MaxPooling de 2x2. Reduciendo una imagen que era originalmente de 4x4, a una imagen de 2x2.

Para la siguiente capa de convolución ya no es necesario declarar la configuración de la imagen (`input_shape`), ya que esto únicamente se realiza en la primera capa. De igual forma, no es necesario definir una función de activación.

Para las siguientes capas, únicamente tuvo como variación, el total filtros que se utilizan, siendo ahora estos de 64 y 128. Y manteniendo el tamaño del kernel en (3 , 3) , y con el mismo padding.

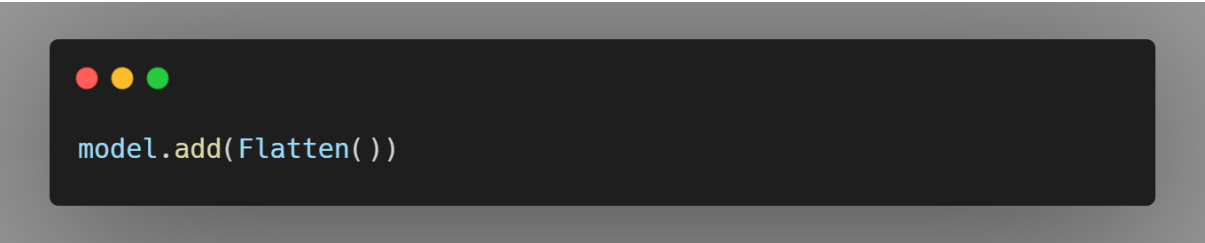


```
#2 capa
model.add(Convolution2D(64 , (3 , 3) , padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))

#3 capa
model.add(Convolution2D(128 , (3 , 3) , padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

Fragmento código 2. 2da y 3era capa.


La siguiente capa es diferente a las anteriores, ahora se aplicará un método llamado Flatten. Esto quiere decir que el volumen de imágenes se hará plano, es decir, que toda la información de la red estará en una sola dimensión.



```
model.add(Flatten())
```

Fragmento código 3. Capa plana.


Después, se crea una capa de tipo Dense, o una capa oculta de 512. Esta capa está conectada profundamente, lo que quiere decir que cada neurona en esta capa, recibe la información de todas las neuronas de las capas anteriores. Y con una función de activación de tipo 'relu'.



```
model.add(Dense(512 , activation='relu'))
```

Fragmento código 4. Capa Densa.


Seguido, se agrega una capa más con un Dropout. Esta tiene como función desactivar o apagar el 50% de las neuronas, con el objetivo de evitar el sobreajuste.



```
model.add(Dropout(0.5))
```

Fragmento código 5. Dropout.

Por último , se tiene una capa de tipo Dense para lo que son las clases. Con una función de activación en este caso, de tipo Softmax. Ya que al tratarse de un análisis de imágenes nos interesa saber cuál tiene más probabilidad y por ende, saber a que clase corresponde.

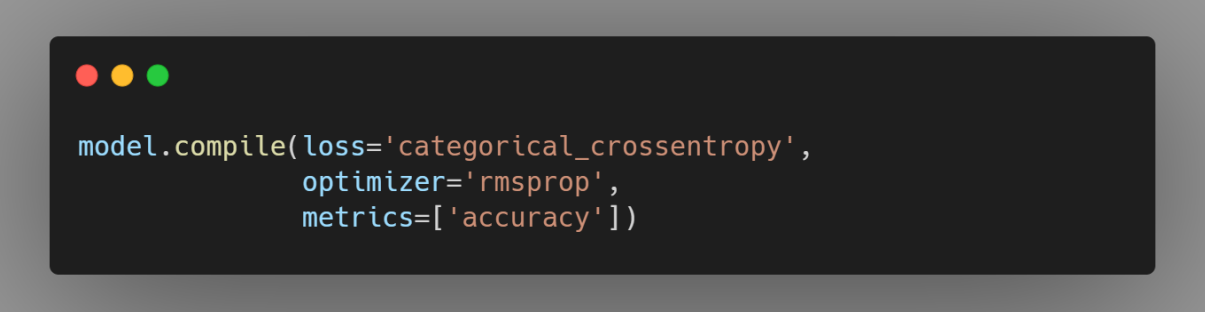


```
model.add(Dense(CLASSES , activation='softmax'))
```

Fragmento código 6. Capa de salida.

5. Compilar el modelo de red.

Con el modelo de red completo, lo siguiente es su compilación. Para ello hay que configurar el proceso de aprendizaje de la red.



```
model.compile(loss='categorical_crossentropy',  
              optimizer='rmsprop',  
              metrics=['accuracy'])
```

Fragmento código 7. Compilación del modelo de red.

Esta función recibe 3 parámetros. El primero corresponde a la función de pérdida, en este caso se está utilizando ‘categorical_crossentropy’ ya que existen 2 o más clases de etiquetas.


El segundo parámetro corresponde al optimizador. Hay varios tipos de optimizadores, P.e. sgd, Adam. En este caso, se está utilizando el optimizador ‘rmsprop’, ya que este utiliza únicamente los gradientes más recientes, lo que hace que tenga una convergencia más rápida.

El último parámetro son las métricas, acá únicamente se utiliza el ‘accuracy’ o la tasa de aciertos.

6. Entrenamiento del modelo de red.

Cuando el modelo de nuestra red haya sido compilado, es momento de realizar el entrenamiento. Para esto, se utiliza la función de fit(). Anteriormente se utilizaba la función de fit_generator() , pero en las versiones más actualizadas, esta función está obsoleta.

El fragmento de código 8, muestra como es la estructura de la función para la realización del entrenamiento.



```
H = model.fit(
    training_image,
    steps_per_epoch = pasos_entrenamiento,
    epochs = epoch,
    validation_data = validation_image,
    validation_steps = pasos_validation)
```


Fragmento código 8. Entrenamiento del modelo de red.

