upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Human Detection

Trabajo Final

Universidad Pública de Navarra
Ingeniería Informática - Visión Artificial

Jhonny Fabricio, Chicaiza Palomo - https://github.com/jhonnyfc
David, Soto Arbizu - https://github.com/Kirtasth
Hanwei, Ke - https://github.com/Sendaia

Fecha 12 de enero de 2020

Índice

Link del Repositorio: https://github.com/jhonnyfc/Human_Detection_Project

Objetivo

El objetivo del proyecto es desarrollar un detector de personas en imágenes con Python.

Resumen

Para el desarrollo del proyecto se disponen de las librerías de Python que facilita la implementación del algoritmo, la información de estas funciones se puede encontrar en internet. Se ha dividido el proyecto en cuatro fases: preparación de datos, extracción de características, aprendizaje y detección.

Fases

Según el guion del trabajo hemos dividido el proyecto en cuatro fases principales:

i. **Preparación de datos.**

ii. **Fase de extracción de características.**

iii. **Fase de aprendizaje.**

iv. **Fase de detección.**

Preparación de datos:

Antes que nada, importamos todas las librerías que vamos a utilizar en este proyecto, por ejemplo: cv2, skimage, sklearn etc…

Se han guardado las fotos de Train y Test en un diccionario con la ayuda de la librería **os,** la cual nos permite leer todas las fotos con facilidad.

Al no haber la cantidad suficiente de fotos Train positivas, se ha creado una función para generar la foto espejo de la original, así se duplica el número de las fotos Train.

Se han aplicado unos **tratamientos** a las imágenes de Train antes de extraer sus características:

- **Cambio de dimensión de las fotos train positivas:** Se ha reducido el tamaño de las imágenes positivas a (128,64) para aprovechar al máximo sus características.

- **Recorte de las fotos train negativas:** Como el tamaño de las imágenes Train negativas son mucho más grande que las imágenes Train positivas, se ha escogido aleatoriamente en cada imagen train negativa bloques del tamaño de las Train positivas, para obtener el mismo número de características con HOG.

- **Ampliación del rango dinámico de las fotos:** Al ampliar el rango dinámico de las fotos se aumenta el rango de intensidades.

- **Transformación de intensidad:** Se ha utilizado la función potencia con exponente 0.35 para comprimir zonas claras y aclarar zonas oscuras.

- **Suavizado de las fotos:** El suavizado permite eliminar ruidos y detalles innecesarios en las imágenes.

Fase de extracción de características:

Para la extracción de características se ha elegido la función **HOG** de la librería skimage.feature, el HOG (histogram of oriented gradients). El algoritmo divide una imagen en porciones iguales(celdas) y cuenta las ocurrencias de las orientaciones del gradiente en cada celda.

Se han probado varios valores para los parámetros de la función **HOG**, al final se decidido poner 9 orientaciones, celdas de 8x8 y 2 celdas por bloque con ello se pueden utilizar el 100% de las características de una imagen debido a que se ha puesto la ventana en base 2.

Por último, se extraen las características de todas las fotos del conjunto de entrenamiento y se guarden en una matriz para su posterior uso, también se crea un array que guarda la clase de cada ejemplo (el array “y”).

Fase de aprendizaje:

El clasificador que hemos elegido es el **SVM** (Support Vector Machine), SVM es un algoritmo de aprendizaje supervisado que resuelve problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (Train) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra.

Dividimos nuestro conjunto original de Train en dos partes, un nuevo conjunto de Train y un conjunto de Val (validación) para comprobar si el clasificador es válido. De los 5166 ejemplos antes de permutar los ejemplos hemos decidido utilizar solo 3113 para que las clases estén más balanceadas.

