



unl

Universidad
Nacional
de Loja

UNIVERSIDAD NACIONAL DE LOJA

FACULTAD DE LA ENERGÍA, LAS INDUSTRIAS Y

LOS RECURSOS NATURALES NO RENOVABLES

INGENIERA EN COMPUTACIÓN

ITINERARIO: SISTEMAS INTELIGENTES

Human Computer Vision

Avance sobre el Sistema de Detección Automática de Dorsales en Carreras y Maratones

AUTORES:

Diego Fernando Lojan Tenesaca

Angel Jahir Román Sánchez

Cecilia Fernanda Trueba Reyes

DOCENTE:

Ing. Luis Chamba-Eras Mg. Sc.

CICLO Y PARALELO:

Octavo “A”

LOJA – ECUADOR

2023 - 2024

1. Contexto del Sistema de Detección Automática de Dorsales en Carreras y Maratones.

Desarrollado por: Eric Bayless

Problemática

El objetivo es crear un modelo que pueda detectar números de dorsales de carreras de manera que demuestre su viabilidad en una aplicación móvil.

Adquisición de Datos

Los datos utilizados para este proyecto incluyen 217 imágenes que contienen 290 números de dorsales, divididos en 3 series de 3 carreras.

La mayoría de las imágenes son fotos de acción.

El conjunto de datos se conoce como el conjunto de datos de Reconocimiento de Números de Dorsales de Carreras (RBNR).

Además, se utilizó el conjunto de datos de Números de Casas de Street View (SVHN), que contiene más de 600,000 imágenes, para la detección de números.

Selección del Modelo

El modelo utilizado para este proyecto es YOLOv4-tiny, elegido por su velocidad, ya que es 8 veces más rápido que YOLOv4.

Resultados

- El conjunto de datos inicial de 127 imágenes se aumentó a 5,088 imágenes.
- La Precisión Media Promedio (mAP) a 0.5 aumentó del 76.03% al 94.42%.
- La precisión de detección de dígitos en el conjunto de datos RBNR fue del 67.59%, y la precisión de detección de extremo a extremo fue del 38.05%.

2. Explicación del código

El proyecto consta de dos clases principales: *Detector.py* y *streamlit_app.py*. *Detector.py* contiene la lógica necesaria para la detección de números, mientras que *streamlit_app.py* utiliza la librería Streamlit para crear una aplicación web sobre la detección de números.

❖ *Detector.py*

El archivo *Detector.py* contiene la lógica necesaria para la detección de números en dorsales de carreras utilizando modelos de aprendizaje profundo basados en YOLO.

El primer modelo está diseñado para detectar "bib" (bib numbers en carreras). El segundo modelo está configurado para leer números y está destinado a reconocer dígitos del 0 al 9. Cada modelo tiene su propia configuración de archivo de configuración y mejores pesos pre-entrenados.

```

# Bib detection model config
bd_configPath = 'Data/YOLO/bib_detector/RBNR2_custom-yolov4-tiny-detector.cfg'
bd_weightsPath = 'Data/YOLO/bib_detector/RBNR2_custom-yolov4-tiny-detector_best.weights'
bd_classes = ['bib']

# Number reader config
nr_configPath = 'Data/YOLO/num_reader/SVHN3_custom-yolov4-tiny-detector.cfg'
nr_weightsPath = 'Data/YOLO/num_reader/SVHN3_custom-yolov4-tiny-detector_best.weights'
nr_classes = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

➤ Clase Detector

La clase `Detector` se encarga de inicializar y utilizar modelos de detección de objetos basados en YOLO. Esta clase es capaz de detectar objetos de interés en imágenes dadas y devuelve las coordenadas de los cuadros delimitadores junto con las etiquetas de clase correspondientes.

```

class Detector:
    def __init__(self, cfg, wts, classes):
        # Inicializa el objeto detector con la configuración del modelo, los pesos y las clases.

    def detect(self, img, conf):
        # Realiza predicciones y devuelve las clases y cuadros delimitadores.

```

➤ Funciones Auxiliares

Además de la clase `Detector`, se proporcionan funciones auxiliares para realizar tareas específicas, como la detección de dorsales de carreras y la adición de anotaciones visuales a las imágenes detectadas.

```

def get_rbn(img, single=False):
    # Obtiene números de dorsales y cuadros delimitadores para dorsales detectados.

def annotate(img, annot, color):
    # Agrega números de dorsales y cuadros delimitadores a una imagen.

```

❖ streamlit_app.py

El archivo `streamlit_app.py` sirve como interfaz de usuario para interactuar con el sistema de detección implementado en `Detector.py`. Esta aplicación web ofrece tres modos de operación: "Demo", "Image" y "Video".

➤ Interfaz de Usuario

En la interfaz de usuario, los usuarios pueden cargar sus propias imágenes o videos para realizar la detección de números en tiempo real. Una vez que se realiza la detección, las imágenes o videos se muestran con las anotaciones correspondientes resaltando los números de los dorsales detectados.

```

import streamlit as st

# ...

st.title('Race Bib Number Detector')
st.sidebar.header("Mode")
mode = st.sidebar.radio(
    'Select Mode',
    options=['Demo', 'Image', 'Video']
)

# ...

if mode == 'Image':
    # Interfaz para la detección en imágenes.
elif mode == 'Demo':
    # Interfaz para la demostración con video pregrabado.
elif mode == 'Video':
    # Interfaz para la detección en videos.

```

➤ Uso del detector

El archivo *streamlit_app.py* utiliza la funcionalidad proporcionada por *Detector.py* para realizar la detección de números en dorsales de carreras y mostrar los resultados en la interfaz de usuario. *Detector.py* se utiliza principalmente en la sección de la interfaz de usuario donde se carga la imagen o el video y se realiza la detección de números. Esto se hace en los modos "Image" y "Video".