A esta función le indicamos que el primer parámetro que recibirá serán las imágenes de entrenamiento que anteriormente se procesaron. El siguiente parámetro corresponde al total de bloques que pasaremos, estos fueron calculados en el paso número 3. Los 'epochs', corresponden a las épocas o las vueltas que dará durante el entrenamiento.

El siguiente parámetro indica las imágenes, pero esta vez las que son utilizadas para la validación, y de igual forma, el último, indica el total de bloques que pasarán.

7. Guardar modelo.

Una vez realizado el entrenamiento de la red, lo último sería guardar el modelo obtenido. Para ello únicamente creamos el directorio donde se almacenarán nuestros archivos. Es importante que los archivos sean guardados con una extensión '.h5', ya que este formato permite el almacenamiento de datos a gran escala.



```
dir = './Model/'
if not os.path.exists(dir):
    os.mkdir(dir)
model.save('./Model/model.h5')
model.save_weights('./Model/pesos.h5')
```

Fragmento código 9. Guardar modelo.

Implementación.

1. Librerías utilizadas.

Para el desarrollo del sistema, se utilizó el lenguaje de programación [Python](#) en su versión 3.8.10 , en un sistema operativo de 64 bits.

Para la configuración de la red, se utilizaron diversas librerías. Para el desarrollo de la red, se utilizó la librería de [TensorFlow](#).

De esta librería, principalmente se utilizó el módulo de '[Keras](#)'. De este, se utilizaron algunos otros módulos como:

1. Preprocessing image: El cual servía para hacer el preprocesamiento de las imágenes.
2. Models: Este módulo es importante para el desarrollo de la red, ya que aquí es donde se define que tipo se utilizará.
3. Layers: Utilizado para definir el tipo y configuración de las capas.

Además de estas librerías, se utilizaron otras como:

1. [Matplotlib](#), la cual se utilizó para realizar la graficación de los resultados obtenidos.
2. [Numpy](#), utilizada para el cálculo numérico y análisis de datos de gran volumen.
3. [Pandas](#), la cual, de igual manera se utilizó para el análisis de datos.

De igual forma se utilizaron algunos módulos de python, como 'time', el cual se utilizó únicamente para visualizar el tiempo que tarda en procesar el entrenamiento de la red. Y el módulo de 'os', el cual se utilizó para poder acceder a los directorios.

2. Arquitectura de la red.

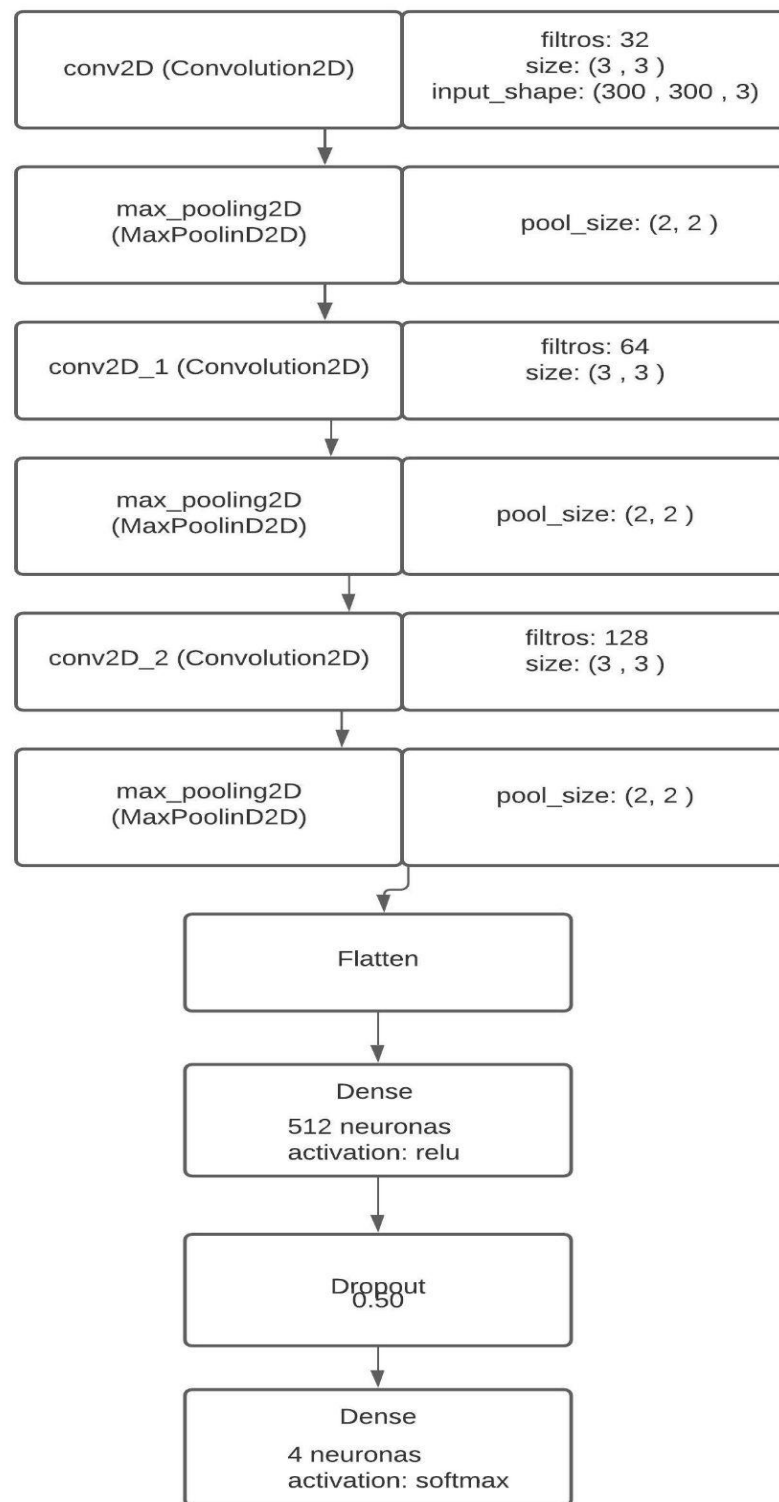


Figura 2.Arquitectura de la red.