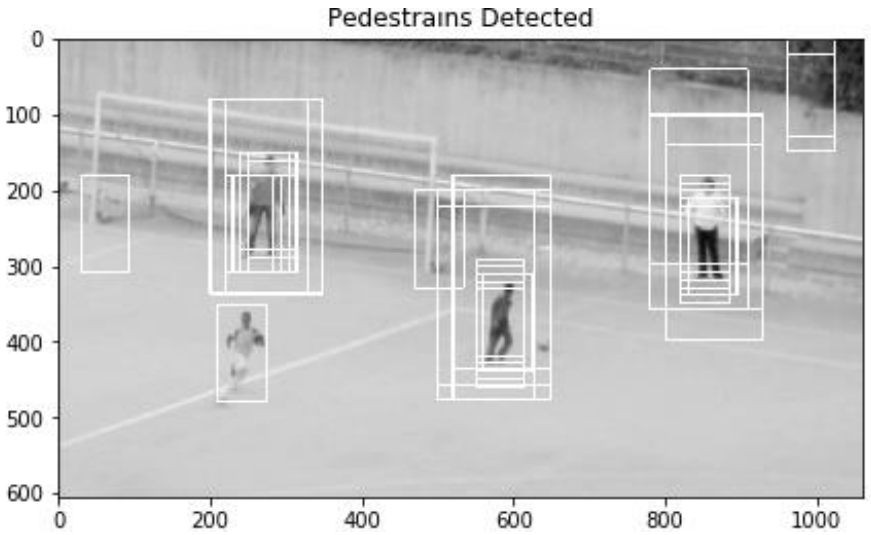
Hemos decidido utilizar el SVM con un Kernel lineal, debido a que ya tenemos suficientes características y además es mas rápido de entrenar. Por otra parte, tenemos que buscar un valor para C, este valor cuantifica el nivel de aprendizaje. Para ello hemos decidido crear una función que pruebe diferentes valores y devuelva el mejor clasificador según el conjunto de validación. Para comprobar si el entrenamiento ha sido efectivo calculamos el precisión de los conjuntos Train y Val (conjunto de validación) con la función score() del modelo:

precisión de los ejemplos de Train ~> 95%
precisión de los ejemplos de Val ~> 95%

La precisión es bastante buena, aunque depende de las permutaciones a la hora de la creación de los datos.

Fase de detección:

Tomamos una imagen del conjunto Test, para detectar si hay personas en ella o no hemos implementado una función de ventana deslizante (**slidWindow**) para extraer partes de la foto. La función **estudiaFoto**() Realiza la pirámide gaussiana de la imagen de entrada y aplica a cada nivel la función ventana deslizante (**slidWindow**), con ello podemos detectar las personas teniendo en cuenta su tamaño y la distancia.



Tenemos un problema de solapamiento ya que puede haber varios positivos en la misma persona, lo hemos solucionado haciendo uso de la función **SNoM** (Supresión No Máximos) que Elimina las predicciones que se solapan dejando solo una de ellas.

Si miramos los resultados obtenidos antes y después de la supresión de no máximos, podemos ver que mejora considerablemente después de eliminar el solapamiento.

Conclusiones

Hemos creado un algoritmo de aprendizaje automático para detectar personas en las imágenes, en el proceso hemos aprendido como usar nuevos algoritmos para la extracción de características y métodos de optimización. Se ha comprobado una significativa mejora usando un preprocesamiento de las imágenes para la mejor detección de bordes en el HOG. Se han probado diferentes técnicas de extracción de características como HU Moments entre otros.

Errores

Existe algunos falsos positivos en el resultado final como las paredes y objetos verticales, nuestro clasificador lo ha clasificado como una persona.
En la supresión no máximo dejamos un positivo para cada persona, pero no siempre elige el mejor positivo.
Se probo a utilizar un kernel Gaussiano para el SVM pero al tener tantos atributos hacia muy lento el aprendizaje.

Posibles mejoras

Como podemos ver nuestro clasificador no es perfecto, posiblemente se podría mejorar con más ejemplos de entrenamiento, mas calidad en las imágenes, o con técnicas que desconocemos.