

Manual de usuario

SISTEMA DE DETECCIÓN DE PEATONES EN LA NOCHE

LUIS BARRENO

Tabla de contenido

Deployment.....	3
Introducción	3
Generalidades del sistema	3
Métodos más importantes.....	3
Requisitos del programa.....	6
Configuración y ejecución.....	7
Configuración rutas.....	7
Compilación y ejecución	8
Entrenamiento	8
Configuración solver.prototxt.....	9
Configuración capa_datos.py.....	10
Configuración train_test.prototxt.....	11
Configuración Train/Test.txt	11
Configuración ejecutar_solver.py	11

Deployment

Introducción

Este manual está orientado para desarrolladores y usuarios que quieran hacer uso de los códigos programados en el sistema de detección de peatones en la noche mediante imágenes infrarrojas, y cubrirá principalmente el modo de uso, compilación y ejecución del sistema desde una forma básica y general. Este manual fue escrito el 10 de junio de 2017.

Generalidades del sistema

El sistema está conformado por cinco objetos pertenecientes a las clases: Detector, Candidatos, Clasificador, HeadDetector y RegionGrowing relacionados según la **Figura 1**.

El objeto de la clase Detector aparte de sus variables de instancia y métodos está compuesto de dos objetos, un objeto de la clase Candidatos y un objeto de la clase Clasificador (llamados candidatos y clasificador respectivamente).

El objeto de la clase Clasificador tiene como función principal calcular y devolver las confianzas de los candidatos que entran en la red, es decir propagar hacia adelante cada candidato.

Encima de las dos clases utilizadas para generar candidatos a peatones (HeadDetector y RegionGrowing) se tiene la clase Candidatos, esta clase permite generar candidatos por cada método o por combinación de los dos.

Métodos más importantes

Los métodos más importante que son imprescindibles para el funcionamiento del sistema se describe en la **Tabla 1**.

Tabla 1. Métodos del sistema más importantes.

Detector		
Método	Valor de retorno	Descripción
empezarDeteccion (Mat & img, float umbral)	void	Es un método de alto nivel que realiza todo el trabajo del sistema. Crea un objeto de candidatos y clasificador para que hagan su parte del trabajo y muestra la imagen con las detecciones obtenidas que pasen un umbral de clasificación.
Candidatos		
getCandidatosHeadDetector()	vector<Mat>	Retorna los candidatos obtenidos por la clase HeadDetector, específicamente primero llama al método de HeadDetector que calcula los candidatos y después los obtiene.
getCandidatosRegionGrowing()	vector<Mat>	Retorna los candidatos obtenidos por la clase RegionGrowing.
getCandidatosTodos()	vector<Mat>	Retorna la unión de los vectores obtenidos por los dos métodos anteriores.

Clasificador		
setSalidaRed(Mat & img, vector<Mat> * regiones)	void	Calcula la propagación hacia adelante de la red, dado la imagen de dos canales y un vector con los candidatos.
getSalidaRed()	Mat	Retorna las confiancias de los candidatos con el mismo orden que tienen en el vector de regiones.