Se estableció la capa de entrada de 300x300 (Imágenes de 300x300 píxeles) , y a partir de 3 capas convolucionales de 32 , 64 y 128 neuronas cada una.

Estas capas serán las encargadas de realizar el procesamiento de las imágenes. Después se agrega una capa de tipo flatten, esta capa lo que hará es reducir el volumen de los datos a una sola dimensión o plano.

Una capa densa o capa oculta de 512 neuronas, con una activación de tipo 'relu', y después una capa más de tipo 'Dropout' con un valor de 0.50, esto quiere decir que apagará el 50% de las neuronas para obtener diversos resultados y que el sistema no tome únicamente un camino.

Por último la capa de salida, esta capa tendrá un total de neuronas igual al número de clases que se tienen (4), y una función de activación de tipo 'softmax', ya que al tratarse de un análisis de imágenes nos interesa saber cuál tiene más probabilidad

Al ir incrementando la cantidad de capas en el modelo, se espera una mayor precisión en los resultados, cabe mencionar que esto también depende de los datos de la aplicación.

3. Arquitectura del dataset.

El dataset está compuesto por imágenes (muestras) de 4 diferentes clases, las cuales son: bananas , manzanas, peras y uvas. Cada una de las muestras tiene una dimensión de 300 x 300 píxeles.

La imagen 3 contiene algunos ejemplos que contiene cada clase.

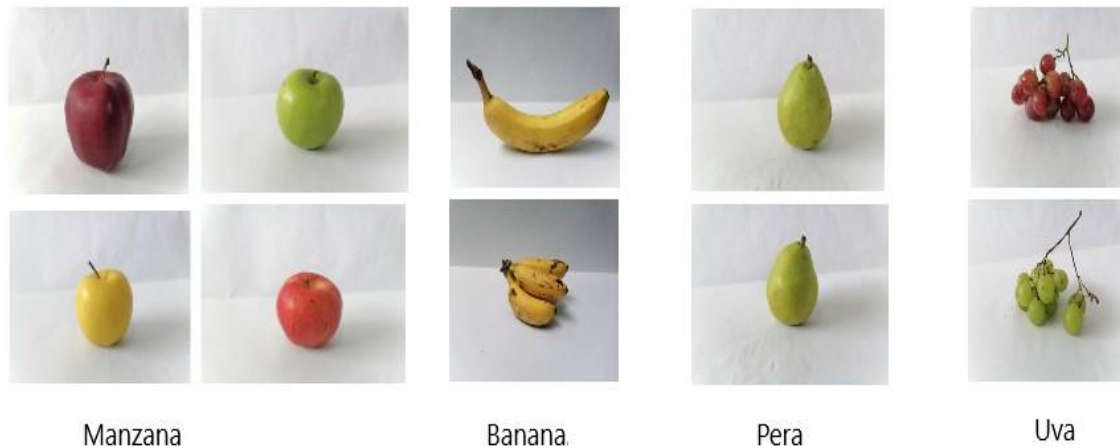


Imagen 2. Ejemplo de imágenes dataset.

La cantidad de elementos que contiene cada clase, y su participación en datos para el entrenamiento , validación y pruebas, se indican en la tabla 1.

	Cantidad de muestras (imágenes)			
Clase	Entrenamiento	Validación	Pruebas	Total
Banana	40	40	6	86
Manzana	136	130	10	276
Pera	67	60	4	131
Uva	64	57	5	129
Total	307	287	25	622

Tabla 1. Cantidad de imágenes.

Algunas de las instancias del subconjunto de entrenamiento, se obtuvieron mediante búsquedas realizadas en el navegador de imágenes de Google.

4. Entrenamiento.

Se realizaron dos entrenamientos con diferentes métricas cada uno, esto con la finalidad de poder obtener diversos resultados y a la vez, validar cuál de los dos era el más óptimo.

1. Entrenamiento 1.

307 muestras, cada una de dimensiones de 300x300.

Epoch = 50

batch_size = 16

steps_per_epoch = muestras // batch_size = 19

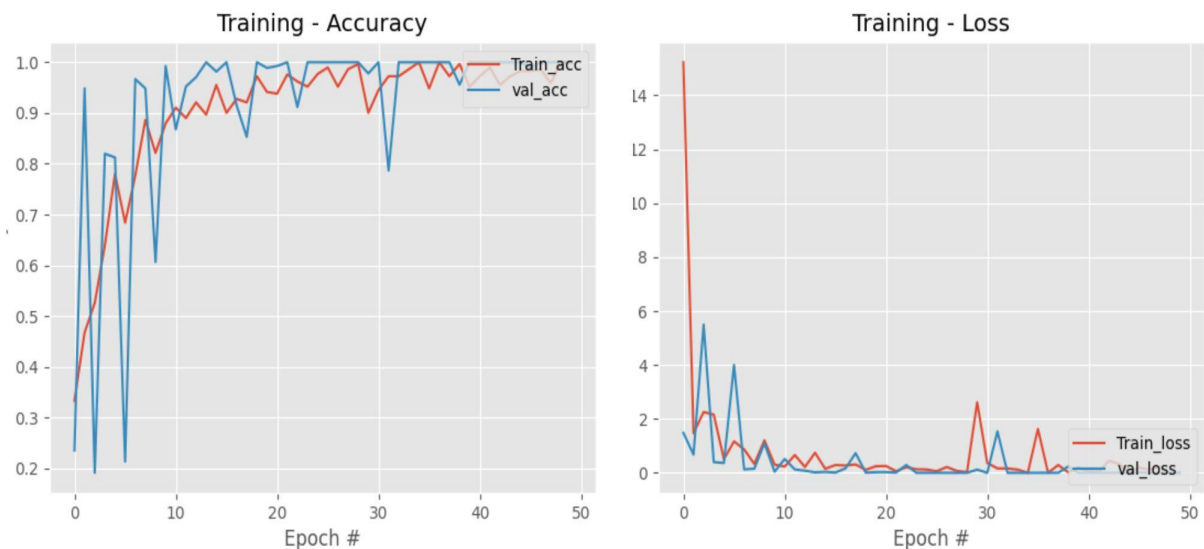
dropout = 0.25

```
Epoch 39/50
19/19 [=====] - 17s 907ms/step - loss: 0.0166 - accuracy: 0.9966 - val_loss: 0.2344 - val_accuracy: 0.9559
Epoch 40/50
19/19 [=====] - 17s 869ms/step - loss: 0.1948 - accuracy: 0.9519 - val_loss: 2.5341e-05 - val_accuracy: 1.0000
Epoch 41/50
19/19 [=====] - 17s 902ms/step - loss: 0.0472 - accuracy: 0.9725 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 42/50
19/19 [=====] - 17s 908ms/step - loss: 0.0301 - accuracy: 0.9897 - val_loss: 1.3167e-04 - val_accuracy: 1.0000
Epoch 43/50
19/19 [=====] - 16s 865ms/step - loss: 0.4425 - accuracy: 0.9553 - val_loss: 2.7885e-06 - val_accuracy: 1.0000
Epoch 44/50
19/19 [=====] - 17s 878ms/step - loss: 0.3288 - accuracy: 0.9725 - val_loss: 2.7906e-04 - val_accuracy: 1.0000
Epoch 45/50
19/19 [=====] - 17s 902ms/step - loss: 0.0319 - accuracy: 0.9828 - val_loss: 7.0096e-06 - val_accuracy: 1.0000
Epoch 46/50
19/19 [=====] - 17s 929ms/step - loss: 0.1863 - accuracy: 0.9828 - val_loss: 0.0130 - val_accuracy: 0.9890
Epoch 47/50
19/19 [=====] - 17s 869ms/step - loss: 0.1195 - accuracy: 0.9863 - val_loss: 1.1331e-05 - val_accuracy: 1.0000
Epoch 48/50
19/19 [=====] - 17s 903ms/step - loss: 0.2252 - accuracy: 0.9588 - val_loss: 5.2968e-06 - val_accuracy: 1.0000
Epoch 49/50
19/19 [=====] - 17s 903ms/step - loss: 2.9689e-04 - accuracy: 1.0000 - val_loss: 3.4891e-06 - val_accuracy: 1.0000
Epoch 50/50
19/19 [=====] - 17s 879ms/step - loss: 6.1443e-04 - accuracy: 1.0000 - val_loss: 6.8649e-06 - val_accuracy: 1.0000
Tiempo de procesamiento (secs): 826.5048360824585
```

Imagen 3. Proceso entrenamiento 1.

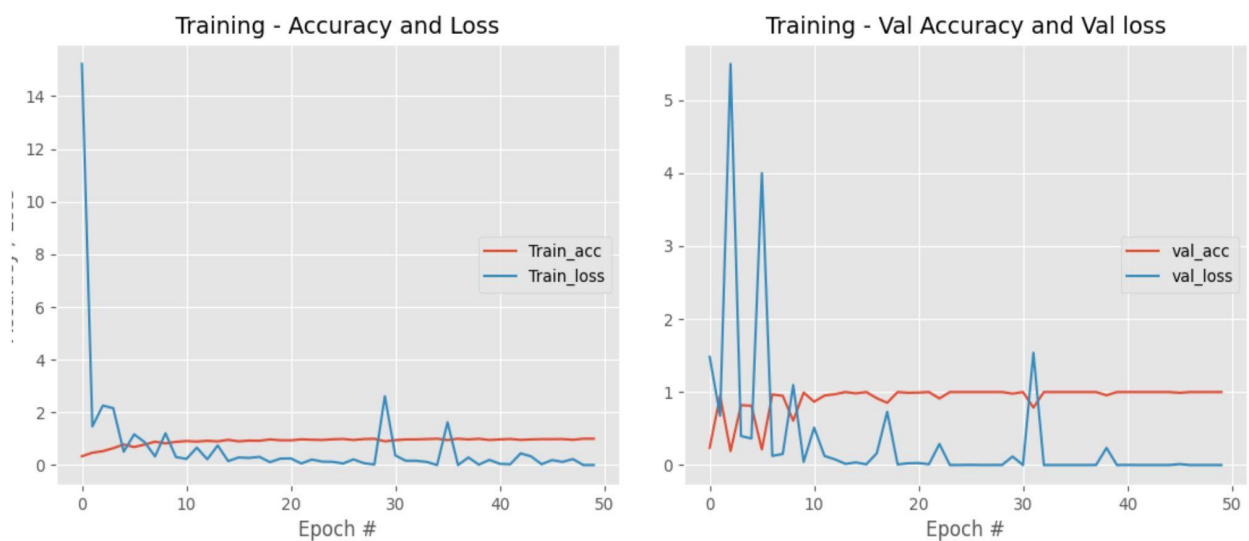
La siguiente imagen muestra el proceso de entrenamiento, así como el tiempo de ejecución que tomó para completar dicho entrenamiento. El cual fue de 826.5048360824585 segundos, lo que equivale a 13 minutos.

La siguiente gráfica muestra el resultado con las imágenes de entrenamiento y validación. La gráfica de la izquierda muestra el aprendizaje o la tasa de aciertos, y la derecha muestra las pérdidas. Ambos resultados son al final de 50 épocas.



Gráfica 1. Resultados obtenidos en entrenamiento 1 .

