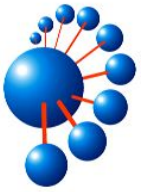
Universidad de Concepción
Facultad de Ingeniería
Depto. Ingeniería Informática y Ciencias de la Computación



Proyecto Final Python Científico

Detección de objetos sospechosos

Integrantes:

Natalia Soto.
Diego Varas.

Docente:

Pamela Guevara.

Concepción -17 Diciembre, 2018

Índice

1. Introducción

El siguiente documento tiene como objetivo evidenciar el trabajo realizado en la asignatura python científico del semestre 2-2018. En primera instancia se hablará del problema a abordar, siguiendo con la solución propuesta, para luego dar paso a la explicación del procedimiento que se siguió finalizando con las conclusiones obtenidas.

1.1. Problema

Dados los cambios tecnológicos experimentados en el último tiempo, han crecido las necesidades de automatización de procesos, donde las tareas de exigencia superior se han delegando a robots o máquinas especializadas en el área señalada, dando paso a que las personas cada vez más se desliguen de dichas tareas, tomando eso, nos centramos en el área de empaquetado de frutas, en la cual solo trabajan personas separando las frutas, es por eso que el proceso está propenso a fallos, ya sea porque simplemente no se vio un objeto ajeno a los productos o se sufrió una desatención en el proceso, es por eso que se necesita otra forma de supervisar el correcto empaquetado, dado que si este proceso no se realizara, puede provocar que se pase un objeto de dudosa procedencia como se menciona sin que nadie lo note, lo que a su vez puede ocasionar serios daños a la salud de los posibles clientes de aquellos productos o porque no decirlo demandas a la empresa en cuestión.

1.2. Objetivos

- Detectar objetos en una cinta transportadora.
- Detectar objetos en imágenes.
- Lanzar alerta cuando se detecta un objeto extraño.
- Aplicar los conocimientos adquiridos en el curso de python y otros cursos.
- Crear una interfaz segura capaz de automatizar el proceso de chequeo de frutas en cintas transportadoras.

2. Solución

Dado lo anterior se propone un sistema de detección de objetos sospechosos en cintas transportadora por medio de programación en lenguaje python, donde se tiene un programa que sea capaz de enviar una alerta cuando se detecte algo distinto a lo que debería pasar por dicha cinta, en este caso se enfocó el proyecto en la detección de manzanas, entonces cuando se mostrara algo distinto a estas emitiera una alerta inmediata.

3. Metodología

El diseño del programa se realizó siguiendo el esquema de Modelo Vista Controlador (MVC) [7], por lo que compone de 3 archivos principales, el primero es *MotorDetector.py*, el cual corresponde al controlador, por lo que en el se encuentran todos los métodos que controlan tanto a la vista, como al modelo, el segundo archivo es *vista.ui*, que como su nombre lo dice, contiene la vista del programa, la cual fue creada en Qt, mediante QtDesigner y por último está el archivo *Reconocimiento.py*, el cuál contiene todos los métodos que permiten al programa realizar el reconocimiento y análisis de las imágenes, correspondiendo al modelo.

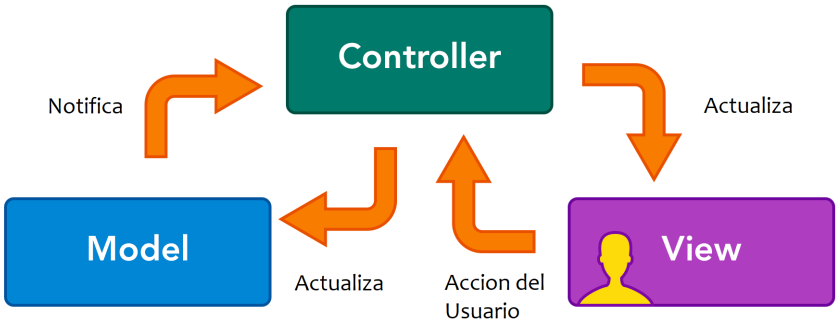


Figura 1: Interacción del usuario en el Modelo Vista Controlador

3.1. Primeras pruebas

Para la primera instancia del proyecto se pensó el procesamiento de las imágenes de manzanas mediante OpenCV y una serie de transformaciones en las imágenes, como escala a grises, Adaptación Gaussiana, y la herramienta de Canny para detección de bordes, creyendo que se podían llegar a resultados esperados, sin embargo el procesamiento se alejaba de lo que realmente se esperaba, como se muestra a continuación, donde por ejemplo los bordes no eran detectados de buena manera, lo que a su vez hacía que mostrara más objetos de lo que realmente habían.