```

import Detector

if mode == 'Image':
    # Sección para la detección en imágenes.
    # Se utiliza Detector.py para realizar la detección en la imagen cargada por el usuario.
    output = Detector.get_rbn(img)

    # Anotar la imagen con los resultados de la detección.
    if output != None:
        for detection in output:
            img = Detector.annotate(img, detection, color)
    else:
        # Manejo cuando no se detectan números de dorsales.
        pass

elif mode == 'Video':
    # Sección para la detección en videos.
    # Se utiliza Detector.py para realizar la detección en el video cargado por el usuario.
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        # get bib prediction
        output = Detector.get_rbn(frame, single=True)

        # annotate image
        if output != None:
            frame = Detector.annotate(frame, output[0], color)

    # Resto del bucle para procesar el video.

```

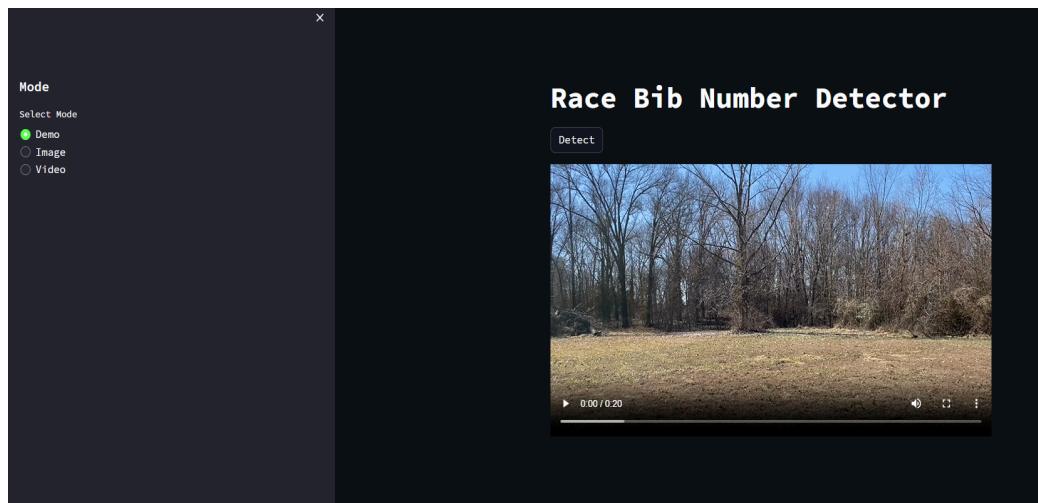
En estas secciones, se llama a las funciones definidas en *Detector.py* para realizar la detección de números en las imágenes o los fotogramas del video. Los resultados de la detección se utilizan luego para anotar las imágenes o los fotogramas con los números de dorsales detectados.

3. Instalación

- *Clonar el repositorio:* Primero se necesita clonar el repositorio en tu máquina local para esto se usa el siguiente comando:
 - git clone <https://github.com/ericBayless/bib-detector.git>
- *Ir al directorio:* Se debe ir al directorio bib-detector que se creó después de clonar el repositorio:
 - cd bib-detector
- *Instalar dependencias:* Este proyecto probablemente tenga dependencias externas, entonces deben ser instaladas:
 - pip install -r requirements.txt
- *Ejecutar la aplicación Streamlit:* Una vez clonado el repositorio e instaladas las dependencias, se ejecuta la aplicación Streamlit con el siguiente comando:
 - streamlit run streamlit_app.py

4. Ejecución y Pruebas

Una vez lanzada la aplicación, el objetivo principal radica en someterla a un análisis meticuloso para verificar su desempeño y funcionalidad, en primer paso podemos ver el orden en lo que se ejecuta la ventana web.



Podemos notar que hay una interfaz desarrollada en Streamlit que incluye un menú con tres opciones: "Demo", "Image", y "Video", en la sección de Demo, se carga automáticamente un archivo .mp4 que permite la detección de los números de dorsales de una persona en el video.



En las imágenes subsecuentes, se evidencia que el sistema logra identificar el número 28 en el video una vez que ha concluido el proceso de detección.

Avanzamos hacia la sección de imágenes, donde se nos presenta una interfaz que permite cargar imágenes con un límite máximo de 200 MB para su análisis posterior de detección.



Aquí podemos observar que el sistema identifica con precisión sólo algunos números, debido a la falta de definición en la imagen, este problema se presenta especialmente cuando las manos de los corredores obstruyen parte del dorsal.

Finalmente, en la sección de video, se realiza el mismo proceso que en la opción de demostración. Sin embargo, aquí se nos brinda la posibilidad de cargar un video de nuestra elección para evaluar el nivel de precisión en la detección de dorsales.



En este análisis, observamos que el sistema logra identificar con precisión el dorsal de un corredor en el video seleccionado, esta observación sugiere que el modelo exhibe una mayor eficacia cuando se trata de secuencias de video en comparación con imágenes estáticas, especialmente en situaciones donde la calidad de la imagen es baja o hay obstrucciones presentes.

Esta capacidad para mantener la precisión a lo largo de múltiples fotogramas resalta la robustez del sistema en entornos dinámicos, en consecuencia, podemos concluir que la efectividad del sistema de detección se ve potenciada en contextos de video, donde se pueden aprovechar mejor las características temporales y la continuidad de la acción para mejorar la precisión de las predicciones.