La gráfica muestra más a detalle los resultados obtenidos. A la izquierda se tiene los resultados de ganancias y pérdidas en imágenes de entrenamiento. A la derecha están los resultados de las imágenes de validación.



Gráfica 2. Accuracy and loss entrenamiento 1 .

2. Entrenamiento 2.

307 muestras, cada una de dimensiones de 300x300.

Epoch = 50

batch_size = 8

steps_per_epoch = muestras // batch_size = 38

dropout = 0.50

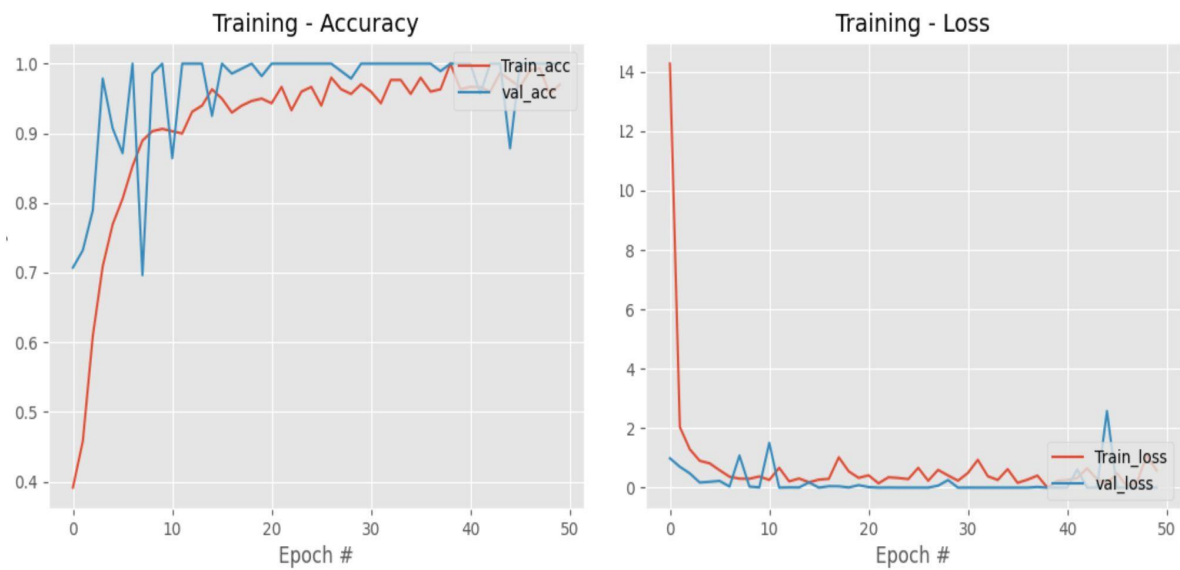
```
Epoch 43/50
38/38 [=====] - 22s 567ms/step - loss: 0.6503 - accuracy: 0.9599 - val_loss: 0.0000e+00 - val_ac
curacy: 1.0000
Epoch 44/50
38/38 [=====] - 22s 585ms/step - loss: 0.2834 - accuracy: 0.9866 - val_loss: 3.1020e-04 - val_ac
curacy: 1.0000
Epoch 45/50
38/38 [=====] - 22s 591ms/step - loss: 0.1251 - accuracy: 0.9766 - val_loss: 2.5776 - val_accu
cy: 0.8786
Epoch 46/50
38/38 [=====] - 22s 591ms/step - loss: 0.4855 - accuracy: 0.9666 - val_loss: 3.9815e-06 - val_ac
curacy: 1.0000
Epoch 47/50
38/38 [=====] - 21s 560ms/step - loss: 0.0502 - accuracy: 0.9901 - val_loss: 0.0000e+00 - val_ac
curacy: 1.0000
Epoch 48/50
38/38 [=====] - 22s 577ms/step - loss: 0.1522 - accuracy: 0.9933 - val_loss: 1.8323e-06 - val_ac
curacy: 1.0000
Epoch 49/50
38/38 [=====] - 22s 575ms/step - loss: 1.0801 - accuracy: 0.9565 - val_loss: 0.0000e+00 - val_ac
curacy: 1.0000
Epoch 50/50
38/38 [=====] - 21s 561ms/step - loss: 0.5774 - accuracy: 0.9699 - val_loss: 1.1495e-08 - val_ac
curacy: 1.0000
Tiempo de procesamiento (secs): 1063.7470407485962
```

Imagen 4. Proceso entrenamiento 2.

Para el segundo entrenamiento, se realizaron algunas modificaciones. La primera fue reducir el tamaño del batch_size, esto con la intención de que los bloques que pasan por épocas, fueran mayores y la red tuviera más muestras para el aprendizaje. Y la segunda fue aumentar el dropout en 0.50, esto quiere decir que apagará el 50% de las neuronas. Esto se realiza con la finalidad de que el sistema aprenda diferentes rutas.

El tiempo de ejecución de este entrenamiento fue de 1063.7470407485962 , equivalente a 17 minutos.

La siguiente gráfica muestra el resultado con las imágenes de entrenamiento y validación. Estos resultados fueron obtenidos después de 50 épocas. A la izquierda se tiene los resultados en imágenes de entrenamiento. A la derecha están los resultados de las imágenes de validación. Ambos muestran los resultados de ganancias y pérdidas.



Gráfica 3. Resultados obtenidos en entrenamiento 2.

La gráfica muestra más a detalle los resultados obtenidos. A la izquierda se tiene los resultados de ganancias y pérdidas en imágenes de entrenamiento. A la derecha están los resultados de las imágenes de validación.



Gráfica 4. Accuracy and loss entrenamiento 2.

3. Entrenamiento 3.

307 muestras, cada una de dimensiones de 300x300.

Epoch = 50

batch_size = 1

steps_per_epoch = muestras // batch_size = 307

dropout = 0.50

Para el último entrenamiento, se utilizó un batch_size de 1, es decir, que por cada época pasarían todas las muestras que se tienen. Las demás métricas se siguen manteniendo.