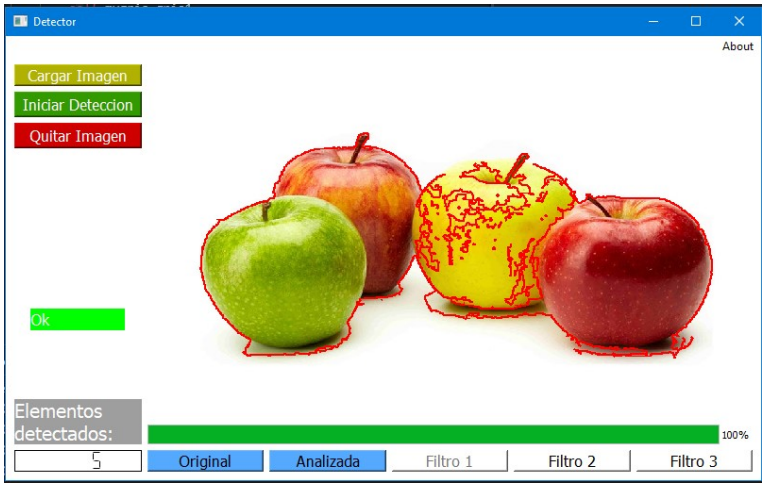


Figura 2: Pruebas usando transformaciones

Dado que las herramientas que se escogieron no eran las adecuadas y muchas de ellas eran un poco obsoletas, se investigan nuevas opciones, llegando a la biblioteca ImageAI [1] una librería creada por Moses Olafenwa y John Olafenwa, encargada de cargar los modelos de tipo RetinaNet [3] y Yolo [5] utilizados para nuestro detector, complementado con la librería TensorFlow [2], la cual nos permite realizar la ejecución del programa utilizando el procesamiento gráfico del computador, en nuestro caso solo nos fue posible habilitar el uso de procesamiento integrado con el procesador, si bien nuestros computadores poseen tarjetas gráficas dedicadas, estas no son compatibles.

3.2. Detector de objetos final

Para la ejecución de este programa se necesita la instalación de *Python 3.6.6* y una serie de librerías detallada que se resumen a continuación [1]

```
*Tensorflow  pip install tensorflow
*Numpy       pip install numpy
*Scipy       pip install scipy
*OpenCv      pip install opencv-python
*Pillow      pip install pillow
*Matplotlib  pip install matplotlib
*H5py        pip install h5py
*Keras       pip install keras
*ImageAI     pip3 install https://github.com/OlafenwaMoses/ImageAI/releases/
download/2.0.2/imageai-2.0.2-py3-none-any.whl
*PQyt5 pip install pyqt5
```

Además se debe obtener los modelos que se procederán a cargar y entrenar los cuales son Retinanet [4] y Yolo [5] dado el tiempo que requiere formar un modelo de estas características se decidió utilizar los de los link señalados. Para iniciar el programa se hace llamando a MotorDetector.py a través del comando python MotorDetector.py donde se abrirá una ventana como se muestra a continuación:

Figura 3: Pantalla Inicial

Luego de eso procedemos a cargar la imagen apretando el botón cargar imagen, seleccionamos el archivo y damos a abrir, una vez realizado ese proceso procedemos a apretar el botón iniciar detección el cual nos arrojará una ventana que nos pide seleccionar el modelo

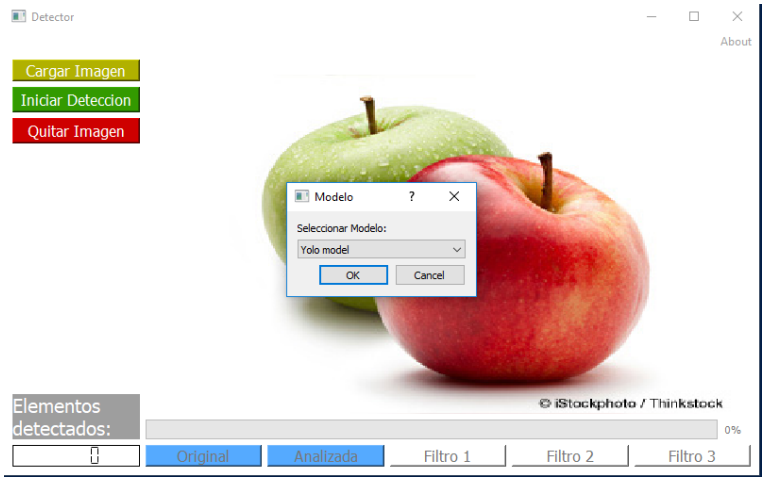


Figura 4: Selección de Modelo

Luego de seleccionado el modelo nos pide seleccionar la velocidad en que se realiza la detección, una detección más rápida implicará más errores en el proceso, una detección dura alrededor de 20 segundos, este tiempo no es alterado por la elección de la velocidad, ya que esta depende netamente del procesador gráfico de la computadora.