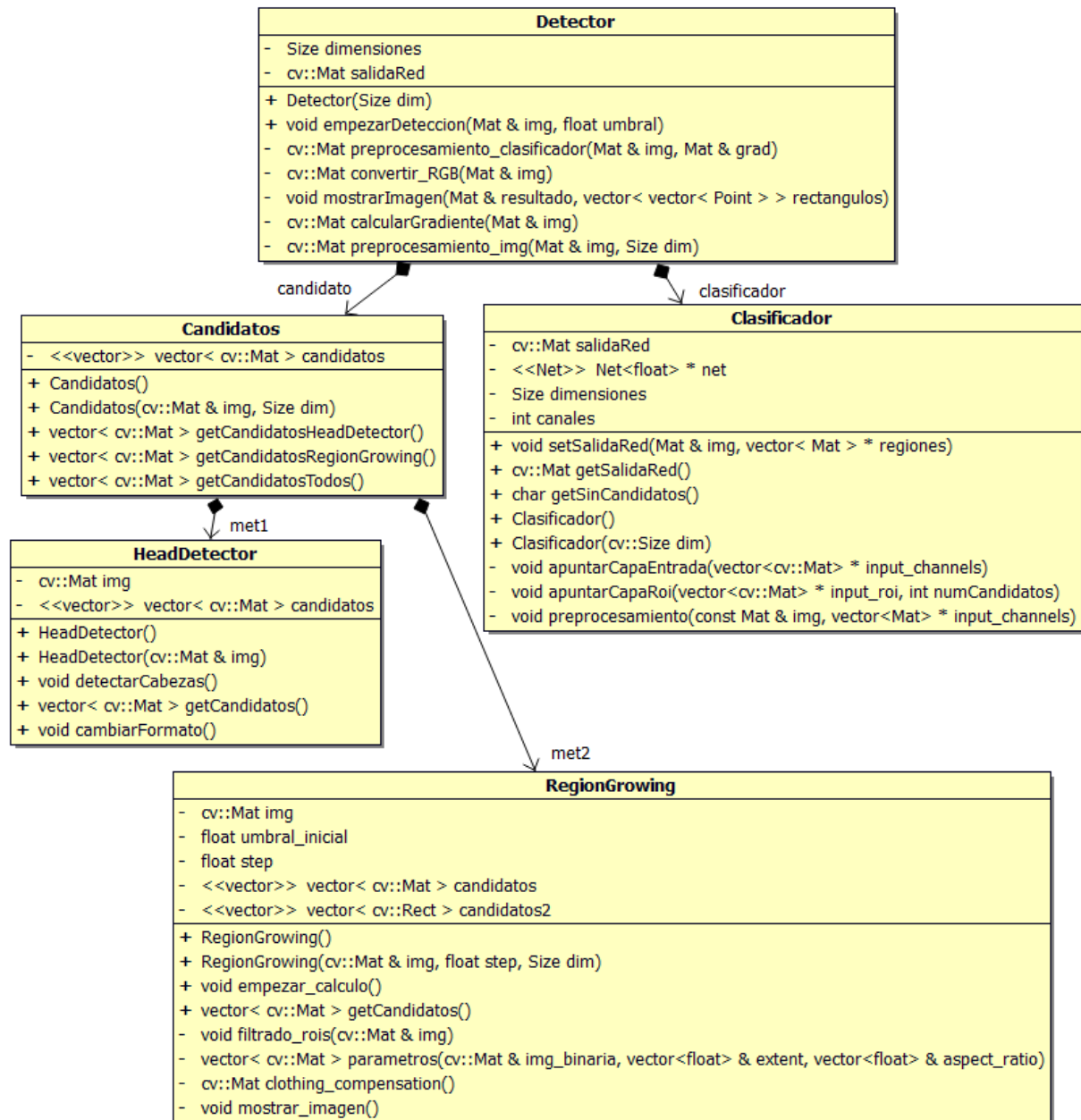


Figura 1. Diagrama de clases del sistema de detección de peatones.

Requisitos del programa

Los requisitos previos de softwares instalados para poder ejecutar el programa son:

Tabla 2. Softwares y librerías utilizadas.

Software/librería	Nombre	Versión	Instalación
Sistema operativo	Ubuntu	14.04	http://releases.ubuntu.com/14.04.5/
Librería visión artificial	opencv y opencv_contrib	3.1.0	http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu/
Librería deep learning con soporte C++ y Python	Caffe-fast	Caffe-fast	https://github.com/rbgirshick/fast-rcnn
Compilador C++11	gcc	4.8.5	Viene instalado en el Ubuntu 14.04.
Interprete python	Python	2.7.6	Viene instalado en el Ubuntu 14.04.
Editor de texto (opcional)	Atom	1.8.0	https://atom.io/
Editor de texto con soporte C++ (opcional)	Eclipse CDT	3.8	Instalado directamente del Ubuntu Software Center.

El sistema fue programado en C++ en un computador de escritorio Intel Core i7-4790 CPU @ 3.60GHz y 8 GB de RAM con sistema operativo Ubuntu 14.04 de 64 bits.

Configuración y ejecución

Configuración rutas

Para compilar y ejecutar el programa primero se tiene que establecer cuáles son las rutas de los diferentes archivos que se utilizan en el sistema. Estas rutas están especificadas dentro del archivo rutas.h.

Tabla 3. Variables que especifican rutas del sistema.

Nombre de la variable	Descripción
-----------------------	-------------

RUTA_FOLDER_LSI	Ruta de la carpeta LSI Test de la base de datos LSI de imágenes infrarrojas.
RUTA_VIDEO	Ruta de algún video sobre el cual se quiera probar el sistema.
RUTA_ALARMA	Ruta del archivo de audio en formato wav, el cual sonara a manera de alarma cuando se detecte un peatón.
RUTA_ARQUITECTURA	Ruta de los archivos necesarios para definir la arquitectura, los pesos y el modelo debe estar en esta carpeta.
MODO_CAFFE	Definición del modo de ejecución de caffe, puede ser CPU o GPU.

Compilación y ejecución

Para compilar se puede utilizar cualquier framework que soporte C++ como eclipse o netbeans o se puede compilar manualmente directamente sobre la terminal de comandos con el archivo makefile proporcionado.

Entrenamiento

Para comenzar con el entrenamiento es necesario configurar los siguientes archivos dentro de la carpeta Entrenamiento preferiblemente en este orden:

1. solver.prototxt
2. capa_datos.py
3. train_test.prototxt
4. Train.txt
5. Test.txt
6. ejecutar_solver.py

Configuración solver.prototxt

En este archivo se definieron los siguientes parámetros:

Tabla 4. Parámetros de entrenamiento.