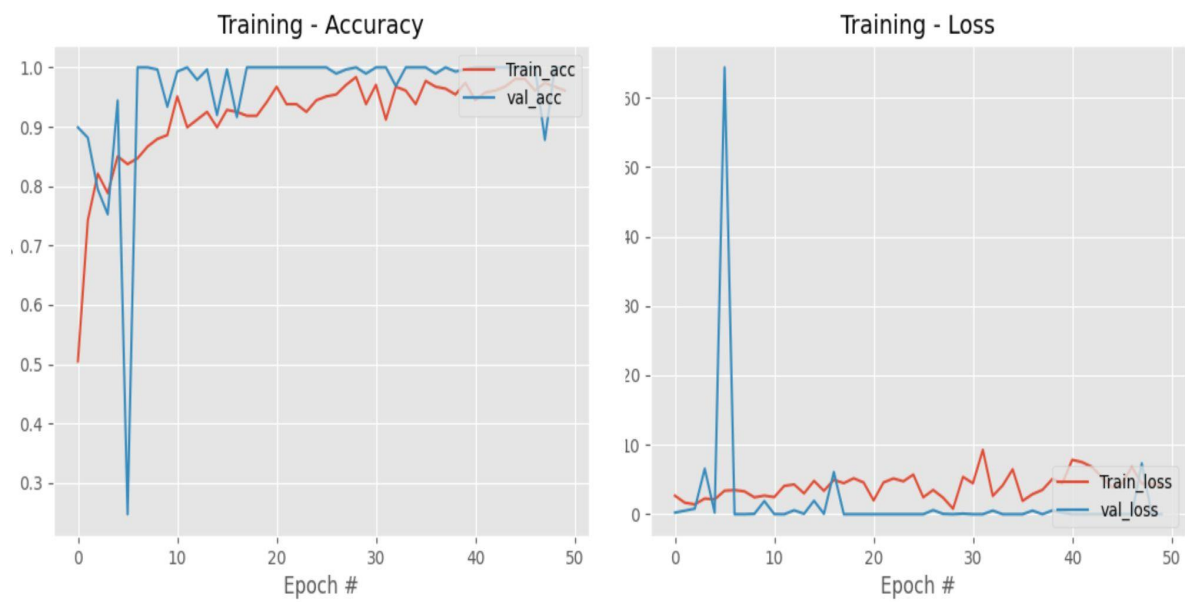
```
307/307 [=====] - 97s 315ms/step - loss: 7.4940 - accuracy: 0.9577 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 43/50
307/307 [=====] - 96s 313ms/step - loss: 6.7987 - accuracy: 0.9609 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 44/50
307/307 [=====] - 102s 331ms/step - loss: 5.2994 - accuracy: 0.9674 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 45/50
307/307 [=====] - 94s 305ms/step - loss: 3.8895 - accuracy: 0.9805 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 46/50
307/307 [=====] - 96s 312ms/step - loss: 4.3805 - accuracy: 0.9805 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 47/50
307/307 [=====] - 95s 310ms/step - loss: 6.8181 - accuracy: 0.9609 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 48/50
307/307 [=====] - 94s 306ms/step - loss: 4.4071 - accuracy: 0.9739 - val_loss: 7.3251 - val_accu
racy: 0.8780
Epoch 49/50
307/307 [=====] - 94s 306ms/step - loss: 4.4178 - accuracy: 0.9674 - val_loss: 0.0000e+00 - val_
accuracy: 1.0000
Epoch 50/50
307/307 [=====] - 97s 315ms/step - loss: 4.1307 - accuracy: 0.9609 - val_loss: 3.7962e-07 - val_
accuracy: 1.0000
Tiempo de procesamiento (secs): 5036.985951185226
```

Imagen 5. Proceso entrenamiento 3.

Al tener más muestras en cada bloque, el tiempo de ejecución de este entrenamiento fue de 5036.985951185226 , equivalente a 83 minutos.

La siguiente gráfica muestra los resultados obtenidos después del entrenamiento.

Al igual que las gráficas anteriores a la izquierda se tiene los resultados de ganancias y pérdidas en imágenes de entrenamiento. A la derecha están los resultados de las imágenes de validación.



Gráfica 5. Accuracy and loss entrenamiento 3.

En la siguiente gráfica se pueden observar los resultados más a detalle.



Gráfica 6. Resultados obtenidos en entrenamiento 3.

Resultados.

Después del entrenamiento, se realizó un test para comprobar el funcionamiento de la red. De los entrenamientos realizados, los resultados obtenidos fueron los siguientes.

