Universidad
Rafael Landívar

Proyecto Inteligencia Artificial 2019

Determinación del estado de Maduración del Durazno

Estudiantes:

Carlos Enrique Menchú Sapón 15387-15
Ismar Alexander Morales Miranda 16371-15

Índice

Características del Durazno

La fruta escogida fue el durazno, esta presenta varias fases de maduración, de las cuales se escogieron las siguientes para que la red pueda identificarlas:

Fases de Maduración	
Durazno Verde	
Durazno Maduro	
Durazno Sobre Maduro	
Durazno Podrido Fase 1	
Durazno Podrido Fase 2	

Calidad del Durazno

La calidad de un durazno se determina según su tamaño, para ello se debe medir el tamaño de su diámetro ecuatorial. Según el diámetro así será su clasificación. El rango de diámetros es el siguiente:

Categoría	Diámetro ecuatorial (cm)
Extra	>7
Primera	6 – 7
Segunda	5 – 6
Corriente	< 5

Marco Teórico

Funcionamiento de las Redes Neuronales convolucionales

Para el proyecto se utilizó una red neuronal convolucional, el modo en que funciona es el siguiente:

Tratamiento de las imágenes:

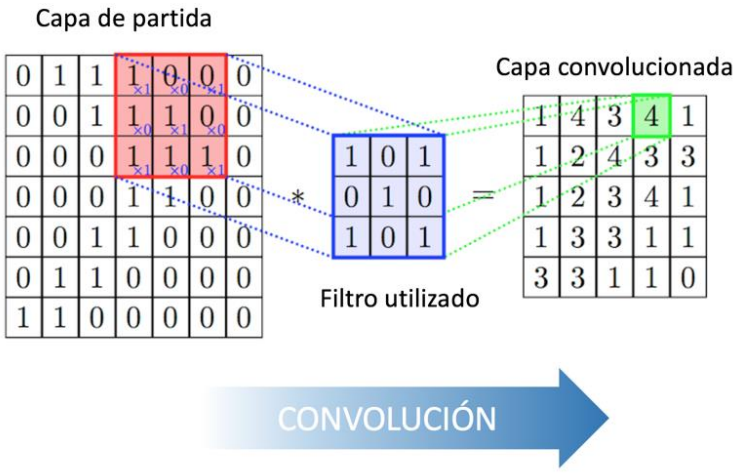
Las redes convolucionales utilizan la siguiente estructura de capas:

- Convolucion
- Pooling
- Capa totalmente conectada

Capa de convolucion

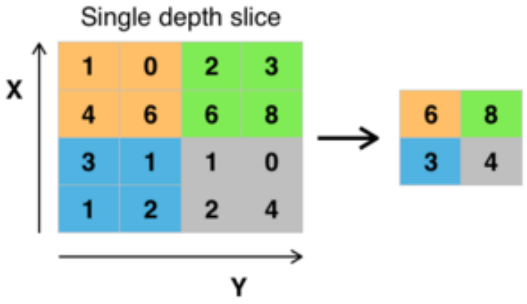
Estas consisten en tomar "grupos de pixeles cercanos" de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz que se llama kernel o filtro. El kernel "recorre" todas las neuronas de entrada (de izquierda-derecha, de arriba-abajo) y genera una nueva matriz de salida, de esta forma logramos reducir el tamaño de los parámetros.

- Tamaño del filtro: Tamaño del parche que se va a mover por la imagen (altura y longitud)
- Profundidad de la capa convolucional: agrandes rasgos es el número de filtros que se le van a aplicar a la imagen en cada una de las convoluciones, mientras más filtros se pasen se identifican formas más elaboradas.
- Paso: Que tanto se va a ir recorriendo el filtro a lo largo de la imagen, mientras más alto sea el paso se va a reducir más el tamaño de la imagen.



Capa de Pooling

La capa de reducción o pooling se coloca generalmente después de la capa convolucional. Su utilidad principal radica en la reducción de las dimensiones espaciales (ancho x alto) del volumen de entrada para la siguiente capa convolucional. No afecta a la dimensión de profundidad del volumen.



La operación que se suele utilizar en esta capa es max-pooling, que divide a la imagen de entrada en un conjunto de rectángulos y, respecto de cada subregión, se va quedando con el máximo valor.

Capa totalmente conectada:

La última capa de esta red es una capa clasificadora que tendrá tantas neuronas como el número de clases a predecir, es la encargada de predecir y determinar un resultado con la información que se le pasa de las capas anteriores.

Matemáticas de las redes convolucionales:

Las redes convolucionales al igual que las redes normales utiliza el algoritmo de backpropagation para propagar el error a las demás capa, solo que lo hace una variación del algoritmo que utilizan las redes normales.

Capa de Convolucion

Dada una imagen de entrada I y un filtro o kernel K de dimensiones $k_1 x k_2$ la operación de convolucion está dada por:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n) K(m,n)$$
$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(-m,-n)$$

Las redes neuronales convolucionales constan de capas convolucionales que se caracterizan por un mapa de entrada I un banco de filtros K y sesgos b.

En el caso de las imágenes, podríamos tener como entrada una imagen con altura H, anchura W y $C = 3$ canales (rojo, azul y verde) tal que $I \in \mathbb{R}^{H*W*C}$. Después para un banco de D filtros que tenemos $K \in \mathbb{R}^{K_1*K_2*C*D}$ y sesgos $b \in \mathbb{R}^{D}$, uno para cada filtro.

La salida de este procedimiento de convolución es la siguiente:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \sum_{c=1}^{C} K_{m,n,c} * I_{i+m,j+n,c+b}$$

La operación de convolución llevada a cabo aquí es la misma que la correlación cruzada, excepto que el núcleo está "volteado" (horizontal y verticalmente).

Notación:

Para ayudarnos a explorar la propagación hacia adelante y hacia atrás, haremos uso de la siguiente notación:

- l es el l^{th} capa donde $l = 1$ es la primera capa y $l = L$ es la última capa.
- Entrada x es de dimensión $H * W$ y tiene i y j como los iteradores.
- El filtro o kernel w es de dimensión $k_1 x k_2$ y tiene m y n como los iteradores.
- $w_{m,n}^{l}$ es la matriz de peso que conecta las neuronas de la capa l con las neuronas de la capa $l-1$
- b^l es la unidad de sesgo en la capa l
- $x_{i,j}^{l}$ es el vector de entrada convuelto en la capa l más el sesgo representado como:

$$x_{i,j}^{l} = \sum_m \sum_n w_{m,n}^{l} * o_{i+m,j+n}^{l-1} + b^l$$

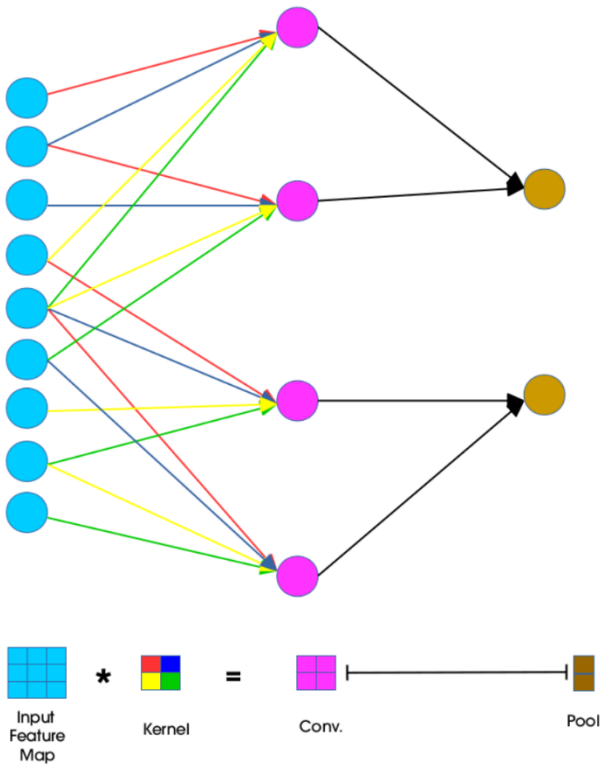
- $o_{i,j}^{l}$ es el vector de salida de la capa l dada por:

$$o_{i,j}^{l} = f\left(x_{i,j}^{l}\right)$$

- $f(\cdot)$ es la función de activación. Aplicación de la capa de activación al vector de entrada convuelto en la capa l es dado por $f\left(x_{i,j}^{l}\right)$

Propagación hacia adelante

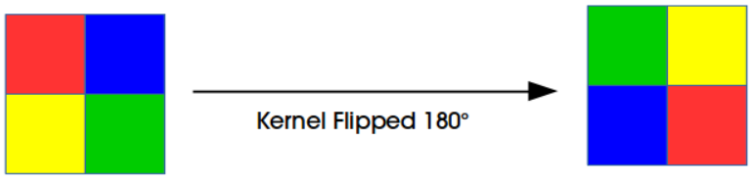
Para realizar una operación de convolución, el núcleo se voltea 180 grados. Este procedimiento se repite utilizando diferentes núcleos para formar tantos mapas de características de salida como se desee. El concepto de reparto de peso se utiliza como se muestra en el siguiente diagrama:



Las unidades en la capa convolucional ilustrada arriba tienen campos receptivos de tamaño 4 en el mapa de características de entrada y, por lo tanto, solo están conectadas a 4 neuronas adyacentes en la capa de entrada. Esta es la idea de la escasa conectividad en las CNN donde existe un patrón de conectividad local entre las neuronas en las capas adyacentes.

Los códigos de color de los pesos que unen la capa de entrada a la capa convolucional muestran cómo se distribuyen (comparten) los pesos del núcleo entre las neuronas en las capas adyacentes. Los pesos del mismo color están obligados a ser idénticos.

El proceso de convolución aquí se expresa generalmente como una correlación cruzada pero con un núcleo invertido. En el diagrama a continuación, ilustramos un núcleo que se ha volteado horizontal y verticalmente:



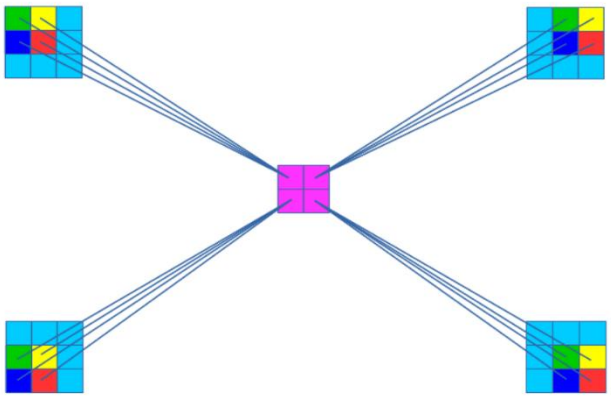
La ecuación de convolución de la entrada en la capa l

$$x_{i,j}^{l} = rot_{180^\circ}\{w_{m,n}^{l}\} * o_{i,j}^{l-1} + b_{i,j}^{l} \qquad ec.(1)$$

$$x_{i,j}^{l} = \sum_{m}\sum_{n} w_{m,n}^{l} * o_{i+m,j+n}^{l-1} + b_{i,j}^{l} \qquad ec.(2)$$

$$o_{i,j}^{l} = f\left(x_{i,j}^{l}\right) \qquad ec.(3)$$

Esto se ilustra a continuación:



Error

Para un total de $Py_p t_p$

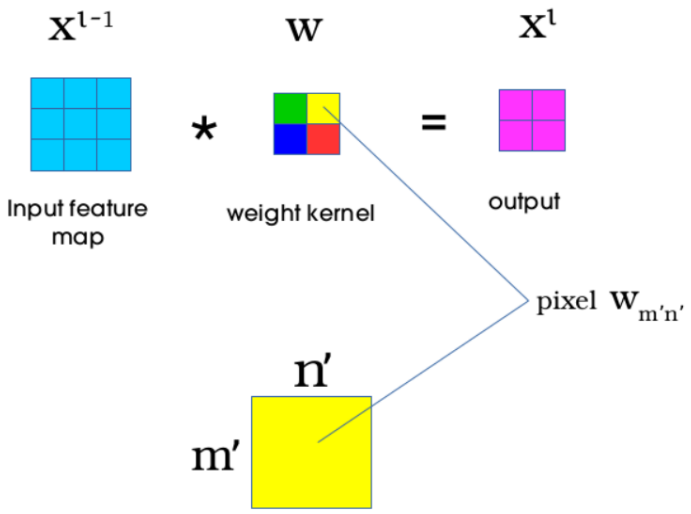
$$E = \frac{1}{2}\sum_{p}\left(t_p - y_p\right)^2 \qquad ec.(4)$$

El aprendizaje se logrará ajustando los pesos de modo que $y_p t_p E$

Propagación hacia atrás (Backpropagation)

Para backpropagation hay dos actualizaciones realizadas, para los pesos y los deltas. Vamos a empezar con la actualización de peso.

Buscamos para calcular $\frac{\partial E}{\partial w^l_{m',n'}} w_{m',n'} E$



Durante la propagación hacia adelante, la operación de convolución garantiza que el píxel amarillo

$$w_{m',n'} w_{m',n'}$$

Convolución entre el mapa de características de entrada de la dimensión:

$$H \times W k_1 \times k_2 (H - k_1 + 1)(W - k_2 + 1)$$

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i,j}} * \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \qquad ec.(5)$$

En la eq. (5), $x^l_{i,j} \sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l$

$$\frac{\partial E}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}}\left(\sum_m \sum_n w^l_{m,n} * o^{l-1}_{i+m,j+n} + b^l_{i,j} \right) \qquad ec.(6)$$

Ampliando aún más las sumas en la eq. (6), $m = m'\, n = n' w^l_{m,n} o^{l-1}_{i+m,j+n}$

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}}\left(w^l_{0,0} * o^{l-1}_{i+0,j+0} + \cdots + w^l_{m',n'} o^{l-1}_{i+m',j+n'} + \cdots + b^l\right)$$

$$= \frac{\partial}{\partial w^l_{m',n'}}\left(w^l_{m',n'} o^{l-1}_{i+m',j+n'}\right)$$

$$= o^{l-1}_{i+m',j+n'} \qquad ec.(7)$$

Sustituyendo eq. (7) en eq. (5) nos da los siguientes resultados:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} * o^{l-1}_{i+m',j+n'} \qquad ec.(8)$$

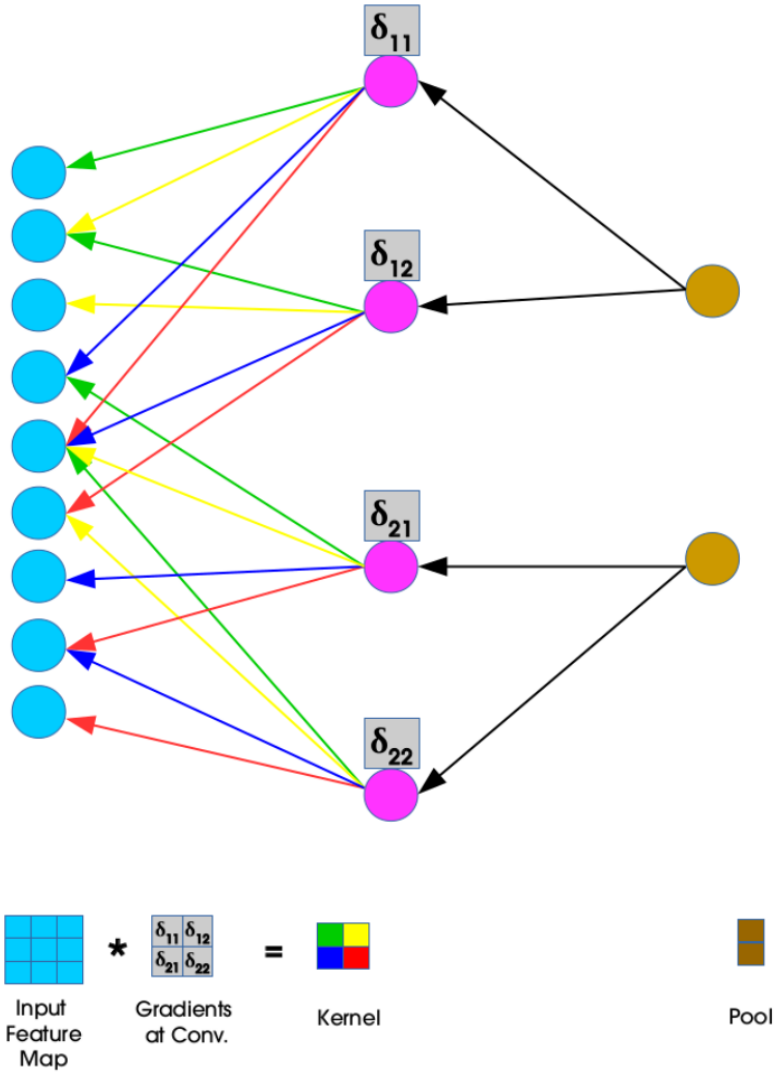
$$= rot180^\circ \left\{\delta^l_{i,j}\right\} * o^{l-1}_{m',n'} \qquad ec.(9)$$

La suma dual en la ec. 8 es el resultado de compartir el peso en la red (el mismo peso del núcleo se desliza sobre todo el mapa de características de entrada durante la convolución). Las sumas representan una colección de todos los gradientes $\delta^l_{i,j}$ provenientes de todas las salidas en la capa l.

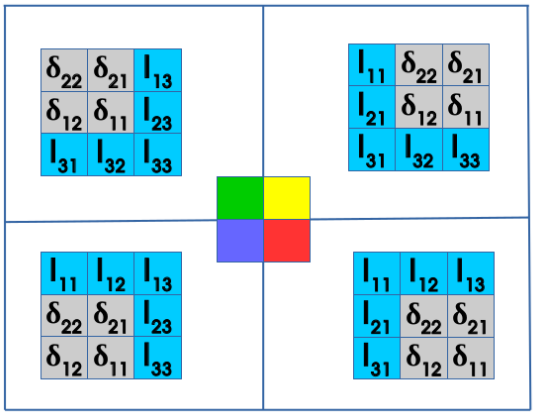
Al obtener gradientes de los mapas de filtros, tenemos una correlación cruzada que se transforma en una convolución al "voltear" la matriz delta $\delta^l_{i,j}$ (horizontal y verticalmente) de la misma manera que movimos los filtros durante la propagación hacia adelante.

A continuación se muestra una ilustración de la matriz delta volteada:

El siguiente diagrama muestra los gradientes ($\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}$) generados durante la propagación hacia atrás:



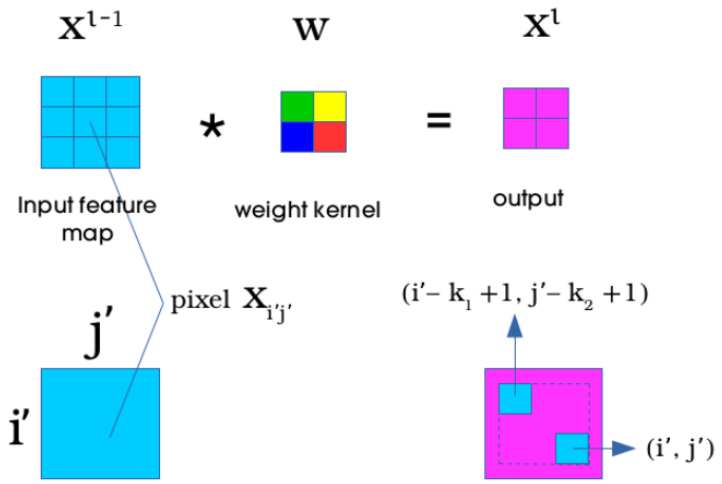
La operación de convolución utilizada para obtener el nuevo conjunto de pesos. Como se muestra a continuación:



Durante el proceso de reconstrucción, se utilizan los deltas ($\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}$) Estos deltas son proporcionados por una ecuación de la forma:

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}} \qquad ec.(10)$$

En este punto, estamos buscando calcular $\frac{\partial E}{\partial x_{i',j'}^{l}}$, que puede interpretarse como la medida de cómo el cambio en un solo píxel $x_{i',j'}^{l}$ en el mapa de características de entrada afecta a la función de pérdida E.



En el diagrama anterior, se puede ver que la región en la salida afectada por el píxel $x^l_{i',j'}$ desde la entrada es la región en la salida delimitada por las líneas discontinuas donde se encuentra el píxel de la esquina superior izquierda $(i'-k_1+1, j'-k_2+1)$ y el píxel de la esquina inferior derecha está dado por i', j'.

El uso de la regla de la cadena y la introducción de sumas nos da la siguiente ecuación

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{i,j\in Q} \frac{\partial E}{\partial x^{l+1}_Q} * \frac{\partial x^{l+1}_Q}{\partial x^l_{i',j'}}$$

$$= \sum_{i,j\in Q} \delta^{l+1}_Q * \frac{\partial x^{l+1}_Q}{\partial x^l_{i',j'}} \qquad ec.(11)$$

Q en la suma anterior representa la región de salida limitada por líneas discontinuas y está compuesta por píxeles en la salida que se ven afectados por el píxel único $x_{i',j'}$ en el mapa de características de entrada. Una forma más formal de representar. La ec. (11) es:

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial E}{\partial x^{l+1}_{i'-m,j'-n}} * \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}}$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} * \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \qquad ec.(12)$$

En la región Q , la altura varía de $i'-0$ a $i'-(k_1-1)$ y el ancho $j'-0$ a $j'-(k_2-1)$ Estos dos pueden representarse simplemente por $i'-m$ y $j'-n$ en la suma, ya que los iteradores m y n existen en los siguientes rangos similares e 0≤m≤k1−1 y 0≤n≤k2−1.

En la ec. (12), $x^{l+1}_{i'-m,j'-n}$ es equivalente a

$$\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} o^l_{i'-m+m',j'-n+n'} + b^{l+1}$$

Y expandir esta parte de la ecuación nos da:

$$\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} = \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} * o^l_{i-m+m',j-n+n'} + b^{l+1} \right)$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}}\left(\sum_{m'}\sum_{n'} w_{m',n'}^{l+1} * f\left(x_{i-m+m',j-n+n'}^{l}\right) + b^{l+1}\right) \qquad ec.(13)$$

Ampliando aún más la suma en la ec.(12) y tomando las derivadas parciales para todos los componentes, se obtienen valores cero para todos, excepto los componentes donde $m' = m\ y\ n' = n$ así que eso $f\left(x_{i-m+m',j-n+n'}^{l}\right)$ se convierte en $f\left(x_{i',j'}^{l}\right)$ y $w_{m',n'}^{l+1}$ se convierte en $w_{m,n}^{l+1}$ en la parte relevante de la suma ampliada de la siguiente manera:

$$\frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^{l}} = \frac{\partial}{\partial x_{i',j'}^{l}}\left(w_{m',n'}^{l+1} * f\left(x_{0-m+m',0-n+n'}^{l}\right) + \cdots + w_{m,n}^{l+1} + \cdots + b^{l+1}\right)$$

$$= \frac{\partial}{\partial x_{i',j'}^{l}}\left(w_{m,n}^{l+1} * f\left(x_{i',j'}^{l}\right)\right)$$

$$= w_{m,n}^{l+1}\frac{\partial}{\partial x_{i',j'}^{l}}\left(f\left(x_{i',j'}^{l}\right)\right)$$

$$= w_{m,n}^{l+1} f'\left(x_{i',j'}^{l}\right) \qquad ec.(14)$$

Sustituyendo la ec. (14) en la ec.(12) nos da los siguientes resultados:

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} * w_{m,n}^{l+1} f'\left(x_{i',j'}^{l}\right) \qquad ec.(15)$$

Para la propagación hacia atrás, utilizamos el kernel volteado y, como resultado, ahora tendremos una convolución que se expresa como una correlación cruzada con un kernel volteado:

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} * w_{m,n}^{l+1} f'\left(x_{i',j'}^{l}\right)$$

$$= rot180^{\circ}\left\{= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} * w_{m,n}^{l+1}\right\} * f'\left(x_{i',j'}^{l}\right)$$

$$= \delta_{i',j'}^{l+1} * rot180^{\circ}\{w_{m,n}^{l+1}\} * f'\left(x_{i',j'}^{l}\right)$$

Capa de pooling

La función de la capa de agrupación es reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y el cálculo en la red y, por lo tanto, controlar también el sobreajuste. Ningún aprendizaje se lleva a cabo en las capas de agrupación.

Las unidades de agrupación se obtienen mediante funciones como agrupación máxima, agrupación media e incluso agrupación de la norma L2. En la capa de agrupación, la propagación hacia adelante hace que un bloque de agrupación de N × N se reduzca a un solo valor, el valor de la "unidad ganadora". La propagación hacia atrás de la capa de agrupación luego calcula el error que es adquirido por este valor único "unidad ganadora".

Las redes neuronales convolucionales emplean una estrategia de reparto de peso que conduce a una reducción significativa en el número de parámetros que deben aprenderse. La presencia de tamaños de campo receptivo más grandes de neuronas en capas convolucionales sucesivas junto con la presencia de capas de agrupación también conduce a la invariabilidad de la traducción. Como hemos observado, las derivaciones de las propagaciones hacia adelante y hacia atrás diferirán según la capa a través de la cual nos estamos propagando.

Diseño de la aplicación

La aplicación costa de tres etapas importantes las cuales son:

- Determinar si la imagen ingresada por el usuario es de un durazno o no
- Determinar el estado de maduración en que se encuentra el durazno
- Determinar la calidad del durazno

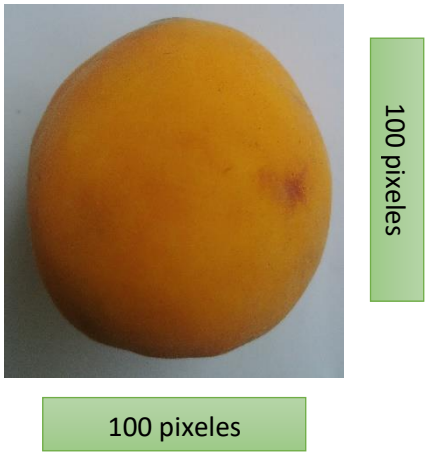
Detección del estado de maduración del durazno

Tratamiento de los datos de entrenamiento

Para que la red neuronal pudiera identificar los estados de maduración del durazno se utilizaron los siguientes datos:

Cada carpeta contiene imágenes con el estado de maduración correspondiente a duraznos maduros, podridos fase 1, podridos fase 2, verdes, sobre maduros.

Los datos se tuvieron que redimensionar a un tamaño de 100x100 pixeles, facilitando el procesamiento computacional que debía hacer la red.



Los datos fueron normalizados, escalándolos a valores en un rango de 0-1, ya que originalmente están en valores de 0 a 255 por los colores. El escalamiento se realizó con el siguiente código:

```
entrenamiento_imagenes = ImageDataGenerator(
    rescale=1./255,
)
```

Su función es agarrar todas las imágenes de entrenamiento y convertir los valores en el rango de 0 a 1. Luego las imágenes son cambiadas de tamaño con el siguiente código:

```
imagen_entrenamiento = entrenamiento_imagenes.flow_from_directory(

    imagenes_entrenamiento,
    target_size=(altura,longitud),
    batch_size=batch_size,
    class_mode='categorical'

)
```

Se indica cual es la ruta donde están las imágenes que van a ser usadas como entrenamiento, luego se les asigna una nueva altura y longitud. A **Class_mode** se le asigna **'categorical'** porque lo que va a realizar el modelo es hasta cierto punto una clasificación de imágenes. Lo que hace categorical es decirle al modelo tiene que clasificar las imágenes según el tipo que sean.
Para saber que índice se le asignó a cada etapa de maduración del durazno se utiliza:

```
imagen_entrenamiento.class_indices
```

Esto imprime que índice tiene asignada cada etapa de maduración del durazno, los índices asignados quedaron de la siguiente manera:

Índice	Etapa de maduración
0	Durazno Maduro
1	Durazno Podrido Fase 1
2	Durazno Podrido Fase 2
3	Durazno Verde
4	Durazno Sobre Maduro

Diseño de la red neuronal

Los parámetros de la red son:

```
altura = 100
longitud = 100
batch_size = 32
pasos = 1500
canales = 3
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3,3)
tamano_filtro2 = (2,2)
tamano_pool = (2,2)
clases = 5
lr = 0.0005
```

- **Canales:** Los canales de la imagen serán 3 por los debido a que las imágenes a analizar están a color por los valores RGB.

- La altura es de 100 pixeles y longitud de 100 pixeles porque a esa será el nuevo tamaño de la imagen.

- **flitrosConv1:** Serán 32 filtros en la primera capa de convolucion para lograr identificar formas o patrones más elaborados con el paso de capa filtro.

- **Tamano_filtro1:** El tamaño del filtro de la primera capa de convolucion será de 3x3 pixeles, se determinó de esta manera para lograr la recolección de la mayor información posible en la imagen.

- **filtrosConv2:** Serán 64 filtros en la segunda capa de convolucion para lograr identificar formas o patrones más elaborados con el paso de capa filtro.

- **Tamano_filtro2:** El tamaño del filtro de la segunda capa de convolucion será de 2x2 pixeles para recolectar la mayor información posible de la imagen que viene de la primera capa de convolucion y así identificar patrones o formas más elaboradas.

- **Lr:** El factor de aprendizaje es de 0.0005 para garantizar que nuestra red va a aprender de la mejor forma y que el error va a ser los más cercano a 0.

- **Clases:** son 5 clases las cuales serán: Duraznos Verdes, Duraznos Maduros, Duraznos Sobre Maduros, Duraznos Podridos Fase 1 y Duraznos Podridos Fase 2 (el índice de cada clase se lo asigna el programa de pendiendo del orden de las carpetas que contienen las imágenes de entrenamiento).

- Se van a usar 1500 pasos para garantizar que nuestra imagen va a ser lo más pequeña posible y así facilitar el costo de cálculos en la computadora.

- **Batch_size:** será de 3.

- **Épocas:** 20 épocas fueron necesarias para entrenar la red neuronal de manera adecuada.

Estructura del modelo de red neuronal:

Funciones de Activación:

- **Capas ocultas:** Relu
- **Capa de salida:** Softmax

Relu: en la activación de las capas ocultas se escogió la función de activación **relu** debido a que con esta función las redes neuronales aprende de manera más rápida y computacionalmente representa un costo de procesamiento no muy grande, además de que dicha función es aconsejable usarla en redes convolucionales específicamente en las capas ocultas.

Softmax: se escogió esa función debido a que devuelve valores entre 0 y 1, debido a lo cual se considera esta función como una distribución de probabilidad categórica, Softmax se encarga de pasar a probabilidad (entre 0 y 1) a las neuronas de salida.

Diseño del modelo:

La red neuronal va a contar con 6 capas en total:

- Primera capa: capara de entrada.
- Segunda capa: capa de pooling.
- Tercera capa: capa de convolución.
- Cuarta capa: capa de pooling.
- Quinta capa: capa densa de con 650 neuronas.
- Sexta capa: capa de clasificación con 5 neuronas.

Primera capa: La primera capa es la de entrada, que al mismo tiempo es una capa de convolucion donde se define la forma en que los datos van a ingresar a la red neuronal:

- Forma de entrada (input_shape): será una matriz de forma 100x100x3, determinada por las variables: altura, longitud, canales.
- Función de activación: relu.
- Padding: same.

```
red_neuronal.add(Convolution2D(filtrosConv1,tamano_filtro1,padding='same',input_shape=(altura,longitud,canales),activation='relu'))
```

Segunda Capa: La segunda capa será una de pooling (maxpooling), es la encargada de pasar un filtro de cierta altura y cierta longitud sobre la imagen, se coloca el filtro sobre un número de pixeles y lo que va hacer es identificar cual es el número más grande que se encuentra en ese segmento de

pixeles lo va a tomar y lo va colocar en una matriz más pequeña, después hace lo mismo con el resto de la imagen. Para este caso se emplea el parámetro pool_size al que se le asigna el tamaño del filtro.

```
red_neuronal.add(MaxPooling2D(pool_size=tamano_pool))
```

Tercera capa: es otra capa de convolucion

- Función de activación: relu
- Padding: same

```
red_neuronal.add(Convolution2D(filtrosConv2,tamano_filtro2,padding='same'
,activation='relu'))
```

Cuarta capa: es otra capa de pooling que va a reducir todavía más la imagen

```
red_neuronal.add(MaxPooling2D(pool_size=tamano_pool))
```

Quinta capa: es una capa normal, con 650(se usó ese número de neuronas debido a que la red debía de aprender colores, en la sección de fase de investigación se explica más detallado como se llegó a ese número de neuronas) neuronas, que será la encargada de recibir los de la capa de maxpooling.

- **Función de activación:** relu.

```
red_neuronal.add(Dense(525,activation='relu'))
```

Sexta capa: es la capa de salida la cual será la encargada de arrojar la predicción de en qué estado de maduración se encuentra el durazno, la capa tiene 5 neuronas denotado por la variable clases, esto porque son 5 estados de maduración que va a clasificar.

- **Función de activación:** softmax.

```
red_neuronal.add(Dense(clases,activation='softmax'))
```

Se utilizará la perdida de 'categorical_crossentropy' y se procurará mejorar el 'accuracy'.

```
red_neuronal.compile(loss='categorical_crossentropy',optimizer=optimizers.Ad
am(lr=lr),metrics=['accuracy'])
```

El entrenamiento fue con 20 épocas, con 1500 pasos por época. Para asegurar que la red aprendió de manera correcta.

```
red_neuronal.fit(imagen_entrenamiento,steps_per_epoch=pasos,epochs=epocas
)
```

Modelo de Predicción

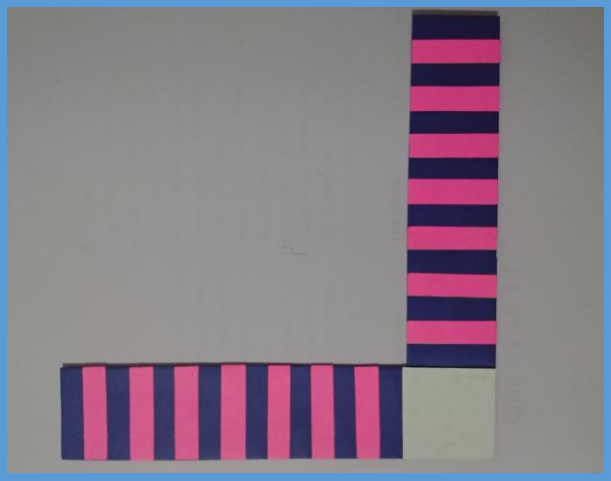
Para usar el modelo de predicción se hizo lo siguiente:

- Cargar el modelo de predicción ya entrenado.
- Redimensionar la imagen a un tamaño de 100 x 100 pixeles.
- Convertir la imagen en un arreglo.
- Enviar como parámetro la imagen al modelo para que de una predicción.
- Dependiendo del resultado que devuelva el modelo, se dará una respuesta al usuario. ejemplo: si se pasa un durazno maduro el resultado que daría el modelo es: [1, 0, 0, 0,0].

Determinación de la calidad del Durazno

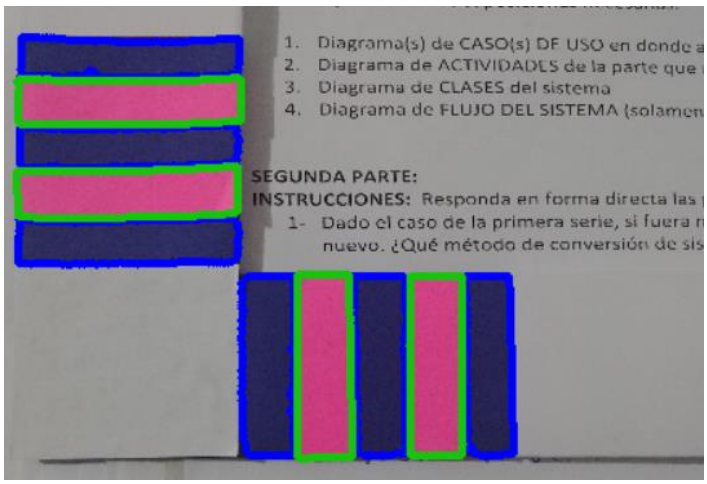
Para determinar la calidad del durazno se utilizó OpenCV, lo que se hizo fue utilizar una especie de regla de rectángulos de colores, Azul y Rosado. Donde estaban intercalados los colores, de esta forma el durazno se coloca a la par de la regla y se delimita hasta la longitud del diámetro del durazno:

Regla Original



Regla delimitada al diámetro del durazno

De esta manera lo que se buscaba hacer era marcar los colores encontrados y contar cuántas veces aparecía el color azul y cuántas veces el color rosado para después sumarlos. Este resultado era el diámetro del durazno y con base al diámetro se podía determinar la calidad del durazno.



Lo que se hace es contar cuantas veces aparece el color azul y cuantas el color rosado. Después cada resultado de divide entre 2 y se suman los resultados para así encontrar el diámetro del durazno y determinar la calidad.

Se definió un rango de colores de azul y rosado para que la no hubieran problemas al momento de detectar los colores, ya que por las condiciones de luz con que se tome la imagen los colores a veces varían. Los rangos de colores fueron los siguientes, empezando del color más bajo al más alto:

Azul:

- azulBajo = [108,100,100]
- azulAlto = [128,255,255]

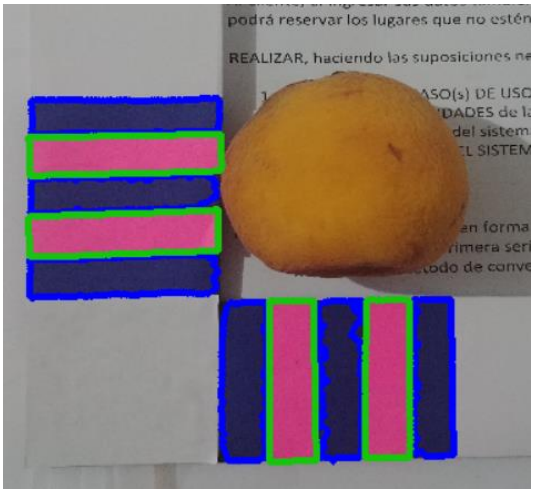
Rosado:

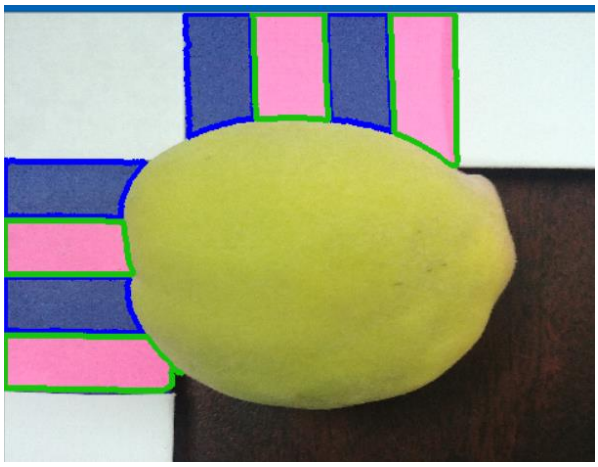
- rosadoBajo = [157,100,100]
- rosadoAlto = [177,255,255]

Los rangos de colores están dados en formato HSV, se utilizó este formato de colores porque con el que trabaja por defecto OpenCV.

Ejemplos:

Los azules encontrados son 6 y los rodados son 4 al dividirlos entre 2 los resultados son 3 y 2 respectivamente, al sumar esos resultados nos da 5 que es el diámetro del durazno y mediante eso podemos determinar la calidad del mismo





Los azules encontrados son 4 y los rosados son 4 al dividirlos entre 2 los resultados son 2 y 2 respectivamente, al sumar esos resultados nos da 4 que es el diámetro del durazno y mediante eso podemos determinar la calidad del mismo

Detección del Durazno

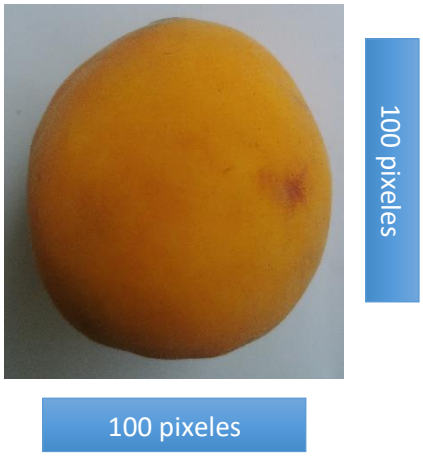
Este modelo consta de detectar si la imagen ingresada es de un durazno o no, si el modelo detecta que la imagen es la de un durazno entonces será enviada al modelo encargado de detectar la calidad y el estado de maduración en el que se encuentra

Tratamiento de los datos de entrenamiento

Para entrenar a la der neuronal que determine si la imagen ingresada es un durazno o no se utilizaron dos categorías para el entrenamiento: los datos que si son duraznos y los que no y se clasificaron de la siguiente manera:

Hay tres carpetas de datos de entrenamiento, dos que contienen imágenes de duraznos y una que no de esta manera la red aprenderá a detectar si la imagen es un durazno o no, por qué hay dos carpetas con duraznos se explica en la fase de investigación.

De igual manera que en el tratamiento de los datos de la red que detecta el estado de madurez, las imágenes fueron redimensionadas de tamaño a uno de 100x100 pixeles.



Los datos fueron normalizados escalándolos de un rango de 0 a 1. Se usó el mismo código que fue utilizado en los datos de entrenamiento de la red neuronal que detecta el estado de maduración.

Los índices de esta red neuronal quedaron de la siguiente manera:

Tipo	Índice
Durazno	0
Durazno	1
NoDurazno	2

Diseño de la red neuronal

El diseño de la red neuronal que detecta si la imagen es un durazno o no es prácticamente el mismo diseño que el de la red neuronal que detecta el estado de maduración del durazno con la excepción de que cambian dos parámetros los cuales son: el número de clases y el número de neuronas.

Los parámetros de la red son:

```
altura = 100
longitud = 100
batch_size = 32
pasos = 1500
canales = 3
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3,3)
tamano_filtro2 = (2,2)
tamano_pool = (2,2)
clases = 3
lr = 0.0005
```

- **Canales:** Los canales de la imagen serán 3 por los debido a que las imágenes a analizar están a color por los valores RGB.

- La altura es de 100 pixeles y longitud de 100 pixeles porque a esa será el nuevo tamaño de la imagen.

- **flitrosConv1:** Serán 32 filtros en la primera capa de convolucion para lograr identificar formas o patrones más elaborados con el paso de capa filtro.

- **Tamano_filtro1:** El tamaño del filtro de la primera capa de convolucion será de 3x3 pixeles, se determinó de esta manera para lograr la recolección de la mayor información posible en la imagen.

- **filtrosConv2:** Serán 64 filtros en la segunda capa de convolucion para lograr identificar formas o patrones más elaborados con el paso de capa filtro.

- **Tamano_filtro2:** El tamaño del filtro de la segunda capa de convolucion será de 2x2 pixeles para recolectar la mayor información posible de la imagen que viene de la primera capa de convolucion y así identificar patrones o formas más elaboradas.

- **Lr:** El factor de aprendizaje es de 0.0005 para garantizar que nuestra red va a aprender de la mejor forma y que el error va a ser los más cercano a 0.

- **Clases:** son 3 clases las cuales serán: Duraznos1, Durazno2, NoDuraznos (el índice de cada clase se lo asigna el programa de pendiendo del orden de las carpetas que contienen las imágenes de entrenamiento).

- Se van a usar 1500 pasos para garantizar que nuestra imagen va a ser lo más pequeña posible y así facilitar el costo de cálculos en la computadora.

- **Batch_size:** será de 32.

- **Épocas:** 15 épocas fueron necesarias para entrenar la red neuronal de manera adecuada.

Estructura del modelo de red neuronal:

Funciones de Activación:

- **Capas ocultas:** Relu
- **Capa de salida:** Softmax

Relu: en la activación de las capas ocultas se escogió la función de activación **relu** debido a que con esta función las redes neuronales aprende de manera más rápida y computacionalmente representa un costo de procesamiento no muy grande, además de que dicha función es aconsejable usarla en redes convolucionales específicamente en las capas ocultas.

Softmax: se escogió esa función debido a que devuelve valores entre 0 y 1, debido a lo cual se considera esta función como una distribución de probabilidad categórica, Softmax se encarga de pasar a probabilidad (entre 0 y 1) a las neuronas de salida.

La red neuronal va a contar con 6 capas en total:

- Primera capa: capara de entrada.
- Segunda capa: capa de pooling.
- Tercera capa: capa de convolución.
- Cuarta capa: capa de pooling.
- Quinta capa: capa densa de con 650 neuronas.
- Sexta capa: capa de clasificación con 5 neuronas.

Primera capa: La primera capa es la de entrada, que al mismo tiempo es una capa de convolucion donde se define la forma en que los datos van a ingresar a la red neuronal:

- Forma de entrada (input_shape): será una matriz de forma 100x100x3, determinada por las variables: altura, longitud, canales.
- Función de activación: relu.
- Padding: same.

```
red_neuronal.add(Convolution2D(filtrosConv1,tamano_filtro1,padding='same',in
put_shape=(altura,longitud,canales),activation='relu'))
```

Segunda Capa: La segunda capa será una de pooling (maxpooling), es la encargada de pasar un filtro de cierta altura y cierta longitud sobre la imagen, se coloca el filtro sobre un número de pixeles y lo que va hacer es identificar cual es el número más grande que se encuentra en ese segmento de pixeles lo va a tomar y lo va colocar en una matriz más pequeña, después hace lo mismo con el resto de la imagen. Para este caso se emplea el parámetro pool_size al que se le asigna el tamaño del filtro.

```
red_neuronal.add(MaxPooling2D(pool_size=tamano_pool))
```

Tercera capa: es otra capa de convolucion

- Función de activación: relu
- Padding: same

```
red_neuronal.add(Convolution2D(filtrosConv2,tamano_filtro2,padding='same'
,activation='relu'))
```

Cuarta capa: es otra capa de pooling que va a reducir todavía más la imagen

```
red_neuronal.add(MaxPooling2D(pool_size=tamano_pool))
```

Quinta capa: es una capa normal, con 150(se usó ese número de neuronas debido a que la red debía de aprender colores, en la sección de fase de investigación se explica más detallado como se llegó a ese número de neuronas) neuronas, que será la encargada de recibir los de la capa de maxpooling.

- **Función de activación:** relu.

```
red_neuronal.add(Dense(150,activation='relu'))
```

Sexta capa: es la capa de salida la cual será la encargada de arrojar la predicción de en qué estado de maduración se encuentra el durazno, la capa tiene 5 neuronas denotado por la variable clases, esto porque son 5 estados de maduración que va a clasificar.

- **Función de activación:** softmax.

```
red_neuronal.add(Dense(clases,activation='softmax'))
```

Se utilizará la perdida de 'categorical_crossentropy' y se procurará mejorar el 'accuracy'.

```
red_neuronal.compile(loss='categorical_crossentropy',optimizer=optimizers.Ad
am(lr=lr),metrics=['accuracy'])
```

El entrenamiento fue con 20 épocas, con 1500 pasos por época. Para asegurar que la red aprendió de manera correcta.

```
red_neuronal.fit(imagen_entrenamiento,steps_per_epoch=pasos,epochs=epocas
)
```

Modelo de Predicción

Para usar el modelo de predicción se hizo lo siguiente:

- Cargar el modelo de predicción ya entrenado.
- Redimensionar la imagen a un tamaño de 100 x 100 pixeles.
- Convertir la imagen en un arreglo.
- Enviar como parámetro la imagen al modelo para que de una predicción.
- Dependiendo del resultado que devuelva el modelo, se dará una respuesta al usuario. ejemplo: si se pasa la imagen de un durazno el resultado que daría el modelo es: [1, 0, 0].

Fase de investigación:

Problema con los datos de entrenamiento

- Las primeras imágenes que se le pasaron a la red neuronal para que entrenara se tomaron con fondos variados (piso, mesa, etc.) además, también se le pasaron con el tamaño original con el que fueron tomadas, esto nos dificultó un problema debido a que las imágenes eran demasiado grandes (tenían un tamaño de 1500x1300 pixeles) hacia que el entrenamiento fuera más tardado y que la red tuviera que procesar una gran cantidad de datos de entrada, también al tomar las imágenes en diferentes fondos ocasionaba que la red no aprendiera de manera correcta y en ocasiones el entrenamiento superaba el 0.75 de aprendizaje, esto se reflejaba en las predicciones que retornaba la red, las cuales eran erróneas.

- Otro problema fue que al principio las imágenes de entrenamiento solo se tomaron con luz natural. Esto fue un problema ya que al momento de pasarle una imagen tomada con luz artificial el resultado de la predicción era erróneo

Ejemplo de como se le pasaron las imágenes durante los primeros entrenamientos:

Solución:

- Se decidió redimensionar las imágenes de entrenamiento a un tamaño de 100x100 pixeles para que no fueran tan grandes y que la red no tuviera que procesas demasiados datos de entrada e incrementar el costo de procesamiento.

- Las imágenes en vez de ser tomadas en diferentes fondos se decidió tomarlas en un fondo blanco para que la red se centrara solo en el durazno.

- También se procedió a tomar las fotografías en diferentes condiciones de luz, se tomaron fotografías con luz natural y artificial, para que así la red aprendiera a identificar el durazno en diferentes condiciones.

Problemas con la estructura de la red neuronal de detección de estado de maduración:

- En el primer diseño de la red neuronal, la capa densa estaba compuesta de 128 neuronas, al entrenar el modelo con esa cantidad de neuronas se notó que la red neuronal no entrenaba de manera correcta, su aprendizaje se estancaba en un punto (aproximadamente 0.57), y al momento de las predicciones estas eran erróneas.

- Otro problema fue determinar el factor de aprendizaje adecuado para que la red neuronal aprendiera, se empezó con un factor de 0.5 sin embargo con este factor al momento de entrenar la red hacia que el aprendizaje fluctuara saltando de un valor mayor a un valor menor haciendo eso todo el entrenamiento, para lo cual se empezó a variar el factor de aprendizaje.

Solución:

- Pensamos que este problema se debía a la cantidad de neuronas, porque para poder predecir correctamente la red debía de aprender los colores de cada etapa del durazno, por lo que asumimos que la red no aprendía los colores de manera adecuada debido a la poca cantidad de neuronas. Empezamos a aumentar el número de neuronas para ir entrenando y probando. Al final se logró que la red aprendiera de manera correcta llegando a un 1.0 de aprendizaje y que tuviera predicciones acertadas, el número de neuronas con el que aprendió bien fue de 650 neuronas.

- Para determinar el valor óptimo de la constante de aprendizaje con la que el modelo pudiera aprender de manera correcta fue necesario ir variando la constante a un número cada vez menor hasta encontrar un valor adecuado para que la red entrenara lo mejor posible. Se llegó un valor de 0.0005, con este valor la red entrenó excelente y dando predicciones correctas.

- Al realizar estos cambios en los datos de entrenamiento. Notamos la mejora en el entreno del modelo, el nivel de aprendizaje del modelo llego a 1.0, que se vio reflejado en las predicciones que realizaba, dando así un resultado correcto.

Problemas para determinar la calidad del durazno

- Para detectar la calidad del durazno se tuvo un problema debido a la solución que se implementó fue de usar la detección de colores de una regla segmentada con dos colores, azul y rosado. El problema consistía al momento de ingresar la imagen y detectar los colores estos no era correctamente identificados (a veces los detectaba y otras no), lo que ocasionaba que el resultado de la calidad fuera erróneo.

- Otro problema fue que al momento de detectar los colores. Estos los detectaba con mucho ruido, hacía que el cálculo del diámetro del durazno fuera erróneo.

Solución:

- La solución fue definir un rango de colores (se definió un rango del color azul desde el más claro hasta el más oscuro, de igual manera se hizo con el rosado) para que, sin importar las condiciones de luz en que fuera tomada la imagen el programa pudiera detectar correctamente los colores y así contar cuantas veces aparecía cada uno, para determinar así la calidad del durazno y devolver un resultado exacto.

- Para solucionar el problema del ruido se creó una condición donde se indica que, si el color detectado tiene un área mayor a 400 que se marque y se cuente, de lo contrario no será tomado en cuenta.

Problemas con la red neuronal que detecta si la imagen es un durazno o no

- Un problema fue que al inicio del entrenamiento solo había una carpeta de imágenes que contenían todos los tipos de duraznos para que la red pudiera aprender a identificar un durazno o no, el problema consistía en que después de entrenar el modelo, cuando se hacía una predicción y se le pasaba una imagen de un durazno la predicción que retornaba era errónea y decía que no era un durazno a pesar de que la imagen si era de un durazno.

Solución

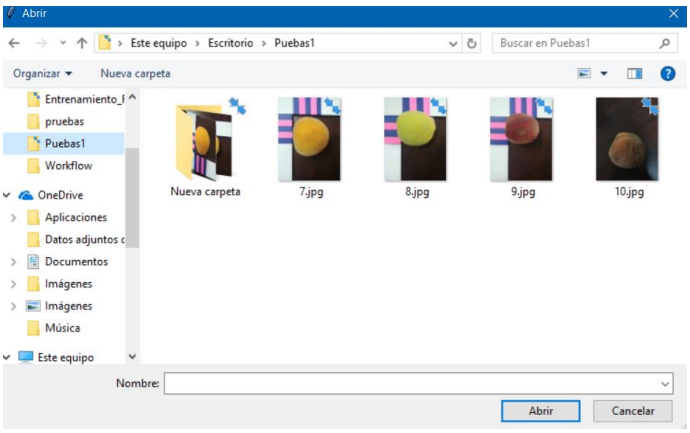
- La solución a este problema fue que se decidió separar en dos carpetas la imágenes de duraznos, las imágenes de los duraznos maduros, sobre maduros, verdes y podridos fase 1 quedaron en una carpeta y las imágenes de duraznos podridos fase 2 quedaron en otra. Esto quedo de esta manera debido a que se pudo observar que el problema de la predicción errónea era porque los duraznos podridos fase 2 ya no tiene una forma circular debido a que se secan, por lo cual al tener mezclado estas imágenes con las de los demás duraznos ocasionaba que la red se confundiera en la clasificación, por lo cual se decidió separar ese tipo de duraznos y al hacer eso el modelo ya pudo retornar predicciones correctas.

Funcionamiento de la aplicación

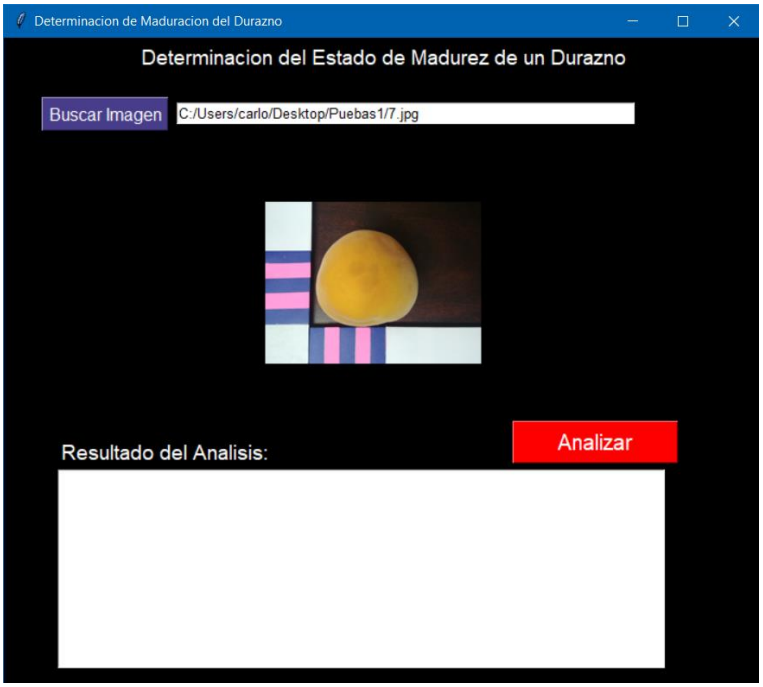
El archivo que se debe correr es el llamado “Interfaz.py”, este contiene la aplicación completa y lista para funcionar.

Al momento de correr la aplicación aparecerá una interfaz gráfica:

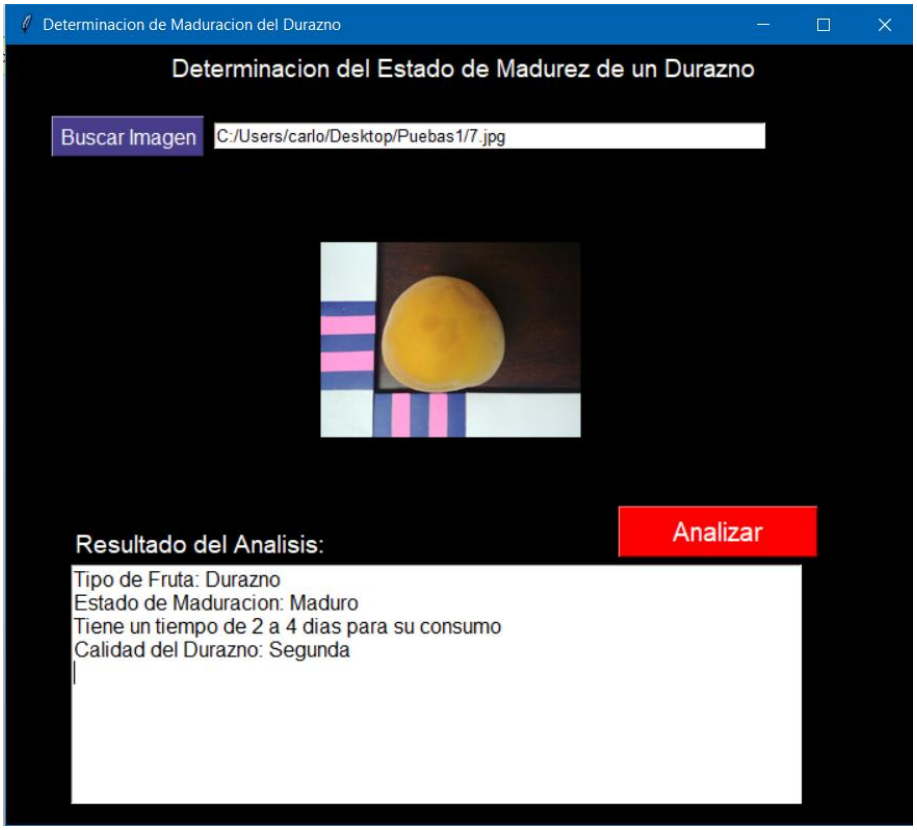
Para buscar la imagen a analizar se debe presionar el botón “Buscar Imagen “este abrirá un cuadro de dialogo donde se buscará la imagen seleccionada:



Después de seleccionar la imagen se presiona el botón abrir aparecerá la imagen seleccionada en la interfaz principal:



Para determinar el estado de maduración de la fruta se debe presionar el botón “Analizar”:



En el recuadro blanco aparecerá toda la información acerca del durazno. Como, por ejemplo: tipo de fruta, estado de maduración en el que se encuentra, tiempo en que se puede consumir y la calidad del durazno.

Si la imagen seleccionada no es de un durazno, en el recuadro blanco aparecerá un mensaje indicando que la imagen no es un durazno por lo cual no se puede analizar.

Nota: Para determinar la calidad del durazno se debe de poner el durazno al lado de la regla, si no se colocar la regla, al momento de analizar al durazno en los resultados no aparecerá la calidad.

Anexos

Dentro de la carpeta de la aplicación van los siguientes archivos:

- El archivo llamado “Interfaz.py” es el que contiene la aplicación completa con los modelos entrenados para detectar si la imagen es un durazno o no y también contiene el modelo que determina el estado de maduración del durazno y la calidad del mismo.

- El archivo llamado “Clasificar.py” fuel el que se usó para entrenar el modelo que determina el estado de maduración del durazno.

- El archivo llamado “DeteccionDurazno.py” fuel el que se usó para entrenar el modelo que detecta si la imagen es un durazno o no.

También en la carpeta principal va otra carpeta llamada “Imágenes Pruebas”, que contiene imágenes de prueba para el modelo.

Se deben de descargar los siguientes archivos:

Archivos del modelo de determinación de la madurez del durazno:

- "modelo_10.h5"
- "pesos_10.h5"

Archivos del modelo de detección del durazno:

- "modelo_Deteccion.h5"
- "pesos_Deteccion.h5"

Sin estos archivos la aplicación no funcionara. estos archivos no fueron incluidos en el repositorio debido a que son demasiado pesados y github no permitido subirlos porque indicaba que superaban el limite máximo permitido, por lo cual fueron subidos a Google Drive, el link de descarga es: **https://drive.google.com/open?id=1SnnNOiNWP93lYWiFM5P-06ee1Spy7B2b**

Después de descargarlos se deben pegar en la carpeta “Modelo” que está en la carpeta principal y agregar la ruta absoluta de estos archivos en el código del archivo “Interfaz.py” (se especifica en donde debe ir en el código).

La ruta absoluta de estos archivos debe ir en:

- El método **Ver_durazno()** aquí debe ir la ruta absoluta de "modelo_Deteccion.h5" y "pesos_Deteccion.h5"

```
def ver_durazno(imagen):

    logitud,altura = 100,100
    #*******************Carga del modelo de deteccion si la imagen es de un duraz
    modelo='C:/Users/carlo/Desktop/Entrenamiento_Frutas/Modelo/modelo_Deteccion.h5'
    pesos= 'C:/Users/carlo/Desktop/Entrenamiento_Frutas/Modelo/pesos_Deteccion.h5'
```

- También debe de ir la ruta absoluta de "modelo_10.h5" y "pesos_10.h5" en la parte principal de programa:

```
from tkinter import *
from tkinter.filedialog import askopenfilename
from PIL import Image
from PIL import ImageTk
from resizeimage import resizeimage
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import load_img,img_to_array
import cv2

nombre_imagen=""
logitud,altura = 100,100
#************************ Carga  del modelo y los pesos para la deteccio
modelo='C:/Users/carlo/Desktop/Entrenamiento_Frutas/Modelo/modelo_10.h5'
pesos= 'C:/Users/carlo/Desktop/Entrenamiento_Frutas/Modelo/pesos_10.h5'
```

Después de agregar ese cambio la aplicación ya será funcional

Código:

En el archivo “Interfaz.py” que la aplicación funcional, están los siguientes métodos:

- **Ver_durazno()**: este método es el encargado de llamar al modelo de detección de durazno y verificar si la imagen es un durazno o no.

- **Mostrar_resultado():** es el método encargado de llamar al modelo que identifica en qué estado de maduración se encuentra el durazno

- **calidadDurazno():** es el método encargado de determinar la calidad del durazno

Las imágenes que se usaron para el entrenamiento están disponibles en el siguiente link: https://drive.google.com/open?id=1XbuBDCGPczddOm-q_YWxJUkO4KUPH7PT
Deben de ir en carpeta de pruebas1-imgduraznos o bien modificar la ruta a donde se tengas las imágenes.