Parámetro	Valor
Tasa de aprendizaje	0.002
Factor de castigo	0.0005
Momentum	0.9
Gamma	0.0001
Power	0.75

Caffe ofrece varios métodos de minimización de la función de costo, el método elegido para resolver esta red es Descenso de Gradiente Estocástico (SGD, por sus siglas en inglés Stochastic Gradient Descent) el cual tiene la ventaja que no calcula la función de costo sobre toda la base de datos de entrenamiento si no esta se calcula solo sobre un minibatch muestreado de la base de datos. La ecuación de actualización de los pesos es la siguiente:

$$(1)$$

$$(2)$$

Donde:

Representa el momentum.

Taza de aprendizaje.

Función de costo.

El valor de la tasa de aprendizaje se actualiza en cada iteración según la ecuación:

Configuración capa_datos.py

Caffe permite que los usuarios definan sus propias capas, en este trabajo se define esta capa como entrada de datos donde se configuran las variables:

```
num_imagenes_por_batch = 10
num_rois_neg = 10
```

Caffe tiene la ventaja que permite definir al usuario sus propias capas, es por esto que para la lectura de datos se creó una nueva capa llamada *CapaDatos* la cual permite leer un archivo de entrenamiento en donde constan todas la imágenes a utilizar en conjunto con los candidatos de la forma como se muestra en la **Figura 2**, en donde la primera línea representa el número de imagen, la segunda línea la ruta de la imagen, las demás líneas representan los candidatos encontrados con el método de generación de candidatos que incluye la clase (peatón o no peatón) esta clase se determina según el grado de solapamiento que se tenga con algún candidato, incluye también el solapamiento y por ultimo las coordenadas del cuadro delimitador.

```
# 0
/ruta img
clase iou x1 y1 x2 y2
clase iou x1 y1 x2 y2
clase iou x1 y1 x2 y2
.....
.....
```

Figura 2. Formato del archivo utilizado en el entrenamiento.

Estas imágenes se leen aleatoriamente en la capa de datos y son alimentadas a la red en cada iteración, una vez que se tiene definida esta capa, se definen las demás capas según la arquitectura determinada en la sección anterior.

Configuración train_test.prototxt

En este archivo se define la arquitectura que se va a entrenar, añadiendo una función de costo que se va a minimizar durante el entrenamiento.

Configuración Train/Test.txt

Estos archivos se generan con la carpeta GenerarArchivosEntrenamiento, donde se escriben dos archivos en formato txt con el siguiente formato:

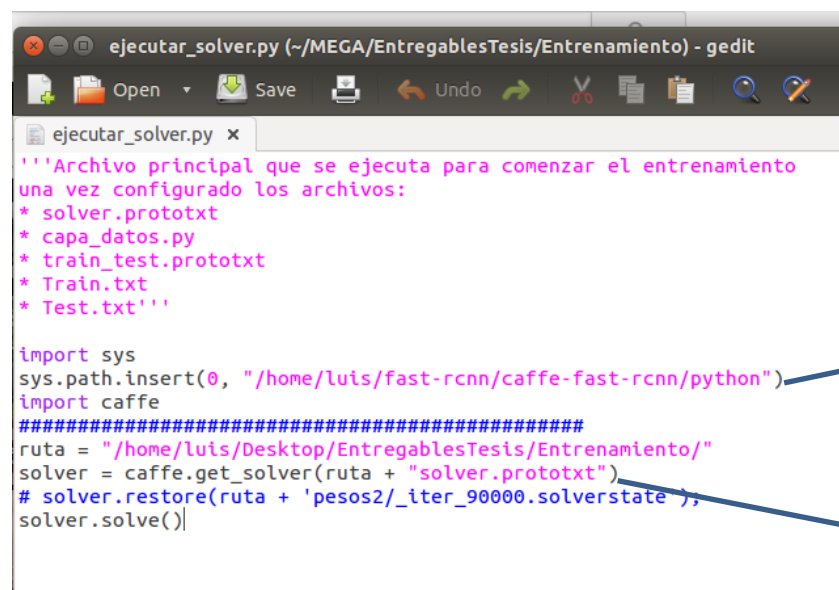
numero de imagen

rutaimagen

Clase sobrelapamiento x1 y1 x2 y2

Configuración ejecutar_solver.py

Una vez que se encuentran configurados todos los archivos mencionados anteriormente se tiene que ejecutar este archivo.



```
ejecutar_solver.py (~/MEGA/EntregablesTesis/Entrenamiento) - gedit
Open Save Undo Cut Copy Paste Find
ejecutar_solver.py x
'''Archivo principal que se ejecuta para comenzar el entrenamiento
una vez configurado los archivos:
* solver.prototxt
* capa_datos.py
* train_test.prototxt
* Train.txt
* Test.txt'''

import sys
sys.path.insert(0, "/home/luis/fast-rcnn/caffe-fast-rcnn/python")
import caffe

#####
ruta = "/home/luis/Desktop/EntregablesTesis/Entrenamiento/"
solver = caffe.get_solver(ruta + "solver.prototxt")
# solver.restore(ruta + 'pesos2/_iter_90000.solverstate '),
solver.solve()
```

Se añade la ruta de caffe al path.

Se obtiene el archivo solver.prototxt y se ejecuta para comenzar el