1. Entrenamiento 1.

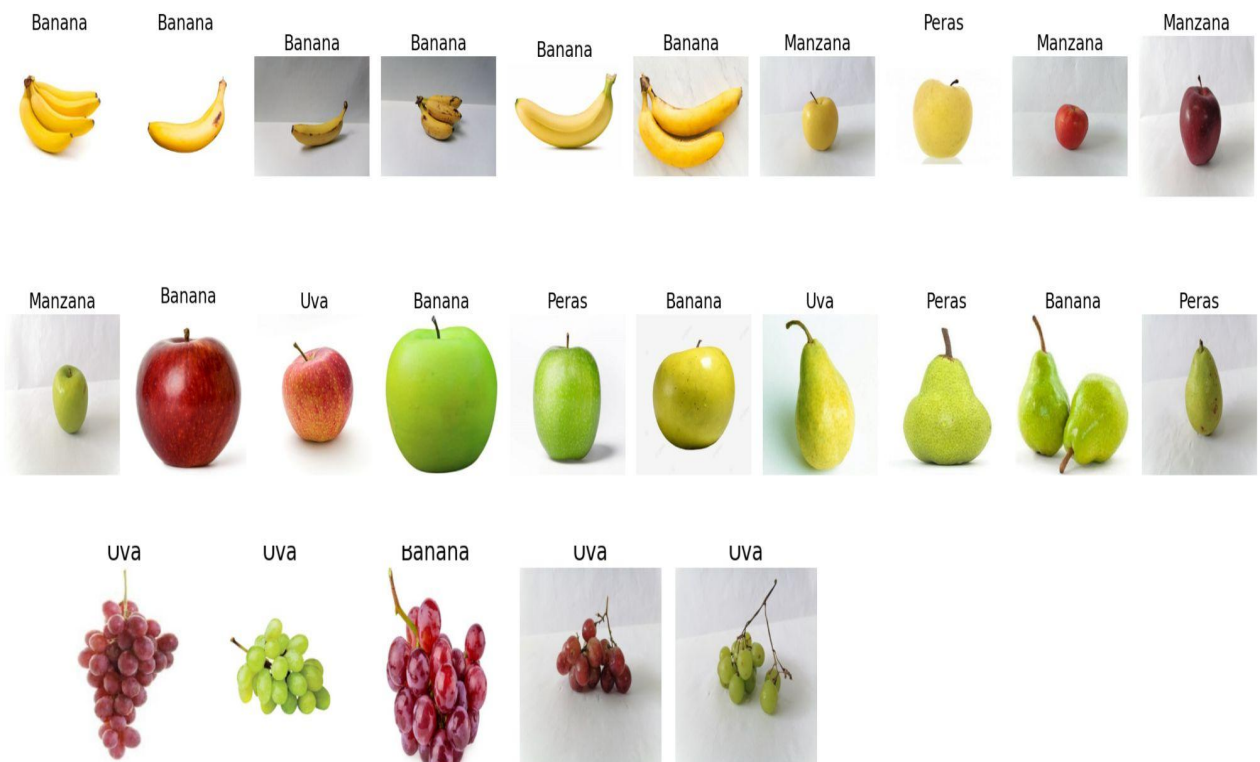


Imagen 6. Resultados entrenamiento 1.

Para una mejor comprensión, los resultados serán evaluados a través de una matriz de confusión.

RED	Real				
		Bananas	Manzanas	Peras	Uvas
	Bananas	6	3	1	1
	Manzanas	0	4	0	0
	Peras	0	2	2	0
	Uvas	0	1	1	4

Tabla 2. Matriz de confusión entrenamiento 1.

2. Entrenamiento 2.

Para el entrenamiento 2, los resultados fueron los siguientes:

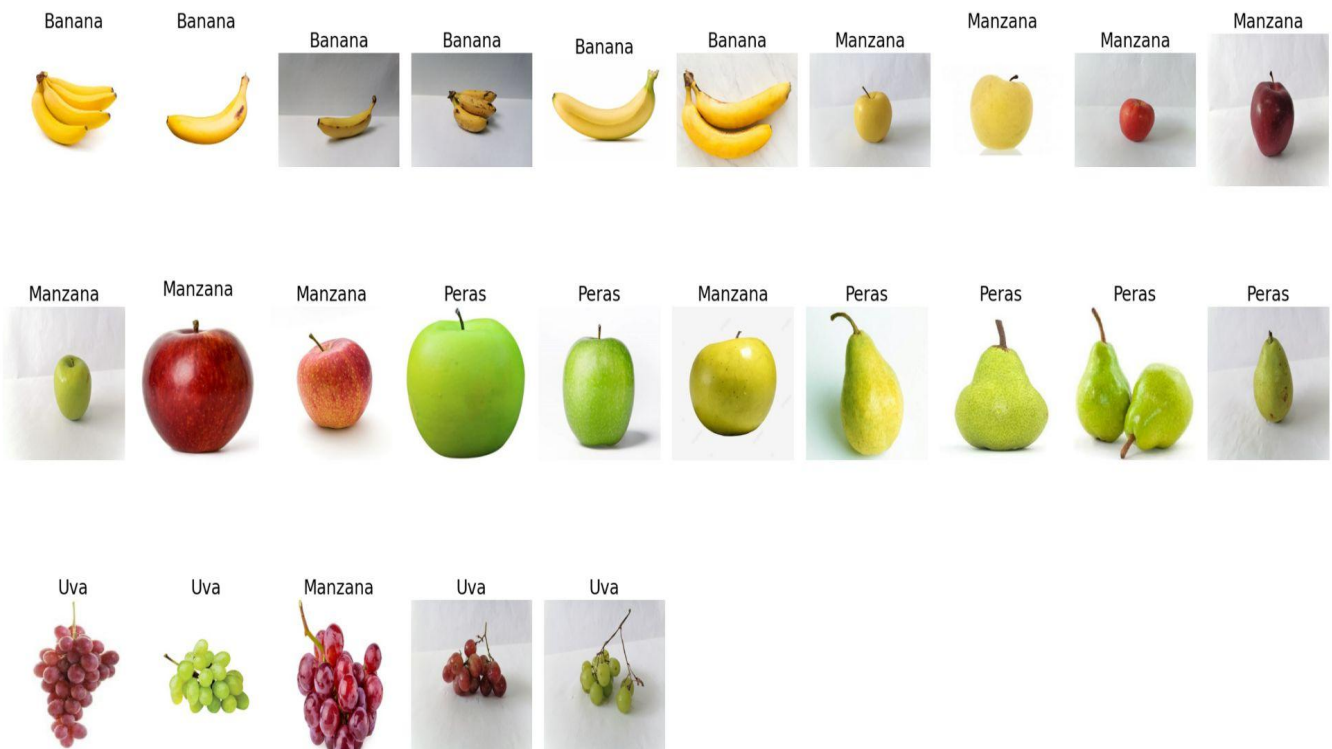


Imagen 7. Resultados entrenamiento 2.

Matriz de confusión del entrenamiento 2.

RED	Real				
		Bananas	Manzanas	Peras	Uvas
	Bananas	6	0	0	0
	Manzanas	0	8	2	0
	Peras	0	0	4	0
	Uvas	0	1	0	4

Tabla 3. Matriz de confusión entrenamiento 2.

3. Entrenamiento 3.

Resultados obtenidos durante el entrenamiento 3.

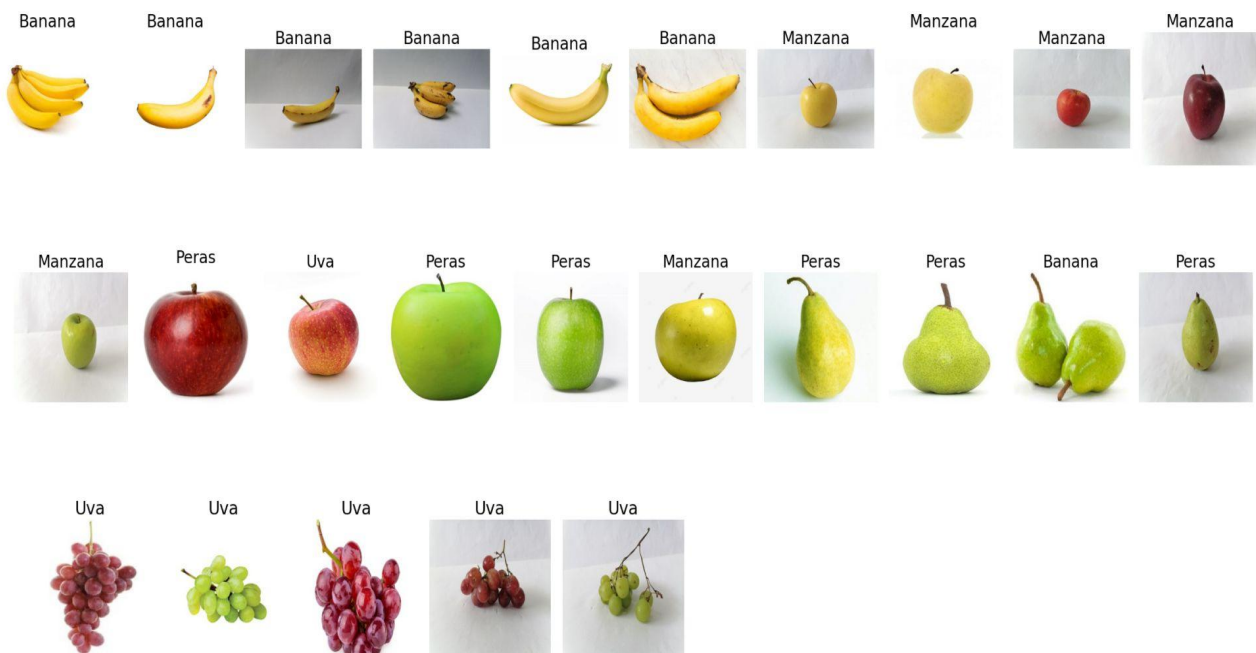


Imagen 8. Resultados entrenamiento 3.

Matriz de confusión del entrenamiento 3.

RED	Real				
		Bananas	Manzanas	Peras	Uvas
	Bananas	6	3	1	0
	Manzanas	0	6	0	0
	Peras	0	3	3	0
	Uvas	0	1	0	5

Tabla 4. Matriz de confusión entrenamiento 2.

Discusiones.

1era. Discusión.

17 de Julio del 2021. Se empezó a recopilar los objetos necesarios para cada una de las clases. En esta misma fecha, se comenzó a tomar las fotografías correspondientes para el entrenamiento de la red.

2da. Discusión.

19 de Julio del 2021. Se aprobó la estructura del dataset, y se complementa con las imágenes faltantes para cada una de las clases.

3ra. Discusión.

21 de Julio del 2021. Se solucionan problemas con un parámetro llamado “steps_per_epoch”. Se realiza el cálculo correspondiente para este parámetro y el problema presentado se soluciona.

4ta. Discusión.

22 de Julio del 2021. Se comienzan con las pruebas para comprobar que el desarrollo de la red y el entrenamiento estuvieran funcionando. A la vez, se van ajustando los parámetros para encontrar un buen funcionamiento.

5ta. Discusión.

23 de Julio del 2021. Se agregan más imágenes de manzanas verdes, ya que estas constantemente estaban siendo confundidas con peras.

Análisis y conclusiones.

Es importante mencionar que las redes neuronales se sustentan principalmente en la experiencia, es decir, que necesita visualizar, aprender diseños y patrones de ciertas clases, para ir aprendiendo de lo que se trata.

Durante el desarrollo de esta aplicación de una red neuronal, se obtuvieron conocimientos acerca del funcionamiento, estructura, configuración. Así como conocimientos en librerías que están diseñadas para el desarrollo de redes neuronales, las cuales nos simplifican los procesos.

Esta aplicación tiene como objetivo el reconocimiento de 4 clases de frutas, las cuales son: bananas, manzanas, peras y uvas. Pero claramente, esta aplicación podría ser diseñada para tareas de más importancia en campos importantes como la educación, medicina, entre otros.

Referencias

1. *Article title:* *Welcome to Python.org*
Website title: *Python.org*
URL: <https://www.python.org/>
2. *Article title:* *TensorFlow*
Website title: *TensorFlow*
URL: <https://www.tensorflow.org/>
3. *Author* *Keras Team*
Article title: *Keras: the Python deep learning API*
Website title: *Keras.io*
URL: <https://keras.io/>
4. *Article title:* *Matplotlib: Python plotting — Matplotlib 3.4.2 documentation*
Website title: *Matplotlib.org*
URL: <https://matplotlib.org/>
5. *Article title:* *NumPy*
Website title: *Numpy.org*
URL: <https://numpy.org/>
6. *Article title:* *pandas - Python Data Analysis Library*
Website title: *Pandas.pydata.org*
URL: <https://pandas.pydata.org/>