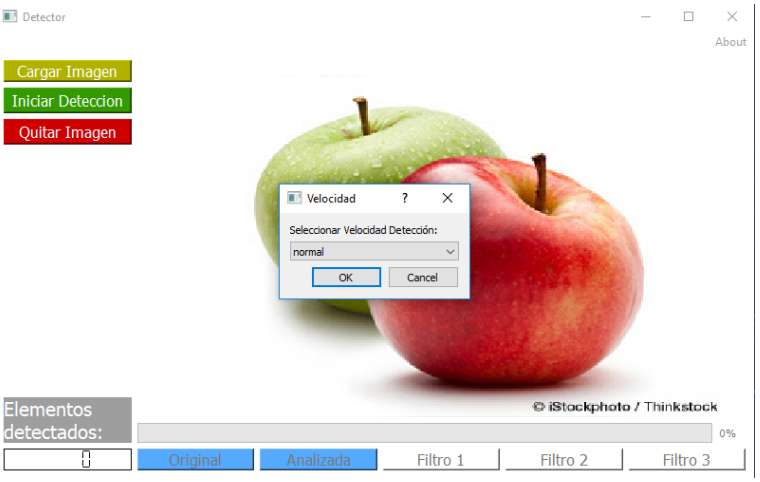


Figura 5: Pantalla de selección de velocidad

Por una parte al utilizar el modelo RetinaNet [3] que si bien detecta manzanas y otros objetos sospechosos, no tiene la misma precisión como si fueran personas o autos para lo que realmente fue pensado, es decir para nuestro trabajo tuvimos que adaptar el modelo de acuerdo a nuestros requerimientos. Es por eso que hace que el reconocimiento contenga un cierto grado de error, obteniendo los siguientes resultados:

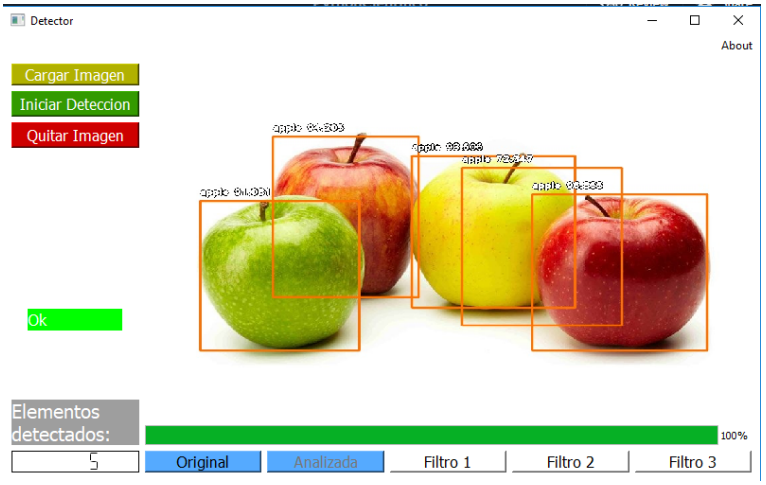


Figura 6: Detector de objetos fase final con RetinaNet

Como se puede apreciar en la *Figura 6*, claramente en la imagen hay 4 manzanas, pero la detección con RetinaNet [3] detecta 5. Por otra parte si utilizamos el modelo Yolo [5] nos entrega una detección más precisa en cuanto a manzanas se habla disminuyendo los errores que se obtenían al usar RetinaNet, obteniendo por ejemplo este resultado con la misma imagen analizada:

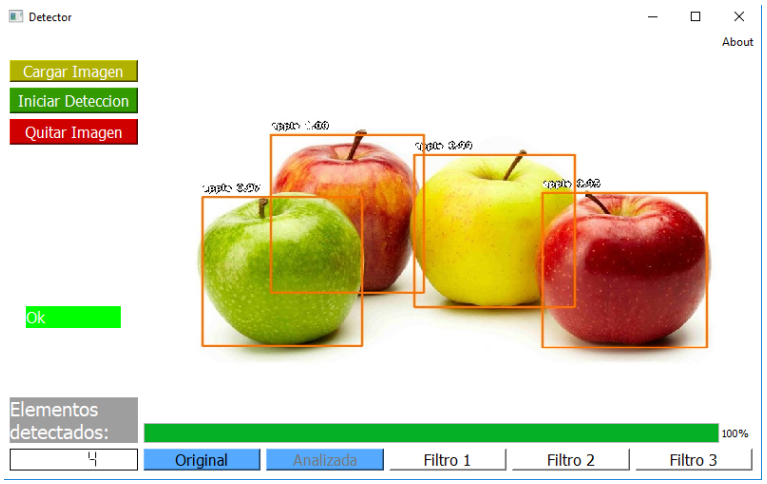


Figura 7: Detector de objetos fase final con YOLO

En la *Figura 7* se utiliza la misma imagen que en la *Figura 6*, pero se utiliza el modelo de detección Yolo [5], donde se puede notar una mejoría en la detección, ya entregando la cantidad correcta de elementos detectados.

3.3. Tareas Realizadas

A lo largo del proyecto se realizaron dos tipos de tareas principales la primera de ellas fue los procesamientos con las imágenes y la forma en que se manejaban los datos y por otro lado se tenía la tarea que se preocupaba de la interfaz del detector, cada uno de los miembros se encargó de una de ellas, lo que fue registrado mediante Github [6], lo que nos permitió tener un control en el trabajo de cada uno y además tener un control de las versiones de nuestra aplicación.

4. Resultados

Todas las pruebas fueron realizadas en una Laptop con procesador Intel® Core™ i3-5005U 2.0Ghz, 8Gb de Ram. El tiempo de detección promedio de nuestro programa es de aproximadamente 20 segundos por imagen, sin importar la cantidad de elementos que esta

contenga, esto se da porque como se explicó en la *sección 3.1* de *Metodología*, el procesamiento lo realiza la Gpu integrada con el procesador, lo cual hace que este proceso tarde ese tiempo, si se pudiera realizar con una Gpu dedicada este tiempo se reduciría significativamente.
Como se puede apreciar en la *Figura 6*, la cual fue una imagen analizada con el modelo RetinaNet, en esta se puede apreciar como detecta una manzana más, siendo que en la *Figura 7*, se analiza con el modelo Yolo, el cual detecta correctamente las manzanas, esto se debe a que ambos modelos están orientados a la detección de personas y autos, pero Yolo posee un entrenamiento superior en la detección de manzanas, lo que permite tener mejores resultados.

4.1. Alcances y limitaciones

- El proyecto es capaz de procesar imágenes de distinto tipo, no así vídeos ni imágenes en tiempo real, dado que no se tenía el conocimiento suficiente para eso aún. Sin embargo se encontró la documentación necesaria para posibles adaptaciones a futuro mediante la librería ImageAI [1].

- El detector no es capaz de analizar imágenes que tienen una toma diagonal, dado que se produce mucho error en la detección, solo soporta imágenes desde arriba o de frente, esto debido al poco entrenamiento que poseen los modelos, lo cual se puede mejorar si estos se enfocan solo en la detección de manzanas.

5. Conclusión

Para finalizar debemos decir al comienzo se pretendió analizar vídeos, lo cual fue cambiado por la detección en imágenes, dado de la complejidad que esto tiene. Este nuevo objetivo se comenzó a llevar a cabo con herramientas que nos dificultaron el proceso, para luego enmendar el rumbo y encontrar la librería *ImageAi*, la cual nos ayudo a que el objetivo planteado al inicio se cumpliera de acuerdo a lo esperado, si bien, como se mencionó hubieron ciertos obstáculos que se debieron pasar, la idea principal no perdió el enfoque.
En el futuro si se llegase a continuar con este proyecto se podría adoptar para la detección en tiempo real.

Referencias

[1] Librería ImageAI, https://imageai.readthedocs.io/en/latest/detection/index.html Programa base para reconocimiento de objetos, con base de datos entrenada.

[2] Tensorflow, definición, https://es.wikipedia.org/wiki/TensorFlow Página visitada el día 13 diciembre 2018

[3] RetinaNet, explicación del modelo, https://github.com/fizyr/keras-retinanet/blob/master/README.md

[4] RetinaNet, Modelo utilizado, https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/resnet50_coco_best_v2.0.1.h5

[5] Yolo, Modelo utilizado, https://github.com/OlafenwaMoses/ImageAI/releases/tag/1.0/ Modelo yolo.h5 para descargar

[6] Repositorio del detector realizado, https://github.com/dvarasm/Python-Cientifico/tree/master/Proyecto Código del detector, ubicada en la branch master

[7] Modelo Vista Controlador https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador