

UNIVERSIDAD RAFAEL LANDIVAR.  
CAMPUS DE QUETZALTENANGO.  
FACULTAD DE INGENIERIA.



INTELIGENCIA ARTIFICIAL  
PROYECTO

04/05/2018

# PROYECTO INTELIGENCIA ARTIFICIAL

El cultivo de naranja en Guatemala, genera 3,607 empleos permanentes.

Se trata de un fruto en forma esférica y color anaranjado brillante.

## Aspectos técnicos

Requerimientos edafoclimaticos

Temperatura:

Es una especie sub- tropical, no tolera las heladas, ya que sufren tanto las flores y frutos como la vegetación, que pueden desaparecer totalmente. En condiciones como las de Guatemala, el rango de temperaturas óptimas para el desarrollo de la naranja esta entre los 23 y los 34 grados centígrados. Precipitación pluvial: Los cálculos en las necesidades de agua de los naranjos entre 1,200 a 1,500 mm. De lluvia por año preferentemente bien distribuida. La lluvia solo se puede sustituir por aplicaciones artificiales de agua (riego).

Altitud: En las zonas tropicales desde el nivel del mar hasta los 1,500m.

Vientos: Intensidad, no mayores de 25 kilómetros por hora, para evitar caída de flores y frutos pequeños rozaduras de frutos por ramas, que afectan la apariencia física.

Luz Solar: El rango adecuado se considera entre 1,600 a 2,000 horas por año.

PH del suelo: El rango ideal es de 5.5 a 7.

Materia Orgánica: El rango ideal es entre el 2% y el 4%.

Topografía: En topografías de onduladas a quebradas orientar las plantaciones de norte a sur; En caso de que no se pueda se aconseja sembrar en curvas de contorno con calles amplias.

Disponibilidad de agua: Se deberá contar con estudios específicos del lugar en cuanto a las tasas de evapotranspiración de la variedad elegida, condiciones de retención de humedad por el suelo (capacidad de campo y punto de marchites permanente) dependiendo de la textura de suelo, para obtener la frecuencia de riego, que supla las demandas de agua por la planta en periodos críticos o para el uso de la técnica del estrés hídrico y poder diseñar el sistema de riego más eficiente al menor costo.

Una vez cosechada la naranja desde que tiene un color verde oscuro (ya para consumir), en condiciones normales, las naranjas pueden aguantar de media de 15 a 30 días.

Pasos fundamentales:

### **Recolección de datos (imágenes):**

Para la recolección de datos se debe tener a la mano, las fotografías de la fruta seleccionada en sus niveles de madurez, el cual puede representarse en una escala de 1 a 10. En nuestro caso, esa escala será la que se utilizará para indicar el nivel de madurez y el nivel de calidad. De preferencia que se tenga 5 frutas por cada nivel, esto para aumentar el nivel de precisión. En nuestro caso utilizaremos una y se tomarán las 5 fotos en distintas posiciones de la fruta.

Para cada uno se contará con al menos 5 fotografías de cada nivel, teniendo un total de 50 fotografías, debidamente clasificadas para madurez y clasificadas también para calidad. La clasificación para cada una será diferente, como puede ser que la calidad de una sea baja, pero esté con el nivel indicado para su consumo, o también puede ser que la calidad sea alta, y con un nivel indicado para su consumo.

El tamaño de la fotografía para el entrenamiento como para las pruebas no importa, debido a que esta información se normaliza para su mejor tratamiento.

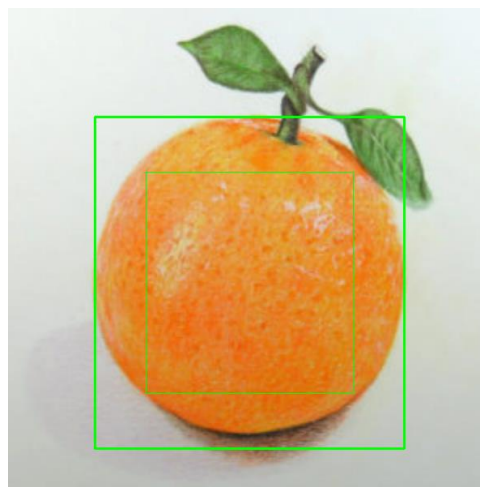
### **Selección de rangos de colores para la fruta (Niveles de madurez):**

La fruta va desde los colores verdes, hasta los colores naranjas, por lo tanto el rango queda abierto a estos dos colores predominantes. Verdes para los inmaduros y naranja para los maduros.

### **Encontrar la fruta dentro de la imagen. (Por cada imagen para el entrenamiento).**

Al ingresar la imagen, lo primero que se realiza es la normalización de la imagen, lo que se mencionaba anteriormente, la normalización se realiza haciendo un resize con un tamaño de 600x600.

Se hace una búsqueda dentro de la imagen de los colores que estamos buscando, y una vez encontrado el color, busca el centro de todo el color y marca en un cuadro donde se está concentrado el color.

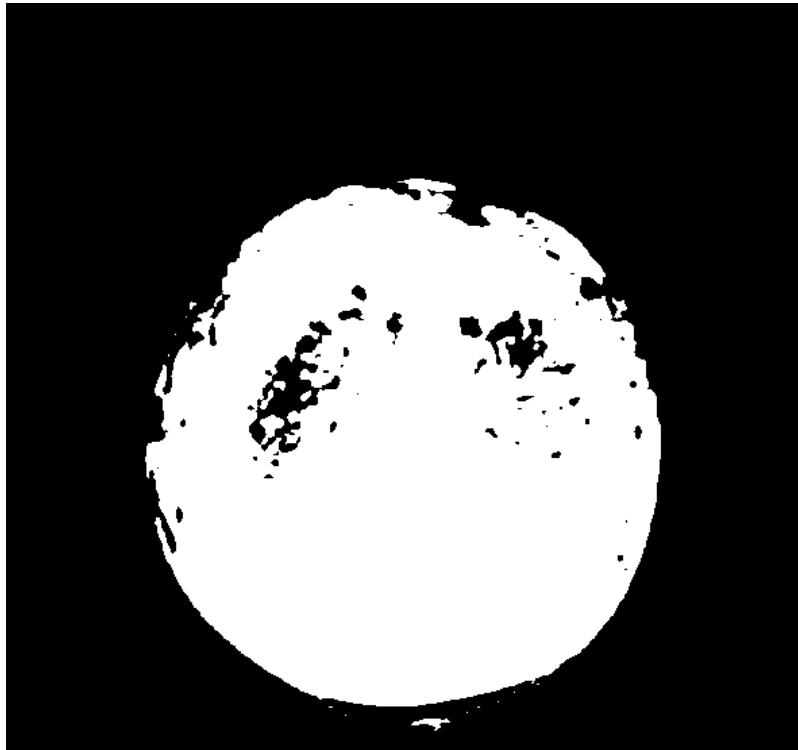


*imagen1*

Podemos observar en la *imagen1*, como se encontró la naranja debido al color que posee, se aclara que si el fondo es verde o naranja, se tendrá confusión y no se podrá procesar adecuadamente la imagen.

El recuadro grande verde en la imagen, representa donde se encuentra concentrada toda la fruta, pero en las esquinas nos genera errores al momento de colocarlo en el entrenamiento, debido a que las esquinas contienen otros colores, para evitar esto se hace un nuevo recuadro, que es el recuadro verde pequeño en la imagen, ya con esto si se tiene solo el color que posee la fruta.

Para poder llegar al resultado, primero se realiza la búsqueda del color haciendo la imagen en blanco y negro, blanco donde el color concuerda.



*imagen2*

Al momento de tener la imagen como la que se muestra en la *imagen2*, se procede a marcar el cuadro donde se encuentra blanco.

Al tener la imagen ya seleccionada la parte donde se encuentra la fruta, procedemos a recortar la imagen, y nos quedaría de la siguiente manera.



*imagen3*

Se procede a normalizar el resultado obtenido, a una nueva de 15x15 pixeles. Esto se hace para poder tener una entrada normalizada y que el entrenamiento se pueda completar al no tener demasiadas entradas. Y queda de la siguiente manera.



Y esa última imagen es la que se utiliza para entrenar nuestra red neuronal. De la misma manera se llega a tener este resultado al obtener las pruebas.

#### ENTRADAS:

Las entradas a la red neuronal son  $15 \times 15 \times 3$ , un total de 675 entradas. Esto es porque nuestra imagen resultante es de 15x15 pixeles y por cada pixel tenemos 3 datos que son los del RGB. Con esto podemos definir nuestras neuronas de la capa de entrada, que serían 675 neuronas de entrada.

Cada dato obtenido de cada imagen para el entrenamiento, es almacenado en un archivo. Al final se debe tener 50 archivos. Los cuáles serán utilizados para el entrenamiento.

Cada un archivo de los 50 tendrá 675 datos que son: 3 valores por cada pixel, rojo, verde y azul.

#### ENTRENAMIENTO:

2 redes neuronales se tendrán. 1 para la calidad y la otra para la madurez.

Para el entrenamiento se tienen 50 patrones, cada uno representa una imagen. Cada patrón con 675 datos. Se cargan todos los 50 patrones en un array y se empiezan a pasar por la red neuronal uno por uno.

Para el entrenamiento se utilizará neurolab.

Antes de iniciar el entrenamiento, se vuelve a normalizar las entradas para que el entrenamiento sea más efectivo, ya que se reducen las cantidades, porque las cantidades del RGB van desde 0 a 255, por lo tanto se normaliza de 0 a 100. Esto se realiza multiplicando por 4 y dividiéndolo dentro del 10. (Si no se realiza esto, el entrenamiento es mucho más lento y no llega a una solución.).

Para las neuronas de salida se tendrán 10, cada uno representará el nivel de madurez y el nivel de calidad.

Y las salidas esperadas (targets) son los siguientes:

Nivel 1: 0000000001

Nivel 2: 0000000010

Nivel 10: 1000000000

Cualquier otro valor obtenido, se considerará como error y no se tomará en cuenta.

RESULTADOS OBTENIDOS DEL ENTRENAMIENTO: Aún en proceso (no se tienen resultados concluyentes aún.)

## Fase 2:

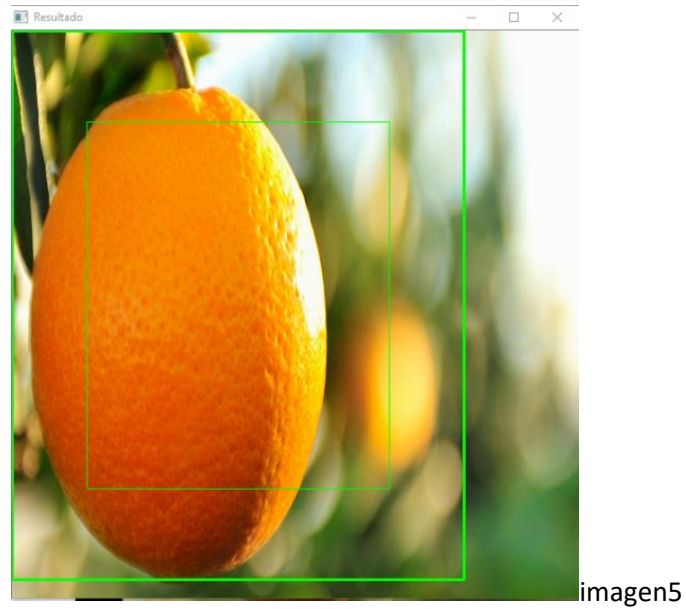
### Recolección de datos (imágenes):

Durante las pruebas de recolección de datos en las imágenes, se tuvieron problemas al momento de encontrar la fruta, por los motivos de que los colores a encontrar son muy abiertos, es decir que van desde el verde incluyendo bajos y altos, hasta los naranjas de igual manera altos y bajos. Y a la hora de buscar por colores, encontraba cosas que no tenían que ir y colocaba lo que no era dentro de los recuadros. Además que definir el rango de colores resulta un poco complicado por el hecho de que existen muchos rangos.

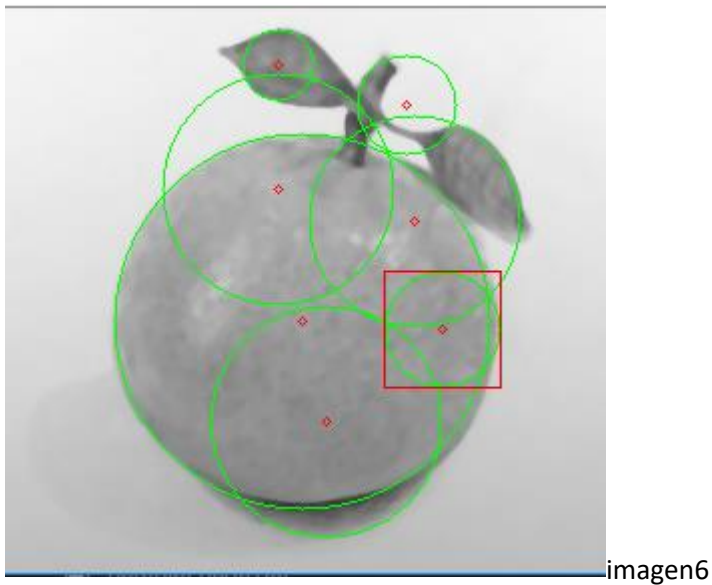


imagen4

En la imagen4 podemos ver que estuvo muy acercado, sin embargo si la fruta es de un tamaño más pequeño, los cuadros siguen siendo del mismo tamaño, dejando en el recorte gran cantidad de datos que no se deben incluir.



Se procedió a realizar la búsqueda por medio de figuras geométricas, con la ayuda de la función de cv2: cv2.HoughCircles.



Como vemos en la imagen6, encuentra demasiados círculos, en este caso por ser fondo uniforme y fácil de detectar la fruta, solo nos encontró unos cuantos, pero a la hora de buscar en una imagen con mayores probabilidades de encontrar círculos en el fondo nos queda de la siguiente manera:



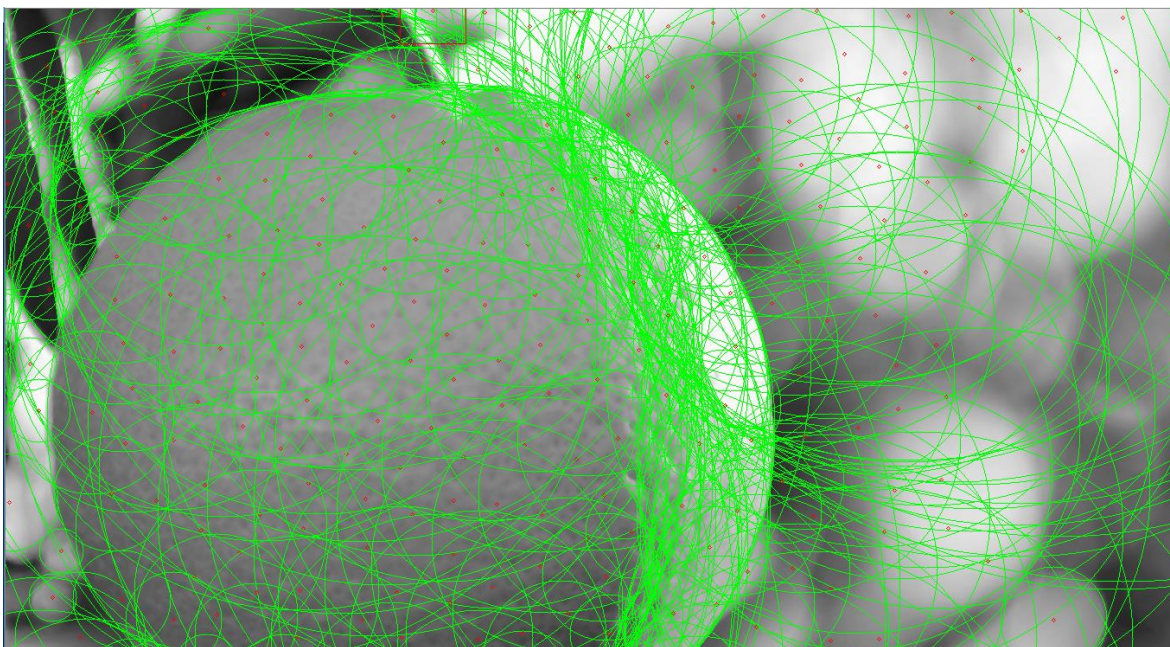


imagen7

En la imagen7 vemos los resultados. Sin embargo se pudo eliminar este error al modificar una variable en la función que realiza la búsqueda de círculos, más adelante vemos los resultados obtenidos después de solucionar este problema.

Los resultados obtenidos fueron mejores en el sentido de encontrar la fruta con mayor exactitud.

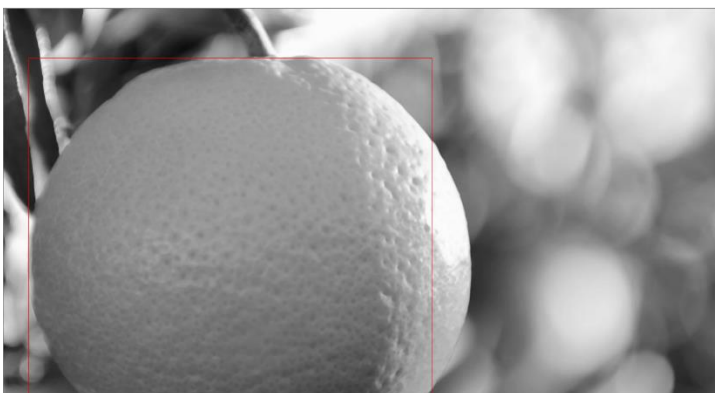


imagen8

En la imagen se muestra el cuadro grande, no el cuadro pequeño, y cómo podemos observar ya se ha encontrado la ubicación de la fruta, sin embargo el tiempo de procesamiento y los recursos requeridos para llegar al resultado es muy elevado, el tiempo fue de 15 segundos, y con la búsqueda por color, se realiza en tan solo 2 segundos. Al buscar por figuras en la imagen 4, no se llega a encontrar nada durante 5 minutos, el programa se queda corriendo, y no llega a encontrar nada, por la cantidad de píxeles que posee la imagen4.

En imágenes donde la cantidad de píxeles es demasiado alto nos genera problemas, sin embargo si la cantidad de píxeles no es elevado entonces llega a encontrar la fruta (círculo) fácilmente.



Por lo tanto se procede a realizar cambios a la imagen original. En este caso tenemos 2 opciones, reducir el tamaño de pixeles en la imagen original o, crear nuevas mascarar para poder encontrar con mayor facilidad los círculos.

#### SOLUCIÓN:

Después de tratar de nuevo buscando por colores, se determinó que buscar por círculos resulta ser más factible, porque al buscar por colores, resulta complicado con el hecho de que van desde verde hasta naranja y esto es muy amplio, que empieza a buscar cosas que no debiera buscar, por lo tanto se optó buscar por círculos.

El error de buscar círculos en imágenes con cantidad de pixeles muy grande, resultó ser tardado, sin embargo al realizar un resize a la imagen original, se solucionó el problema de los recursos y el tiempo empleado. Para esto se hizo de la siguiente manera:

if height > 1000 or width > 1000:

```
img = cv2.resize(img, (0,0), fx=0.3, fy=0.3)
```

if height > 500 or width > 500:

```
img = cv2.resize(img, (0,0), fx=0.7, fy=0.7)
```

if height > 300 or width > 300:

```
img = cv2.resize(img, (0,0), fx=0.9, fy=0.9)
```

Las líneas marcadas en verde, nos realizan un resize de la imagen, en el orden que aparece. Por ejemplo, si la imagen tiene dimensiones ya sea x o y, mayores a 1000, se realizará un resize del 30% del tamaño original y así para las demás condiciones. Esto se hizo con el objetivo de que las imágenes resultantes tengan un tamaño entre los 250 y 450 pixeles.

Después de normalizar la imagen de entrada, se obtuvieron buenos resultados del tiempo de respuesta, la imagen que no encontró respuesta alguna, después de normalizarla se pueden encontrar círculos y con tan solo 3 segundos.

Sin embargo, a pesar de obtener resultados satisfactorios, se obtuvieron algunas variantes que se presentan a continuación:

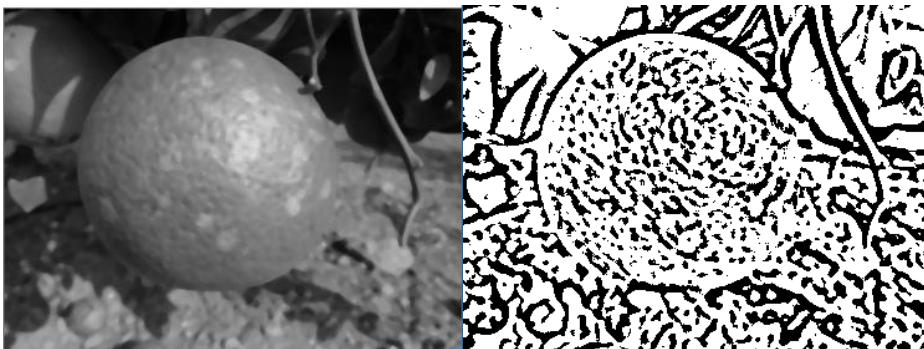


imagen8



En las imágenes se puede observar el resultado obtenido al buscar la fruta en un fondo confuso, Al eliminar el código que resulta de la imagen8, nos queda:



De igual manera podemos observar que debido al fondo confuso, no se llega a encontrar la fruta. Por lo tanto, al momento de tomar la captura de la fruta a evaluar, se recomienda que el fondo no sea confuso con la fruta, y de preferencia que no tenga colores parecidos al de la fruta.

Con esto se concluye la fase de recolección de datos, y las entradas a la red neuronal.

En la siguiente entrega se estará indicando los resultados obtenidos durante el entrenamiento de la red neuronal con las entradas que ya se han recolectado hasta el momento.

### FASE 3: Entrenamiento.

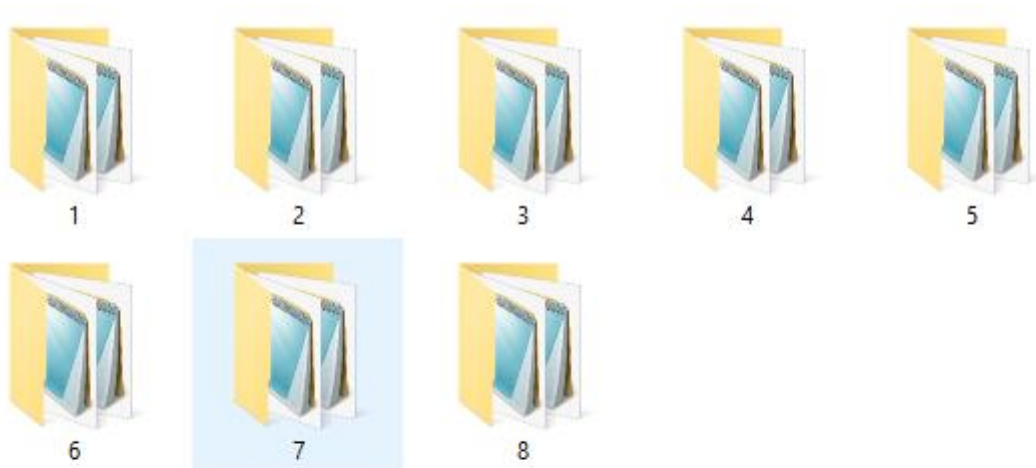
Para el entrenamiento ya se cuenta con las fotografías de las frutas ya clasificadas, se había mencionado 10 niveles en entregas anteriores, sin embargo se optó por 8 niveles de madurez para su clasificación.

Para las pruebas se realizó con 10 fotografías por cada nivel, teniendo un total de 80 fotografías para el entrenamiento, y con una entrada a la red neuronal de 1200 valores, los cuales van desde 0 a 1.

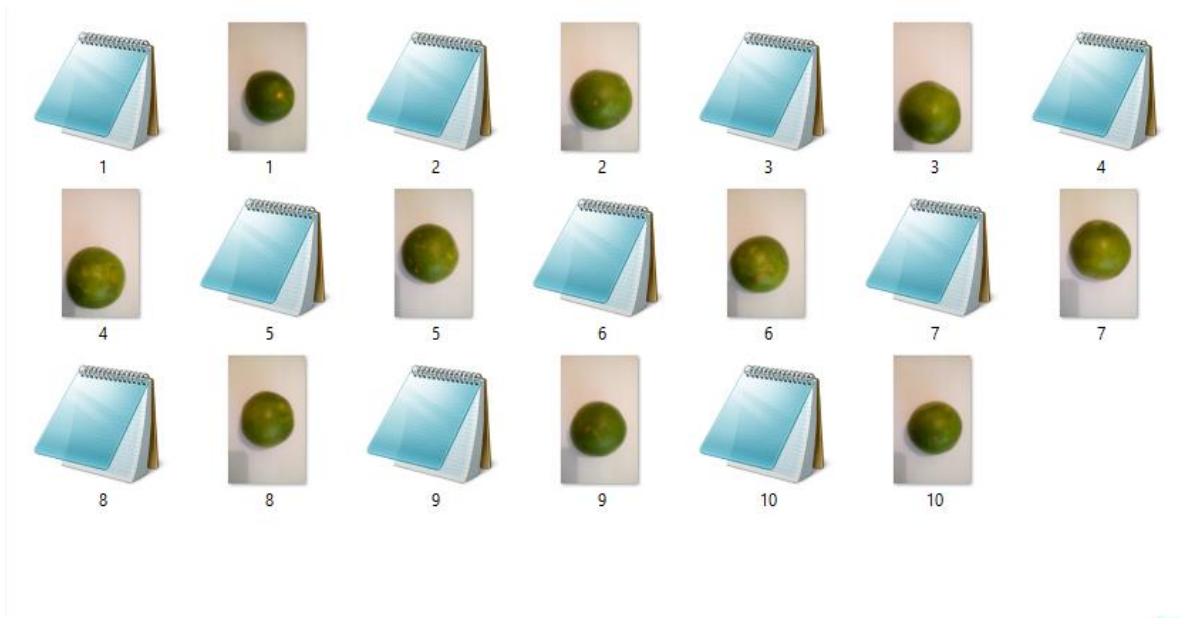
Se realizó una normalización al resultado de la imagen, y esto se hizo dividiendo el valor del RGB directamente entre 255, con el objetivo que el valor de la entrada nos quede con el rango de 0 a 1.

Para las pruebas de entrenamiento se realizó con recortes de las fotografías normalizadas a 20x20 pixeles, con el objetivo de entender el funcionamiento de neurolab respecto a las variaciones de tiempo que conlleva tener mayor o menor número de entradas, que es con lo que vamos a realizar la fase del entrenamiento.

Para esta fase, se debe tener clasificada las imágenes:



Cada uno representa el nivel de madurez (lo mismo para los niveles de calidad). Y dentro de cada uno se encuentran las imágenes, con los archivos generados en la búsqueda de la fruta.



Cada archivo, contiene las entradas que serán utilizadas a la red neuronal.

Durante la carga de datos de las imágenes, se encontró con un error que no dejaba avanzar la fase del entrenamiento.

```

48 rango = np.zeros((len(input1[0]),2))
49 b = np.zeros(len(input1))
50 for y in range(0, len(rango)):
51     b=b-b
52     for x in range(0, len(input1)):
53         b[x] = input1[x,y]
54     rango[y]=[b.min(), b.max()]
55 #print rango
56 net = nl.net.newff(rango, [5, 8])
57 # Train network

Python - Entrenar.py:54
C:\Users\abyez\Anaconda2\lib\site-packages\neurolab\init.py:132: RuntimeWarning: divide by zero encountered in divide
  x = 2. / (minmax[:, 1] - minmax[:, 0])
C:\Users\abyez\Anaconda2\lib\site-packages\neurolab\init.py:133: RuntimeWarning: invalid value encountered in multiply
  y = 1. - minmax[:, 1] * x

```

Se procedió a revisar la carga de archivos para encontrar el error, también en la clasificación y en el array que se tenía como salida esperada, y ninguno de esos contenía errores. Hasta que se encontró el error del porqué nos aparecía que teníamos división entre 0. En la imagen de arriba se puede apreciar en donde estaba el error. Y se marca a continuación en donde se encontró el error.

```

48 rango = np.zeros((len(input1[0]),2))
49 b = np.zeros(len(input1))
50 for y in range(0, len(rango)):
51     b=b-b
52     for x in range(0, len(input1)):
53         b[x] = input1[x,y]
54     rango[y]=[b.min(), b.max()]
55 #print rango
56 net = nl.net.newff(rango, [5, 8])
57 # Train network

```

Python - Entrenar.py:54

```

C:\Users\abyez\Anaconda2\lib\site-packages\neurolab\init.py:132: RuntimeWarning: divide by zero encountered in divide
  x = 2. / (minmax[:, 1] - minmax[:, 0])
C:\Users\abyez\Anaconda2\lib\site-packages\neurolab\init.py:133: RuntimeWarning: invalid value encountered in multiply
  y = 1. - minmax[:, 1] * x

```

En la documentación de neurolab y en algunas ayudas que se pueden encontrar en internet, nos mencionan que datos debe llevar nuestra red al momento de crearla, y como primer parámetro debe llevar los valores posibles de entrada en un rango, es decir: Si tenemos como entrada [2,3,4,5,6,7,8,9], nuestros valores serían [2,9], los cuales representa el mínimo y el máximo. Y eso es lo que se pretendió con el código que se ve en pantalla. Pero se procedió a sustituir esa línea, por un mínimo de 0, por ejemplo [0, 1] (se dejó constante), porque al momento de realizar una búsqueda del mínimo o máximo como se muestra en la imagen de arriba, nos generaba un error de división entre 0, al dejar constante el rango en [0,1], se pudo empezar a entrenar.

Al momento de realizar el primer entrenamiento, se obtuvieron los siguientes resultados:

Python - Entrenar.py:54

```

Epoch: 2; Error: 1710.47976264;
Epoch: 4; Error: 1442.45970993;
Epoch: 6; Error: 1230.46309768;
Epoch: 8; Error: 1104.27339089;
Epoch: 10; Error: 1035.86890601;
Epoch: 12; Error: 988.263771368;

```

Como se puede observar en la imagen de arriba, nuestra red neuronal está funcionando como se esperaba que fuera, pero como la prueba se realizó con 1200 entradas en lugar de 675, eso provocó que el entrenamiento fuera muy lento, de manera que las 12 épocas que se observan en la imagen, llevó un tiempo de 12 minutos.

Al cambiar la entrada a como se tenía planeado con 675 valores, se obtuvieron resultados más rápidos. 12 épocas se realizó con tan solo 2.5 minutos. La diferencia entre las 1200 entradas y 675 entradas, es muy significativa respecto al tiempo, por el momento se han dejado las entradas en 80 fotografías y cada una con 675 entradas para la red neuronal. A continuación se ilustra los resultados.


Python - Entrenar.py:54

```

Epoch: 2; Error: 1569.00146307;
Epoch: 4; Error: 1298.79248684;
Epoch: 6; Error: 1012.49379006;
Epoch: 8; Error: 964.989380253;
Epoch: 10; Error: 939.128967407;
Epoch: 12; Error: 869.918590552;

```

Aproximadamente 30 minutos de entrenamiento.

Python - Entrenar.py:54 

```
Epoch: 158; Error: 27.7083911553;  
Epoch: 160; Error: 27.6189830269;  
Epoch: 162; Error: 27.5114275137;  
Epoch: 164; Error: 27.4087522629;  
Epoch: 166; Error: 27.2961633216;  
Epoch: 168; Error: 27.2122827827;  
Epoch: 170; Error: 27.1291598648;  
Epoch: 172; Error: 27.0855838578;  
Epoch: 174; Error: 27.0554788587;  
Epoch: 176; Error: 26.977742739;  
Epoch: 178; Error: 26.9316926456;
```

Con los datos provistos hasta el momento no se pudo completar el entrenamiento, teniendo como Error 20. Y el valor ya no descendió de 20. Después de realizar varias pruebas y revisiones, se pudo constatar que el rango de datos de la entrada en la red, no tenía los números que debía tener, porque se tenía entre 0 y 1 el rango, y las entradas iban desde 0 hasta 10. Esto provocó que no se llegara a la meta de error en el entrenamiento, y al modificar los datos de entrada, normalizándolos y dejándolos en el rango de 0 a 1, se obtuvieron mejores resultados, descendiendo el error de 20 a 8, Los resultados obtenidos fueron los siguientes:

- Para todas las pruebas con una entrada de 675, y una salida de 6 los cuales representan los niveles de madurez de la fruta. Lo único que se varió fueron las neuronas de la capa intermedia.
  - 1 neurona capa intermedia. Error mínimo obtenido: 25.
  - 2 neuronas capa intermedia. Error mínimo obtenido: 22.
  - 3 neuronas capa intermedia. Error mínimo obtenido: 15.
  - 4 neuronas capa intermedia. Error mínimo obtenido: 13.
  - 8 neuronas capa intermedia. Error mínimo obtenido: 10.
  - 12 neuronas capa intermedia. Error obtenido: 25. Con las 12 neuronas solamente se realizó una prueba de entrenamiento, debido al tiempo que lleva realizarlo.

Se realizó un cambio en la entrada, originalmente se tenía 8 niveles de madurez, pero para que la red llegara a entrenar se redujo a 6, eliminando el 2 y el 7 de la lista que se tenía, que eran los que más similitud tenían con el nivel anterior o posterior. Se dejó así como predeterminado, 6 niveles.

No se pudo entrenar con el diseño que se tiene, por lo tanto se optó por cambiar las funciones de transferencia, inicialmente o por defecto, newff cuenta con la función de transferencia TanSig. Y cuenta con más posibilidades de función de transferencia, entre los cuales están: LogSig, y PureLin.

Al realizar pruebas con las nuevas funciones de transferencia se obtuvieron resultados ya concluyentes para el entrenamiento. Los resultados fueron:

- Para todas las pruebas con una entrada de 675, y una salida de 6 los cuales representan los niveles de madurez de la fruta. Lo único que se varió fueron las neuronas de la capa

intermedia. Con las funciones expresadas de la siguiente manera. (Funcion\_capa\_oculta Funcion\_capa\_salida)

- 1 neurona capa intermedia. Error mínimo obtenido: 18. (TanSig LogSig)
- 2 neuronas capa intermedia. Error mínimo obtenido: 5. (TanSig LogSig)
- 3 neuronas capa intermedia. Error mínimo obtenido: 3. (TanSig LogSig)
- 4 neuronas capa intermedia. Error mínimo obtenido: 1. (TanSig LogSig)
- 5 neuronas capa intermedia. Error mínimo obtenido 0.5 (TanSig PureLin)
- 6 neuronas capa intermedia. Error mínimo obtenido: 0.05. (PureLin PureLin)
- 8 neuronas capa intermedia. Error mínimo obtenido: 0.1. (TanSig PureLin)

Durante las pruebas de entrenamiento se pudo observar como varía la capacidad de entrenamiento en base a cada combinación de función de transferencia, resultando como mejor opción en este caso PureLin para la función de transferencia en la capa oculta y en la capa de salida. Con esto ya se tiene entrenada la red neuronal.

Para los valores de calidad, se realizó el mismo procedimiento, incluyendo las mismas funciones de transferencia. A diferencia que para la calidad, se tienen contemplado 3 tipos de calidad, baja, media y alta.



## Manual para usar los documentos del proyecto que se encuentran en el repositorio

Dentro del directorio de IA2018, se encuentran los siguientes archivos:

- calidad: Carpeta donde contiene las imágenes para el entrenamiento de calidad, catalogadas en base a un número del 1 al 3.
- naranja: En esta parte se encuentran las imágenes que se utilizaron para el entrenamiento de madurez de las naranjas, catalogadas del 1 al 8, 1 representa inmadurez y 8 representa que está muy madura.
- pruebas: En esta parte se colocan las pruebas a realizar para ver su estado de madurez y calidad, pueden estar las imágenes a probar en cualquier otra parte, siempre y cuando se cambie en el directorio a buscar la imagen.

Los archivos donde se almacenaron las redes neuronales ya entrenadas son:

- Calidad: La mejor entrenada es calidad.net, sin embargo si llega a fallar en algunos resultados, su sucesor es calidad2.net.
  - calidad.net. Error mínimo de 0.1 (TanSig, LogSig).
  - calidad2.net. Error mínimo de 0.1 (PureLin, PureLin).
  - calidad3.net. Error mínimo de 0.05 (PureLin, PureLin).
- Madurez: Entre todas las entrenadas, la mejor es red4.net, sin embargo en algunas ocasiones no es factible y otras como la red3.net si es factible o viceversa.
  - red.net. Error mínimo de 5. (TanSig, LogSig).
  - red2.net. Error mínimo de 0.5. (TanSig, PureLin).
  - red3.net. Error mínimo de 0.1 (TanSig, PureLin).
  - red4.net. Error mínimo de 0.05 (PureLin, PureLin).
- Documentos .py (Programación):
  - Encontrar naranja en imagen: Este es el primer documento a utilizar, debido a que es el encargado de encontrar la fruta en la imagen y realizar el recorte para preparar la entrada a la red de entrenamiento. Este es el encargado de generar los documentos donde se encuentra la información de los datos obtenidos para el entrenamiento. Para más detalles técnicos o de programación, ver documentación en documento.
  - Entrenar madurez: En este se encuentra la red neuronal encargada de realizar el entrenamiento para la madurez, los archivos que genera son las redes de madurez. Las entradas para este son las salidas de "Encontrar naranja en imagen".
  - Entrenar calidad: Es el encargado de realizar el entrenamiento para los tipos de calidad de la naranja, este es el responsable de generar las redes de calidad, sus entradas son las salidas de "Encontrar naranja en imagen".
  - Tomar foto: Este es el que se utiliza para realizar fotografías desde la cámara de la computadora, aunque no es recomendado utilizar si se tiene baja resolución de cámara. Se utiliza para realizar pruebas, los datos se almacenan en pruebas.
  - Encontrar foto y probar: Este es el último en ser utilizado, con este se realiza la búsqueda de la imagen a que se desea saber su estado de madurez y de calidad,

este busca en la imagen si encuentra la fruta, posteriormente en base a la red que se haya seleccionado por la cual van a pasar los datos realiza un análisis y genera un resultado en base a las condiciones de la fruta.