UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

"ESTIMACIÓN DE DENSIDAD DE MULTITUDES DE PERSONAS A TRAVÉS DE VISIÓN POR COMPUTADOR"

AUTOR: DANIEL DAVID ZAMBRANO ANDRADE

DIRECTOR: CARLOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR
NOVIEMBRE 2021

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL AUTOR			
CÉDULA DE IDENTIDAD	1003875125		
APELLIDOS Y NOMBRES	ZAMBRANO ANDRADE DANIEL DAVID		
DIRECCIÓN	Ecuador, Imbabura, Cotacachi, La Pradera		
EMAIL	ddzambranoa@utn.edu.ec - ddzambranoa@gmail.com		
TELÉFONO FIJO	062491093	TELÉFONO MÓVIL	0979178582
DATOS DE LA OBRA			
TÍTULO	“ESTIMACIÓN DE DENSIDAD DE MULTITUDES DE PERSONAS A TRAVÉS DE VISIÓN POR COMPUTADOR ”		
AUTOR	ZAMBRANO ANDRADE DANIEL DAVID		
FECHA	17/11/2021		
PROGRAMA	PREGRADO		
TÍTULO POR EL QUE OPTA	INGENIERO EN MECATRÓNICA		
DIRECTOR	CARLOS XAVIER ROSERO CHANDI		

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CONSTANCIA

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto, la obra es original, y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 17 días del mes de noviembre de 2021

Daniel David Zambrano Andrade
C.I.: 1003875125

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado "ESTIMACIÓN DE DENSIDAD DE MULTITUDES DE PERSONAS A TRAVÉS DE VISIÓN POR COMPUTADOR", presentado por el egresado DANIEL DAVID ZAMBRANO ANDRADE, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, a los 17 días del mes de noviembre de 2021

Carlos Xavier Rosero Chandi
DIRECTOR

Agradecimiento

Agradezco principalmente a Dios, por permitirme seguir mi camino y aprender cada día algo nuevo, porque sólo por él pude llegar a culminar esta etapa de mi vida, ya que sus planes son buenos, agradables y perfectos.

Agradezco de una manera inexplicable a mi madre, Mariana Zambrano por su amor y apoyo incondicional, por sostenerme anímica y económicamente cada día de mi vida, ya que es seguro que sin su apoyo y lucha constante no hubiese conseguido lo que hoy he logrado.

Agradezco con mucho amor a Madelaine Reascos, quien estuvo conmigo en los momentos más difíciles de mi vida, A pesar de todo, con mucho amor y paciencia me motivó y ayudó para ser lo que ahora soy, y no hay palabras para expresar todo lo que significas para mí.

Un agradecimiento sincero a Carlos Xavier Rosero Chandi por su esfuerzo y dedicación, quien, con sus conocimientos, su experiencia, su paciencia y su tiempo aportaron a la realización de este trabajo de grado.

Daniel David Zambrano Andrade

Dedicatoria

Este logro lo dedico al pilar de mi vida, mi madre Mariana Zambrano, quien es un ejemplo de lucha y perseverancia, quien ha vencido a cualquier adversidad y a pesar de las circunstancias difíciles nunca ha perdido su fe.

Daniel David Zambrano Andrade

Resumen

La estimación del número de personas generalmente es desempeñada de forma manual, pero las aglomeraciones de personas son un problema por su densidad y es normal dar resultados que subestimen la realidad. Por lo tanto, se opta por automatizar el proceso por medio de sistemas más exactos de visión por computador, siendo la base para poder implementar sistemas de videovigilancia más inteligentes con la capacidad de monitorear aglomeraciones.

La solución propuesta para este trabajo se enfoca en la implementación de la red neuronal convolucional CSRNet (Congested Scene Recognition Network). La arquitectura permite la extracción de características bidimensionales para generar mapas de densidad y la aproximación del número de personas.

La arquitectura es implementada en software libre aprovechando las funcionalidades de la biblioteca de aprendizaje automático PyTorch. Los resultados obtenidos se muestran en una interfaz gráfica diseñada a través del conjunto de herramientas de interfaz PyQt5. Los datos se almacenan en un historial de consultas realizado a través del motor de base de datos SQLite.

Finalmente, mediante las pruebas realizadas al sistema, se garantiza el funcionamiento de este proyecto empleando la dimensión estándar de 1024 x 768 e imágenes a color con posición de la cámara en ángulo en picado o ángulo en cenital. Además, el peso de la imagen debe estar aproximado a 138 kB para evitar la saturación de la unidad de almacenamiento, de esta forma da validez al sistema propuesto.

Abstract

The estimation of the number of people is generally performed manually, but crowds of people are a problem due to their density and it is normal to give results that underestimate reality. Therefore, it is opted to automate the process through a by means of more exact computer vision systems, being the basis to be able to implement smarter video surveillance systems with the ability to monitor agglomerations.

The solution proposed for this work focuses on the implementation of the convolutional neural network CSRNet (Congested Scene Recognition Network). The architecture allows the extraction of two-dimensional features to generate density maps and the approximation of the number of people.

The architecture is implemented in free software taking advantage of the functionalities of the PyTorch machine learning library. The results obtained are shown in a graphical interface designed through the PyQt5 interface tool set. The data is stored in a query history made through the SQLite database engine.

Finally, through the tests carried out on the system, the operation of this project is guaranteed using the standard size of 1024 x 768 and color images with the camera position at a dive angle or a zenith angle. In addition, the weight of the image must be approximately 138 kB to avoid saturation of the storage unit, thus giving validity to the proposed system.

Índice general

Índice de figuras

Índice de tablas

Lista de Programas

Introducción

La densidad de una aglomeración de personas es difícil de estimar manualmente y es una práctica regular dar resultados que subestimen la realidad. Una opción más exacta consiste en la automatización de este proceso a través de sistemas de visión por computador.

En este trabajo se presenta la implementación de un sistema de estimación de densidad de multitudes de personas a través de visión por computador, que emplea la arquitectura CSRNet (Congested Scene Recognition Network). Además, utiliza aprendizaje profundo para extraer características bidimensionales de imágenes y generar un mapa de densidad con escalamiento y filtrado para su presentación final.

Como requisitos de funcionamiento del sistema se han considerado las dimensiones máximas y mínimas, la unidad de procesamiento gráfico del computador utilizado, además de que la capacidad de la arquitectura se vea reflejada en el formato de imagen.

La arquitectura es implementada en software libre aprovechando las funcionalidades de la biblioteca de aprendizaje automático PyTorch. Los resultados obtenidos se muestran en una interfaz gráfica diseñada a través del conjunto de herramientas de interfaz PyQt5. Los datos se almacenan en un historial de consultas realizado a través del motor de base de datos SQLite.

La demostración del sistema se realiza mediante la comparación de los resultados obtenidos de las métricas de rendimiento, la estimación del número de personas y mapas de densidad generados por cada modelo entrenado de la red neuronal convolucional en el conjunto de datos de ShanghaiTech.

Problema

La reciente pandemia denominada Covid-19 [1], ha mostrado muchas falencias a la hora de evitar contagios masivos pero las aglomeraciones de personas incrementan la probabilidad de la propagación de esta pandemia [2]. Ecuador ha presentado protocolos que intentan frenar el avance del Covid-19 [3], pero las aglomeraciones muestran falta de control e irresponsabilidad de la ciudadanía. Aunque existen mecanismos de vigilancia y monitoreo del Servicio Integrado de Seguridad ECU-911 [4], estos son inservibles al momento de dar datos sobre multitudes.

La estimación del número de personas generalmente es desempeñada de forma manual, pero las multitudes congestionadas son un problema por su densidad. Por tanto, en la búsqueda de una solución se opta por automatizar el proceso por medio de sistemas de visión por computador [5].

La aplicación de visión por computadora en el área de análisis de multitudes ha dado como resultado varias soluciones para la estimación de personas. Sin embargo, la aplicación práctica de la mayoría de las técnicas existentes tiene limitaciones importantes como la incapacidad para manejar multitudes de cientos o miles [6]. En la última década, se abordó las limitaciones del análisis de multitudes con un aprendizaje profundo de los modelos basados en redes neuronales convolucionales para obtener mapas de densidad mejorando la capacidad de estimar el número de personas en multitudes densas [7].

Desde la visión artificial a través de cámaras instaladas se puede estimar la densidad de multitudes de personas empleando redes neuronales convolucionales [8], siendo la base para poder implementar sistemas de videovigilancia más inteligentes con la capacidad de monitorear aglomeraciones.

Objetivos

Objetivo general

Estimar el número de personas sobre imágenes de multitudes empleando redes neuronales convolucionales.

Objetivos específicos

- Identificar la arquitectura de la red neuronal convolucional referente a la estimación de densidades de multitudes de personas.
- Desarrollar el sistema de estimación de personas en software libre.
- Verificar el funcionamiento de la propuesta mediante simulación.

Justificación

Hoy en día los sistemas de vigilancia y monitoreo tradicional han mostrado dificultades cuando el número de cámaras excede a las capacidades de los operadores humanos para monitorizarlas, por lo cual, se intenta potencializar estos equipos de vigilancia con técnicas de visión por computadora para describir y comprender imágenes [5].

La importancia de este trabajo es aportar una estimación de personas en imágenes multitudes que pueda contribuir en la calidad de vida en diferentes ámbitos de la sociedad, tales como la gestión de multitudes durante o post pandemia y diseño de lugares públicos con normas ambientales y de seguridad.

La idea fundamental de los sistemas actuales de estimación de multitudes es implementar redes neuronales convolucionales para capturar características de alto nivel para generar mapas de densidad de alta calidad sin expandir brutalmente la complejidad de la red [9], para proveer una estimación del número de personas en un lugar aglomerado.

Alcance

El sistema propuesto estimará el número de personas en imágenes de multitudes mediante la generación de mapas de densidad basado en redes neuronales convolucionales. El modelo empleará capas convolucionales para admitir imágenes de entrada con resoluciones flexibles de tipo RGB. Se usará un conjunto de datos de imágenes para el entrenamiento y validación de la red neuronal convolucional, este sistema no es en tiempo real y no incluye un prototipo físico.

Capítulo 1

Revisión Literaria

La estimación es un proceso de aproximar una medida incluso si los datos de origen están incompletos, inestables o inciertos [10]. En cuanto a la estimación del número de personas en multitudes, es un problema difícil de resolver y es normal dar de manera subestimada la cantidad de personas según la perspectiva humana.

Hoy en día con el fin de mejorar la aproximación del número de personas, la visión por computador es de gran ayuda por los conjuntos de algoritmos y estrategias que tienen la capacidad de procesar conjuntos de imágenes.

1.1. Métodos de estimación de personas

- La estimación basada en detección emplea un detector similar a una ventana móvil para detectar y aproximar el número de personas, por lo que requiere detectores HOG (Histograma de Gradientes Orientados) y clasificadores SVM (Máquinas de Vectores de Soporte) entrenados para extraer características de bajo nivel de todo el cuerpo humano [11].

- La estimación basada en regresión aprende de la relación entre las características de primer plano y texturas extraídas de los parches de las imágenes para generar información de bajo nivel y luego calcular la estimación del número de personas [12].

- La estimación basada en densidad considera la información espacial de las imágenes para la generación de mapas de densidad a nivel de píxeles, de tal modo que extraen características de bajo nivel para estimar el recuento de objetos en cualquier región de una imagen [13].

- Los anteriores métodos mencionados permiten la estimación de personas, sin embargo, su rendimiento es deficiente en escenas muy congestionadas, ya que la mayoría de las personas están ocultas. Para abordar este problema, los investigadores detectan partes particulares del cuerpo en lugar de todo el cuerpo para completar el análisis de escenas de multitudes.

Siguiendo la idea propuesta por Li *et al.* [9], la estimación basada en redes neuronales convolucionales permite extraer características bidimensionales de imágenes para el reconocimiento de formas complejas, generación de mapas de densidad y el recuento del número de personas.

1.2. Revisitando las redes neuronales

1.2.1. Redes neuronales artificiales

Las redes neuronales artificiales (ANN, por sus siglas en inglés) son un sistema adaptativo que se enfoca en imitar las funciones cerebrales del ser humano, mediante elementos de procesamiento conocidos como nodos o neuronas que se enlazan entre sí por medio de conexiones que tiene una fuerza variable conocida como peso [14].

De manera general, las neuronas artificiales se definen como:

$$y = \varphi(b + \sum_{i=1}^{m} x_i w_i) \tag{1.1}$$

Donde, la principal función de una neurona artificial consiste en multiplicar cada valor de entrada (x), por la fuerza de las conexiones (w), y su resultante se suma a un valor de umbral o bias (b), y según la función de activación (φ), la neurona permitirá activar la salida (y), como se presenta en la Figura 1.1.

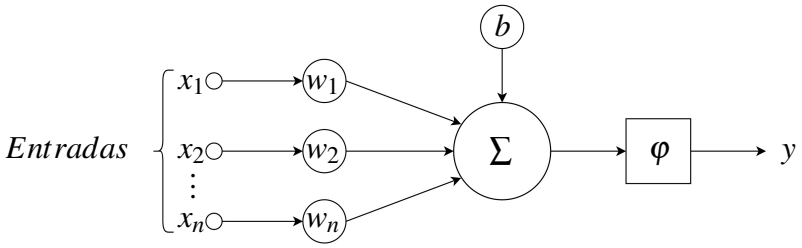


Figura 1.1: Nodo o neurona artificial [15].

Un sistema neuronal está compuesto por neuronas apiladas verticalmente y agrupadas en capas, como se presenta en la Figura 1.2. La capa de entrada representa a los valores ingresados en la red neuronal. A continuación, las capas ocultas realizan transformaciones a los valores de entrada según la Ecuación 1.1. Finalmente, la capa de salida unifica la información recibida de la última capa oculta.

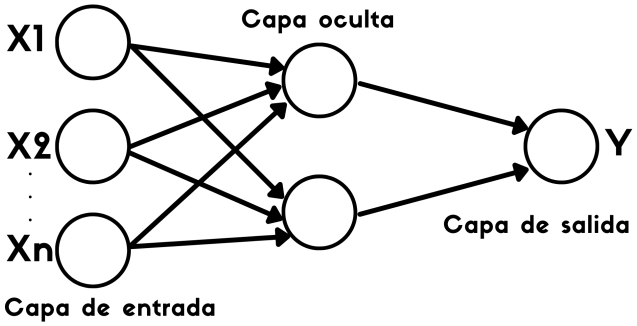


Figura 1.2: Sistema neuronal artificial [16].

1.2.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) están diseñadas para trabajar con imágenes de entrada, empleando capas de convolución se realizan las operaciones matemáticas de la Ecuación 1.1 mientras escanean los valores de entrada mediante filtros que extraen valores específicos y devolviendo mapas de características.

Por otra parte, la convolución dilatada introduce un parámetro conocido como tasa de dilatación, lo cual define un espacio en el kernel, como se presenta en la Figura 1.3. Es decir, extrae características profundas que la convolución normal no puede detectar.

De manera general, la convolución se define como:

$$y(m,\ n) = \sum_{i=1}^{M} \sum_{j=1}^{N} x(m + r * i, n + r * j) w(i,\ j) \tag{1.2}$$

Donde, $y(m,\ n)$ es la salida, $x(m,\ n)$ es la entrada de valores, $w(i,\ j)$ es el kernel donde M y N es su longitud y ancho respectivamente, r es la tasa de dilatación, siendo $r = 1$ para convolución normal y $r >= 2$ para convolución dilatada.

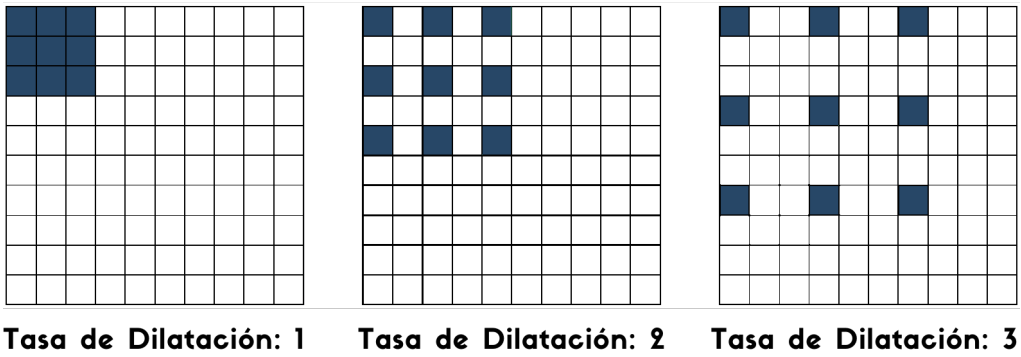


Figura 1.3: Kernel de 3x3 con diferente tasa de dilatación [9].

1.2.3. Representación de imágenes digitales

La imagen digital es una representación bidimensional de una imagen analógica basada en conjuntos de números que pueden ser almacenados y manejados por una computadora [17].

Las imágenes digitales se clasifican en:

- **Imágenes vectoriales. -** Este tipo de imágenes se basan en fórmulas matemáticas que representan a entidades geométricas.
- **Imágenes de mapa de bits. -** Este tipo de imágenes se representan mediante pixeles donde cada píxel describe un color.

Tensores. - Son una estructura de datos que almacenan valores numéricos, por ejemplo, los conjuntos de píxeles que contienen las imágenes. Además, los tensores en función de la aplicación pueden ser de diferentes dimensiones, como se presenta en la Figura 1.4 [18].

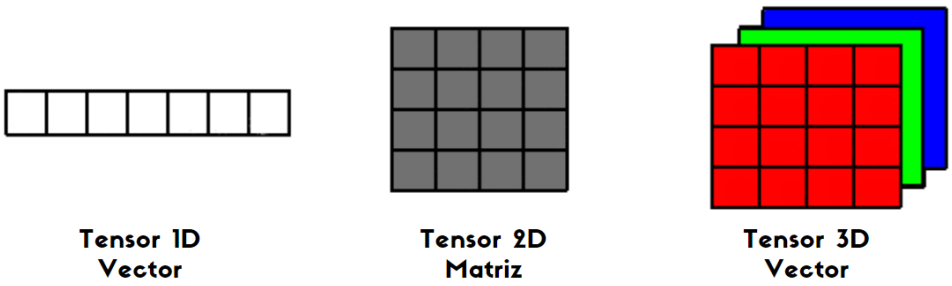


Figura 1.4: Tensores de diferentes dimensiones.

Ángulos en imágenes. - Son la inclinación respecto a una línea imaginaria o respecto a la zona donde se realiza la fotografía, como se presenta en la Figura 1.5.

- **Ángulo a nivel o normal. -** Este ángulo se encuentra ubicado a la misma altura que el nivel de las cabezas de multitudes. Este ángulo se caracteriza por no tener inclinación.
- **Ángulo en picado. -** Este ángulo se ubica en un nivel superior enfocando a las cabezas. Su rango de inclinación es mayor a 0° e inferior a 90°
- **Ángulo en contrapicado. -** Este ángulo está ubicado en la parte inferior respecto al nivel de las cabezas. Su rango de inclinación es mayor a 0° e inferior a 90°
- **Ángulo en cenital. -** Este ángulo se enfoca desde la parte superior perpendicular al suelo. Su rango de inclinación es igual a 90°
- **Ángulo en nadir. -** Este ángulo se enfoca desde el suelo en dirección perpendicular a la parte superior. Su rango de inclinación es igual a 90°

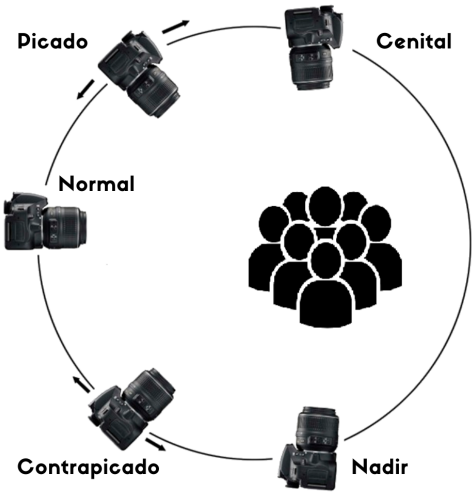


Figura 1.5: Ángulos de fotografía.

1.2.4. Conjuntos de datos

Conjunto de datos (Dataset) son una recopilación de patrones tabulados que alimentan a un modelo de red neuronal durante la fase de aprendizaje.

Los conjuntos de datos se clasifican en:

- **Conjunto de aprendizaje. -** Este conjunto permite entrenar a la red neuronal ajustando parámetros para reducir el error de resultado.
- **Conjunto de validación. -** Este conjunto proporciona una evaluación imparcial del ajuste de parámetros de un modelo durante la fase de aprendizaje.
- **Conjunto de pruebas. -** Este conjunto permite comprobar el modelo de la red neuronal con imágenes no utilizadas al finalizar el aprendizaje.

1.2.5. Métricas de rendimiento

Siguiendo la idea propuesta por Li *et al.* [9], las métricas de rendimiento permiten medir la precisión del modelo entrenado respecto al conjunto de datos utilizado. Se definen matemáticamente por el error absoluto medio y el error cuadrado medio expresado en las siguientes ecuaciones:

- **Error Absoluto Medio:**

  El error absoluto medio (MAE, por sus siglas en inglés) es el promedio de los valores absolutos en la diferencia entre el conteo real y la estimación de personas de todas las imágenes que contiene un conjunto de datos.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|C_i - C_i^{GT}| \tag{1.3}$$

- **Error Cuadrado Medio:**

  El error cuadrado medio (MSE, por sus siglas en inglés) es la medida de dispersión de datos, es decir, indica qué tan dispersa es la diferencia entre el conteo real y la estimación de personas con respecto a la media. Mientras mayor sea el error cuadrado medio, mayor será la dispersión de los datos.

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|C_i - C_i^{GT}|^2} \tag{1.4}$$

Donde N es el número de imágenes, C_i^{GT} es el conteo real, C_i representa a la estimación de conteo la cual se define matemáticamente como:

$$C_i = \sum_{l=1}^{L}\sum_{w=1}^{W} z_{l,w} \tag{1.5}$$

L y W es longitud y ancho y $z_{l,\,w}$ son los pixeles en $(l,\ w)$ del mapa de densidad.

1.3. Arquitecturas de redes neuronales

- Very Deep Convolutional Networks for Large-Scale Image Recognition es una arquitectura de red neuronal convolucional con diversas configuraciones entre las más destacables se encuentra la configuración VGG-16, la cual está compuesta de 16 capas de convolución con parámetros modificables para la extracción de características bidimensionales en imágenes [19].

- Multi-Column Convolutional Neural Network (MCNN, por sus siglas en inglés) es una arquitectura multicolumna que tiene como objetivo estimar el número de personas y generar mapas de densidad a partir de imágenes de multitudes arbitrarias, empleando filtros con campos receptivos variantes adaptándose a diferentes tamaños de las cabezas generado por el efecto de perspectiva de la imagen [20].

- Perspective-Guided Convolution (PGCNet, por sus siglas en inglés) es un modelo de red neuronal convolucional que utiliza información de las perspectivas en imágenes para resolver el problema de la variación de las escalas dentro de las escenas, por lo cual adopta como línea base a otra arquitectura que alimente la resolución de perspectiva que necesita el modelo PGCNet para generar un mapa de densidad y la estimación del recuento de personas [21].

- Structure-Aware Network + Spatial Awareness Network (SANet+SPANet, por sus siglas en inglés) es una arquitectura que propone capturar las variaciones espaciales al encontrar la subregión de pixeles. Además, con el marco propuesto integra a arquitecturas existentes para mejorar significativamente las métricas rendimiento de las líneas base [22].
- Adaptive Dilated Self-Correction Network (ADSCNet, por sus siglas en inglés) es una red neuronal convolucional que emplea dilatación adaptativa y marco de aprendizaje supervisado autocorrectivo para la predicción de un valor dinámico y continuo de la tasa de dilatación, optimizando el conteo de personas y la uniformidad en la generación del mapa de densidad [23].
- Congested Scene Recognition Network (CSRNet, por sus siglas en inglés) es un modelo de reconocimiento de escenas de multitudes densas y dispersas que puede estimar la cantidad de personas y generar un mapa de densidad de resolución de 1/8 con relación a la imagen original de entrada. El modelo emplea una red neuronal convolucional que extrae características bidimensionales y una red neuronal convolucional dilatada, que expande el campo de recepción y extrae características profundas que la convolución normal no logra detectar [9].

1.4. Propuesta

Las arquitecturas de redes neuronales mencionadas en este capítulo necesitan de líneas base para su desarrollo, por lo cual, este trabajo de grado se centrará en la implementación de la arquitectura base CSRNet en software libre aprovechando las funcionalidades de bibliotecas de aprendizaje automático. Los resultados obtenidos se mostrarán en una interfaz gráfica diseñada a través de un módulo de interfaz de usuario para la creación de aplicaciones de computador. Los datos se almacenarán en un historial de consultas realizado a través de un motor de base de datos.

Capítulo 2

Metodología

La investigación que se ha realizado en este trabajo es de tipo aplicación tecnológica, ya que se enfoca en la solución de problemas directos de la sociedad, en este caso, aglomeraciones de personas. Para ello se propuso diseñar un sistema para observar mapas de densidad y el número aproximado de personas con el fin de conseguir una situación de control de multitudes.

2.1. Descripción del sistema

Este sistema se diseña con el propósito de obtener mapas de densidad y la estimación del número de personas en imágenes, mediante la implementación de una arquitectura neuronal, la cual se encarga de procesar las imágenes. Los resultados se visualizan en una interfaz gráfica y se almacena la información en una base de datos.

El sistema debe cumplir con las siguientes características:

- Estar desarrollado en software libre.
- Admitir imágenes RGB.
- Proporcionar mapas de densidad y el número estimado de personas.
- Almacenar y visualizar la información.

2.2. Descomposición funcional

El sistema propuesto, de manera general, se desarrolla a partir de tres elementos principales: Datos de entrada, red neuronal convolucional y datos de salida, como se presenta en la Figura 2.1.

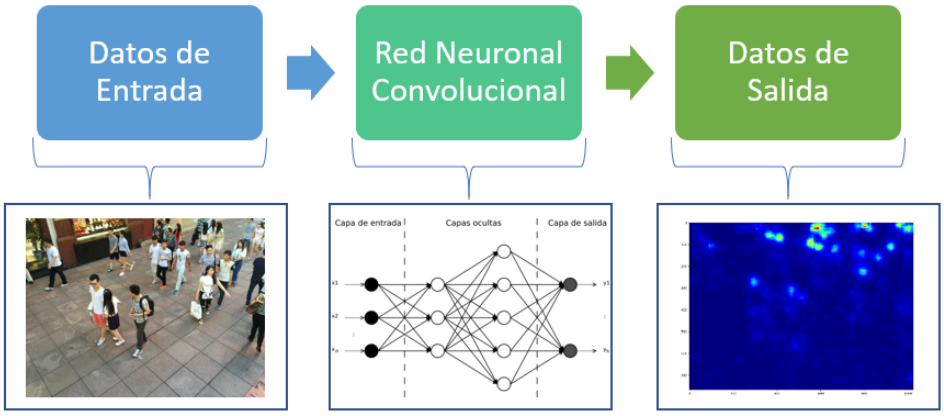


Figura 2.1: Descomposición funcional.

El primer elemento del diagrama comienza con la entrada de datos, esta sección representa a las imágenes de personas en multitudes. Las cuales deben ser seleccionadas y adaptadas para la arquitectura neuronal, proporcionando la información necesaria como son las anotaciones del centro de cabezas. La condición mínima establecida para selección del conjunto de datos es el formato de las imágenes especificado como RGB.

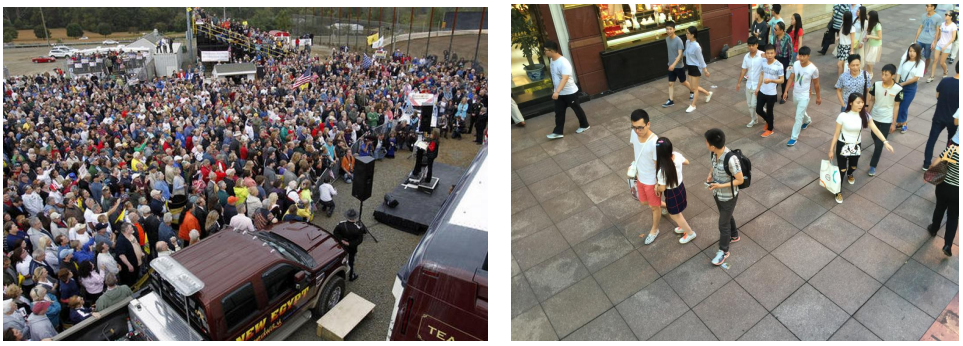
La red neuronal convolucional es el elemento fundamental del sistema, debido a que, de su arquitectura se procesará los datos de entrada hasta la generación de la salida. Esta sección contiene la arquitectura CSRNet con su respectivo tipo de tensores, valores de normalización y parámetros del modelo y entrenamiento.

La salida de datos hace referencia a los de mapas de densidad y la aproximación de personas. Esta sección se visualizará en una interfaz gráfica y almacenará en una base de datos los resultados obtenidos del diseño propuesto para comprobar su funcionamiento.

2.3. Selección del conjunto de datos

ShanghaiTech Dataset es el conjunto de datos más utilizado por las arquitecturas neuronales para el conteo de multitudes contiene dos partes con 1198 imágenes con 330,165 anotaciones del centro de la cabeza. Este conjunto de datos se encuentra disponible en el servidor Kaggle [20].

- **ShanghaiTech Parte A. -** Este subconjunto contiene 482 imágenes de aglomeraciones de alta densidad seleccionadas de Internet aleatoriamente (Fig. 2.2a).
- **ShanghaiTech Parte B. -** Este subconjunto contiene 716 imágenes de aglomeraciones de baja o media densidad tomadas en las zonas más concurridas de la ciudad de Shanghái (Fig. 2.2b).



a) b)

Figura 2.2: a) Multitud de alta densidad, b) Multitud de media densidad [20].

El conjunto de datos de ShanghaiTech aborda a las imágenes de multitudes de baja, media y alta densidad por separado, por lo cual se crea una carpeta con una combinación de estas para analizar posteriormente sus resultados a partir de los modelos a entrenar. En la Figura 2.3 se presenta un árbol de directorios de la organización de las carpetas y su contenido.

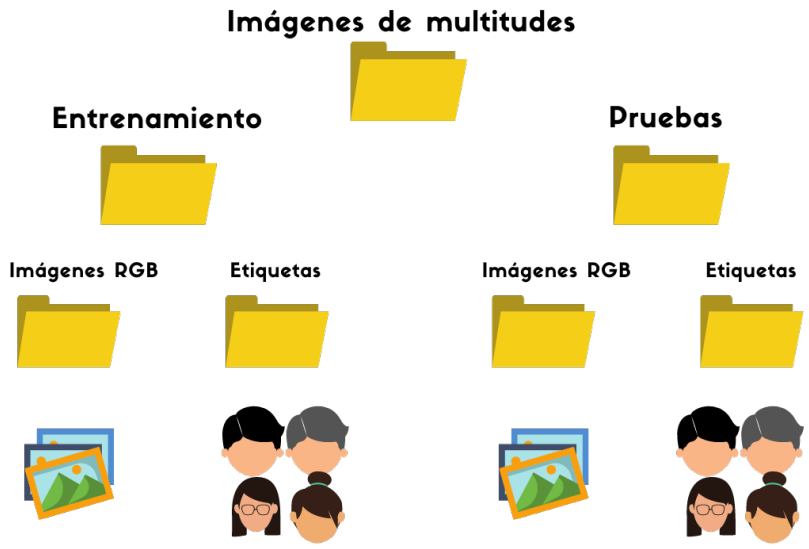


Figura 2.3: Árbol de directorios.

Para cada multitud se crea 2 archivos con imágenes aleatorias y no repetitivas en formato JSON de la carpeta de entrenamiento, el cual permite intercambio de información de forma sencilla. El primer archivo contiene un 80 % de imágenes y se emplean para el aprendizaje de los modelos, el segundo archivo contiene el 20 % restante que permite la validación del modelo durante el entrenamiento.

2.4. Red neuronal convolucional

La configuración de la arquitectura del modelo CSRNet está basada en la red neuronal VGG-16, como se presenta en la Tabla 2.1. Esta arquitectura se divide en dos partes conocidas como frontend y backend, detallados a continuación:

Parámetros de la red neuronal CSRNet

- **Pesos. -** Durante la primera época de entrenamiento es igual a 0, cuando la red neuronal cambia a la siguiente época el peso se define mediante las anotaciones del conjunto de datos empleado y la red VGG-16.
- **Bias.-** Al igual que el peso durante la primera época el bias tiene una constante igual a 0 pero cambia cuando empieza la siguiente época y se mantiene en una constante igual a 1.
- **Kernel Gaussiano. -** Escanea los píxeles de la imagen según el movimiento de stride difuminando la cabeza de cada persona.
- **Stride. -** Modifica el movimiento del kernel definido igual a 2 para evitar la pérdida y saturación de información.
- **Frontend**
  - **Pooling. -** Las operaciones de Max-Pooling extraen el máximo valor escaneado en el kernel para reducir progresivamente el tamaño espacial de las imágenes logrando disminuir los cálculos excesivos e innecesarios.
  - **Función de activación ReLU. -** ReLU funciona de tal forma que, si la entrada tiene un valor negativo, este se convierte en cero, caso contrario si el valor de entrada es positivo devuelve el mismo valor a la salida.
  - **Dilatación. -** Frontend abarca todo lo referente a convolución normal por lo que se define a la tasa de dilatación igual a 1.
- **Backend**
  - **Dilatación. -** Se define la tasa de dilatación igual a 2, esto permite extender el Kernel para la búsqueda de información oculta o que no encontró la convolución normal.
- **Método forward. -** Mantiene la dirección en un solo sentido, es decir, cuando empieza el análisis de una imagen la información ingresada recorrerá toda la red neuronal convolucional hasta la capa de salida.
- **Transferencia de aprendizaje. -** Permite mejorar la eficiencia de aprendizaje de un modelo empleando una red neuronal convolucional pre-entrenada para transferir información previamente adquirida para el aprendizaje de nuevas funcionalidades.

Tabla 2.1: Configuración tipo B CSRNet.

Conv(Tamaño de kernel)-(Numero de filtros)-(Tasa de dilatación)
Número de Capas: 18
Entrada: Imagen RGB
Front-end (Base de VGG-16)
conv3-64-1
conv3-64-1
max-pooling
conv3-128-1
conv3-128-1
max-pooling
conv3-256-1
conv3-256-1
conv3-256-1
max-pooling
conv3-512-1
conv3-512-1
conv3-512-1
Back-end (Modificación de VGG-16)
conv3-512-2
conv3-512-2
conv3-512-2
conv3-256-2
conv3-128-2
conv3-64-2
conv1-1-1

2.4.1. Tensor

Como se mencionó en la sección 1.2.3, existen diferentes tipos de estructura de tensores, pero, teniendo en consideración la composición de una imagen con los colores rojo, verde y azul, como se presenta en la Figura 2.4. Se emplea PyTorch para realizar la transformación de la estructura de datos de píxeles a tensores 3D, esto permite que los valores que contengan cada canal puedan ser normalizados. La biblioteca de aprendizaje profundo se encuentra mediante la instalación de PyTorch disponible en línea [24]. Se optimiza los cálculos mediante una tarjeta gráfica NVIDIA y la plataforma CUDA para acelerar el procesamiento de las imágenes, esta plataforma está disponible mediante la instalación de CUDA [25].

Figura 2.4: Imagen RGB.

2.4.2. Normalización

La normalización ayuda a que los valores que están en la nueva estructura de datos puedan definirse en un rango específico, reduciendo la asimetría de las imágenes, esto permite mejorar el aprendizaje durante la fase de entrenamiento del modelo y para comprobación con nuevas imágenes que no estén en el conjunto de datos.

La normalización convierte los píxeles del rango $(0,\ 255)$ al rango $(0,\ 1)$ en el tensor, por lo cual para imágenes RGB es necesario obtener la media y desviación estándar del conjunto de datos. PyTorch necesita tres datos de normalización para la media y la desviación estándar con la forma $(C\ x\ H\ x\ W)$ donde C es el número de canales, H es la altura y W es el ancho de la imagen, pero, PyTorch ofrece valores estandarizados para la normalización de imágenes solo si se emplea una red pre-entrenada como es el caso de VGG-16 y un conjunto de datos conocido.

A continuación, los valores estandarizados para normalización por PyTorch:

Desviación Estándar = [0.229, 0.224, 0.225]
Media = [0.485, 0.456, 0.406]

2.4.3. Entrenamiento del modelo

La fase de aprendizaje tiene como objetivo buscar los pesos de las neuronas que ofrezcan mejores resultados evitando el sobre aprendizaje y generando el menor error posible entre la salida obtenida y la salida esperada.

- **Tasa de aprendizaje. -** Se define una taza de aprendizaje constante igual a $1e^{-6}$ como velocidad para el paso de cada interacción de la red neuronal y los datos de entrada.
- **Épocas. -** Las épocas son un parámetro sujeto a la experiencia de los desarrolladores de redes neuronales convolucionales basado en ensayos y errores, por lo cual, en función de replicar los valores obtenidos del autor del modelo de CSRNet en las métricas de rendimiento se establece un valor de 150 épocas.

Al finalizar el entrenamiento se genera un archivo comprimido *tar* el cual incluye el modelo entrenado con extensión *pth* de PyTorch, el cual almacena los pesos de las neuronas con el que se obtuvo mejor rendimiento. Este archivo puede ser llamado desde Python para realizar el procesamiento de imágenes nuevas sin tener la necesidad de volver a entrenar el modelo.

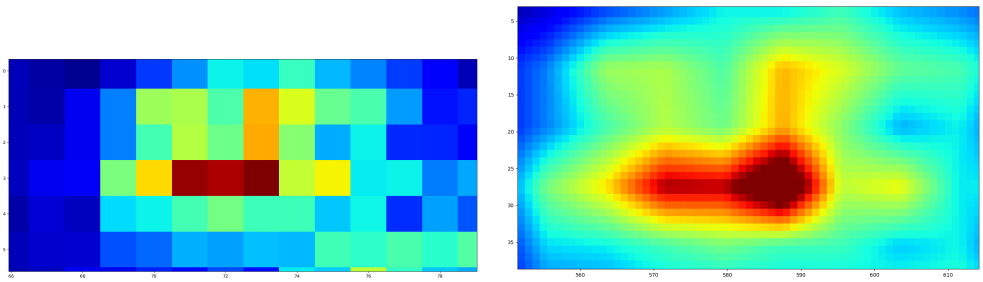
2.5. Datos de salida

Los patrones generados por la arquitectura neuronal en búsqueda de formas complejas como son las cabezas de las personas permiten contar y difuminar cada detección, estos valores son almacenados internamente hasta la salida devolviendo la suma total personas y el mapa de densidad.

Como se mencionó en la Sección 1.3, el modelo CSRNet genera a su salida un mapa de densidad con resolución de 1/8 con relación a la imagen original, por lo cual, la lectura de este mapa de densidad depende de las dimensiones H y W de la imagen de entrada.

Para mejorar la lectura del mapa de densidad se emplea una nueva escala multiplicando inversamente la reducción de la salida original, pero, la lectura del mapa de densidad aun sería ilegible (Fig. 2.5a), por lo cual se añade suavizado de contorno o antialiasing (Fig. 2.5b) y así reacondicionar el mapa anterior y generar un nuevo mapa de densidad. La biblioteca para redimensionar y aplicar antialiasing en imágenes se encuentra disponible mediante la instalación Scikit ejecutando en la consola el siguiente comando:

```
pip3 install scikit-image        % Biblioteca para escalar y suavizar imágenes.
```



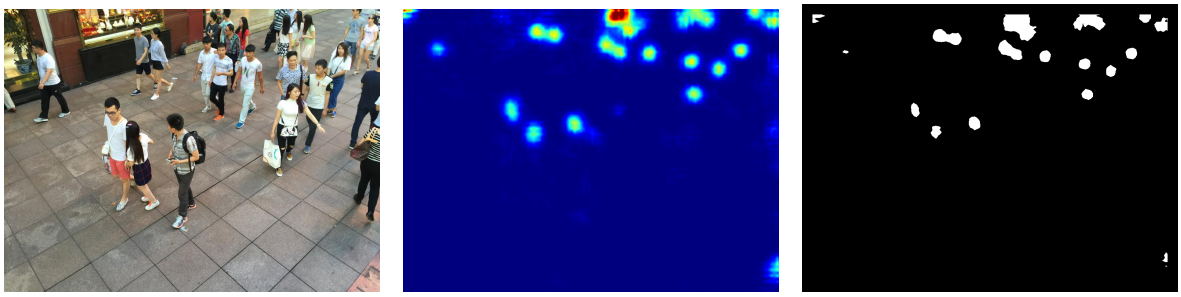
a) b)

Figura 2.5: a) Aliasing, b) Antialiasing.

Con la aplicación del suavizado de contornos es posible determinar las regiones de mayor intensidad en el mapa de densidad, las cuales se determinan por el rango de colores HSV. La imagen de salida se convierte de formato RGB a HSV para binarizar el contenido convirtiendo los píxeles del fondo en 0 y las zonas intensas en 1, como se presenta en la Figura 2.6. La biblioteca para la binarización de imágenes se encuentra en el servidor GitHub de OpenCV [26].

Rango de colores HSV:

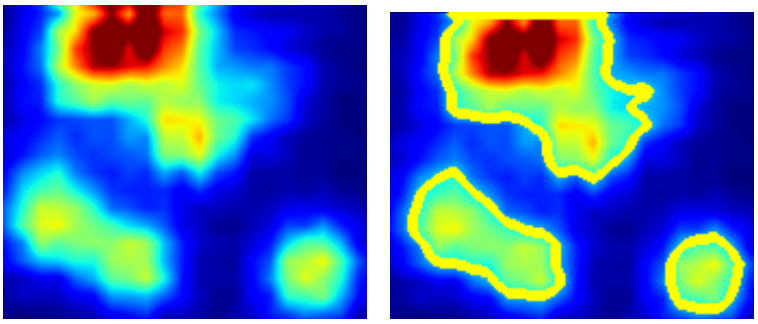
Rango de colores = [0, 50, 20], [30, 255, 255]
Rango de colores = [150, 50, 20], [180, 255, 255]



a) b) c)

Figura 2.6: a) Imagen original [20], b) Mapa de densidad, c) Binarización

Al tener una referencia de las zonas intensas en el mapa de densidad se dibuja los contornos en el mapa de densidad sin binarizar y sumar el total de estas regiones, como se presenta en la Figura 2.7.



a) b)

Figura 2.7: a) Sin contorno, b) Con contorno.

2.5.1. Interfaz gráfica

La interfaz gráfica se diseña con el propósito de visualizar los resultados de salida mediante el conjunto de herramientas de PyQt5. Esta biblioteca se encuentra disponible mediante la instalación de PyQt5 Designer ejecutando en la consola los siguientes comandos:

```
pip3 install PyQt5          % Biblioteca de herramientas para Interfaz Gráfica.
pip3 install PyQt5Designer  % Visualizador de Interfaces.
```

La interfaz principal como se presenta en la Figura 2.8 tiene varias funciones descritas a continuación:

- La primera función permite seleccionar uno de los modelos entrenados.
  - Multitudes dispersas.
  - Multitudes densas.
  - Multitudes combinadas.
- La segunda función permite ingresar una imagen como dato de entrada, estas imágenes puede ser de las siguientes extensiones:
  - JPG.
  - PNG.

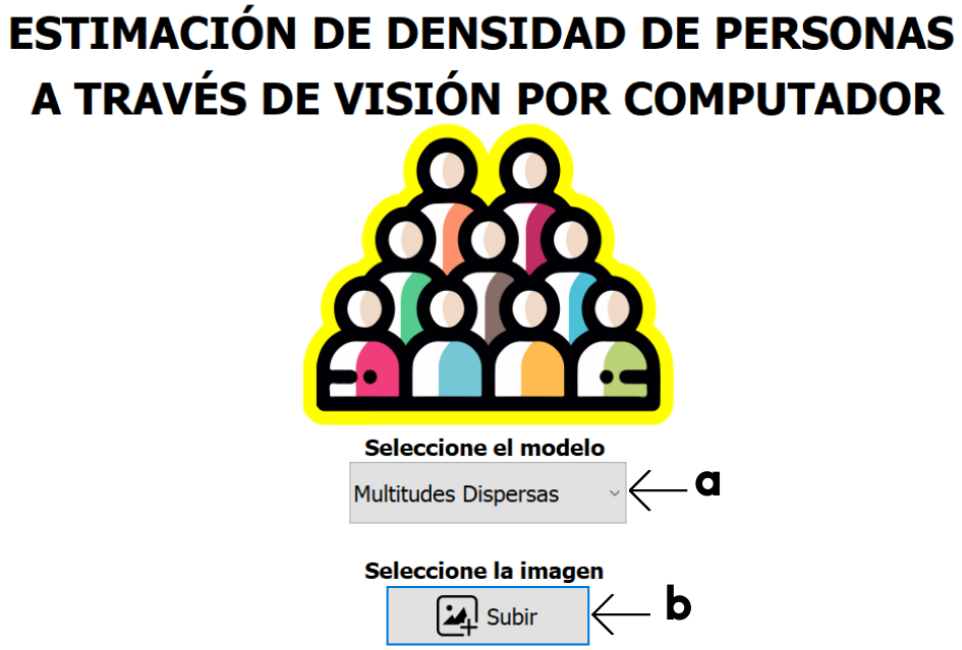


Figura 2.8: Interfaz inicial. a) Selección del modelo entrenado, b) Subir imagen.

La interfaz de resultados como se presenta en la Figura 2.9 tiene varias funciones descritas a continuación:

- La primera función permite visualizar la imagen de entrada.
- La segunda función consiste en el mapa de densidad generado por la red neuronal más el contorno en las regiones de mayor intensidad.
- La última función se observa el conteo estimado de personas en la imagen.

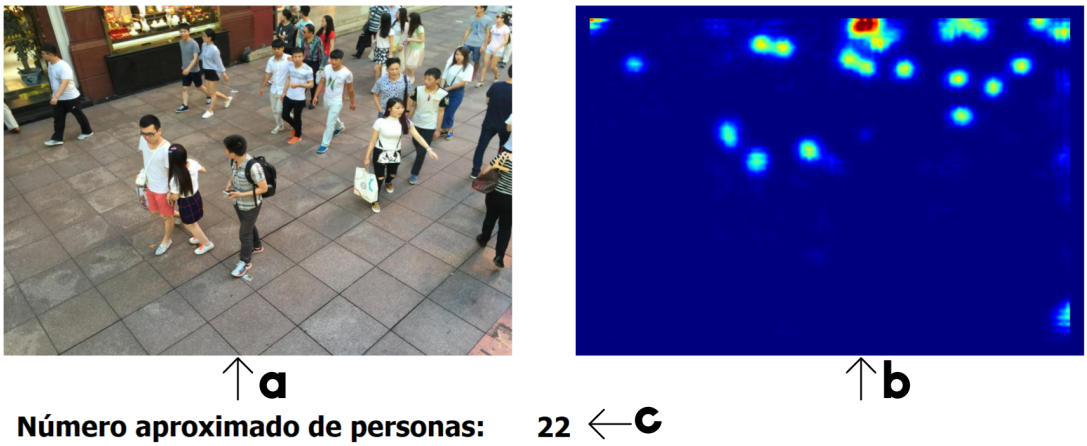


Figura 2.9: Interfaz de visualización de datos. a) Imagen original, b) Mapa de densidad, c) Número de personas.

2.5.2. Base de datos

Para almacenar los datos de salida se crea una base de datos mediante SQLite, como se presenta en la Figura 2.10. En esta se almacena los principales resultados que se visualizan en la segunda interfaz gráfica como un historial de consultas. Además, la información se almacena en carpetas contenedoras con los registros de la base de datos incluyendo, los archivos de imágenes, archivo de texto, un reporte general en formato PDF y XLSX para que cualquier persona o programa autorizado pueda acceder a ella, independientemente del uso que se haga con esta información. La biblioteca se encuentra disponible mediante la instalación de SQLite3 y sus complementos ejecutando en la consola los siguientes comandos:

```
sudo apt install sqlite3                % Biblioteca de base de datos SQLite3.
sudo apt-get install sqlitebrowser      % Visualizador de las características internas SQLite3.
sudo pip3 install reportlab             % Biblioteca para genera archivos PDF.
sudo pip3 install workbook              % Biblioteca para genera archivos XLSX.
sudo pip3 install arrow                 % Biblioteca para obtener la hora actual.
sudo pip3 install pytest-shutil         % Biblioteca para realizar operaciones en carpetas.
```

Archivos Ayuda Opciones

No. Consulta	Nombre de la Imagen	No. Aproximado de personas	Modelo Seleccionado	Hora de Consulta	Fecha de Consulta
1	CONSULTA_0.png	22	Multitudes Dispersas	14:18:22	Junio 13, 2021
2	CONSULTA_1.png	22	Multitudes Dispersas	15:23:39	Junio 13, 2021
3	CONSULTA_2.png	22	Multitudes Dispersas	15:43:39	Junio 13, 2021
4	CONSULTA_3.png	22	Multitudes Dispersas	15:44:05	Junio 13, 2021
5	CONSULTA_4.png	22	Multitudes Dispersas	21:52:03	Junio 13, 2021
6	CONSULTA_5.png	22	Multitudes Dispersas	19:43:34	Junio 15, 2021
7	CONSULTA_6.png	25	Multitudes Densas	19:43:57	Junio 15, 2021
8	CONSULTA_7.png	54	Multitudes Combinadas	19:44:19	Junio 15, 2021
9	CONSULTA_8.png	1819	Multitudes Dispersas	19:54:58	Junio 15, 2021
10	CONSULTA_9.png	810	Multitudes Densas	19:55:35	Junio 15, 2021
11	CONSULTA_10.png	780	Multitudes Combinadas	19:55:48	Junio 15, 2021
12	CONSULTA_11.png	22	Multitudes Dispersas	13:41:17	Julio 06, 2021
13	CONSULTA_12.png	1819	Multitudes Dispersas	13:47:34	Julio 06, 2021
14	CONSULTA_13.png	810	Multitudes Densas	13:47:56	Julio 06, 2021
15	CONSULTA_14.png	22	Multitudes Dispersas	13:48:09	Julio 06, 2021
16	CONSULTA_15.png	1819	Multitudes Dispersas	13:48:47	Julio 06, 2021
17	CONSULTA_16.png	22	Multitudes Dispersas	13:49:19	Julio 06, 2021
18	CONSULTA_17.png	1819	Multitudes Dispersas	13:49:40	Julio 06, 2021
19	CONSULTA_18.png	22	Multitudes Dispersas	13:50:01	Julio 06, 2021
20	CONSULTA_19.png	1819	Multitudes Dispersas	13:50:28	Julio 06, 2021

Figura 2.10: Base de Datos.

El diseño final estuvo enfocado en la sencillez y facilidad de uso para evitar la mayor cantidad de errores de funcionamiento del sistema. En base a lo mencionado anteriormente se hace énfasis en varios criterios de usabilidad que hay que tener en cuenta:

- **Facilidad de aprendizaje. -** La primera experiencia del usuario con el sistema tiene como objetivo el aprendizaje del funcionamiento y la observación de la capacidad de este, el sistema está desarrollado de tal manera que no se necesite instrucciones previas y que tarden lo menos posible en aprender a manejarla de manera eficiente.
- **Facilidad de uso. -** Los nuevos usuarios pueden interactuar con el sistema de forma fácil e intuitiva debido a la interfaz gráfica que permite en pocos pasos conseguir un resultado.
- **Flexibilidad. -** El sistema tiene la capacidad de exportar la información almacenada en la base de datos a otros formatos como lo son PDF y XLSX o directamente desde la base de datos SQLite a motores de búsqueda más avanzados.
- **Robustez. -** La robustez de este sistema depende de la arquitectura neuronal y la unidad de procesamiento gráfico los cuales se relacionan con el tiempo de espera de los resultados por parte del usuario.

Capítulo 3

Resultados

3.1. Métricas de rendimiento

Como se mencionó en la Sección 1.2.5, MAE corresponde al promedio del valor absoluto de la diferencia entre la aproximación real y la predicción dada por la salida del modelo entrenado y MSE hace referencia al promedio de la dispersión de datos, en el conjunto de imágenes.

Se calculó los resultados MAE y MSE de las 182 imágenes del subconjunto de alta densidad, 400 imágenes del subconjunto de media densidad y 582 imágenes del subconjunto creado con la combinación de todas las densidades.

Tabla 3.1: Resultados obtenidos de las métricas de rendimiento.

Métricas de Rendimiento						
Modelos Entrenados CSRNet	Imágenes RGB del conjunto de datos ShanghaiTech					
	Alta densidad		Media densidad		Combinaciones	
	MAE	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**
Resultados Oficiales [9]	68.20	115.00	10.60	16.00	-	-
Alta densidad	**65.96**	104.99	20.39	33.84	**37.04**	68.95
Media densidad	137.21	232.67	**10.65**	16.16	56.90	141.25
Combinación de densidades	75.69	112.37	26.17	34.49	44.27	73.27

En la Tabla 3.1 se presenta los resultados obtenidos de cada modelo entrenado para las diferentes densidades, para multitudes de alta densidad se obtuvo un error absoluto medio de 65.96 personas y para multitudes de media densidad se obtuvo un error absoluto medio de

10.65 personas, cuya diferencia con los resultados oficiales se debe a las imágenes aleatorias en los archivos JSON generados para el entrenamiento, para la combinación de densidades quien prevalece entre los modelos entrenados es el de alta densidad.

3.2. Funcionamiento de la salida

Para observar el comportamiento de la salida en imágenes de multitudes de media y alta densidad se genera el mapa de densidad para cada modelo entrenado y su predicción del número de personas. Esta predicción se compara con las métricas de rendimiento de la Tabla 3.1, además de la diferencia con el conteo real de la imagen proporcionada por el conjunto de datos.

Para la demostración de cada salida se elige dos imágenes aleatorias de media y alta densidad del conjunto de datos de ShanghaiTech. La Figura 3.1 corresponde a una multitud con densidad media y la Figura 3.2 corresponde a una imagen de multitud con densidad alta.

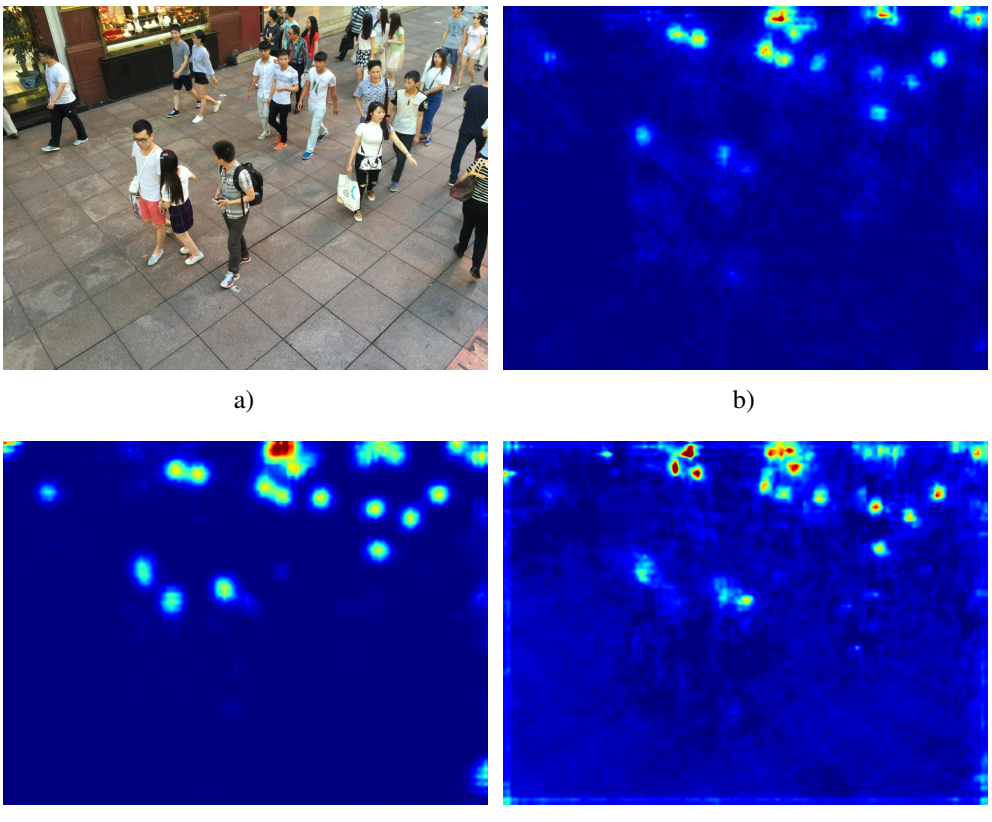
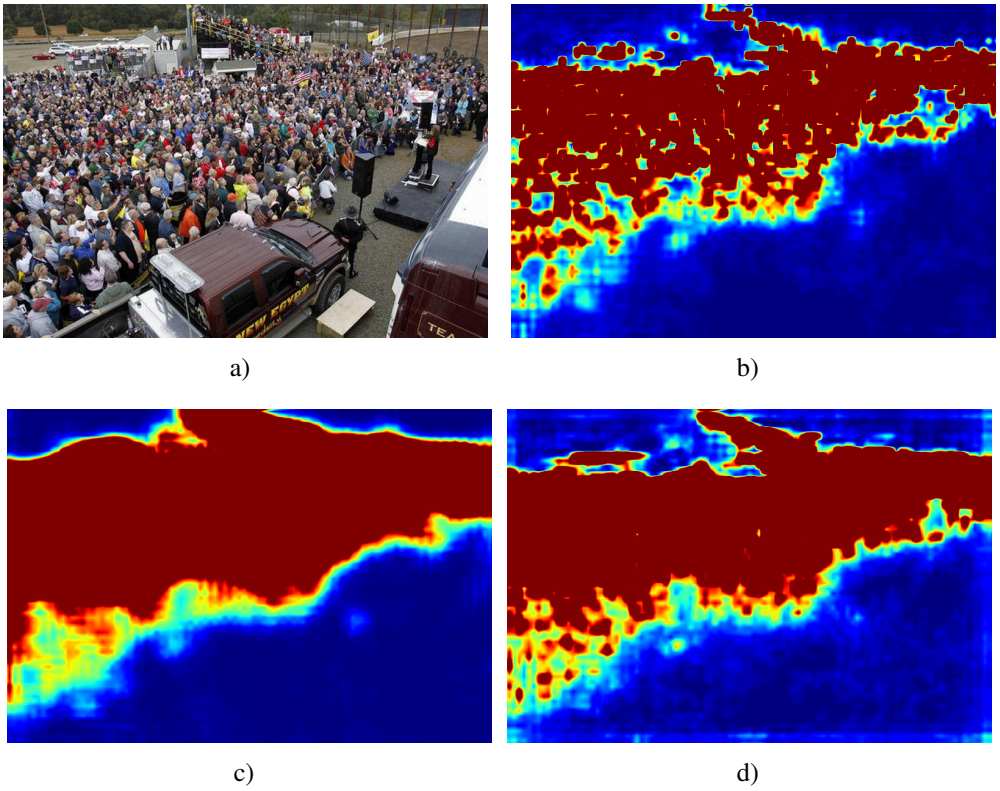


Figura 3.1: Multitud de media densidad. a) Imagen de ejemplo [20],
b) Modelo de alta densidad, c) Modelo de media densidad, d) Modelo combinado.



a) b)

c) d)

Figura 3.2: Multitud de alta densidad. a) Imagen de ejemplo [20], b) Modelo de alta densidad, c) Modelo de media densidad, d) Modelo combinado.

Tabla 3.2: Estimación de personas en imágenes de multitudes de media y alta densidad.

Diferencia de estimación de personas con cada modelo entrenado				
Nombre	Conteo original	Modelo de multitudes		
		Alta densidad	Media densidad	Combinación
Figura 3.1	21	25	22	54
	Diferencia	4	1	33
Figura 3.2	918	810	1819	780
	Diferencia	108	901	138

En la Tabla 3.2 se observa los resultados obtenidos del ejemplo de las Figuras 3.1a, 3.2a, con cada modelo se generó el mapa de densidad (Fig. 3.1b, 3.1c, 3.1d 3.2b, 3.2c, 3.2d), donde se puede identificar en la Figura 3.1c las zonas donde se ubican las cabezas de las personas, pero debido a las aglomeraciones de personas muchas de ellas se mezclan como es el ejemplo de la Figura 3.2b, en el cual el mapa de densidad se satura completamente. También, se calculó la diferencia de personas según la salida en cada caso, siendo el error de conteo acorde a las métricas de rendimiento de la Tabla 3.1.

3.3. Dimensiones y pesos de las imágenes

Las imágenes del conjunto de datos que fueron empleadas para el entrenamiento de cada modelo tienen un peso mínimo y máximo, el peso total de la carpeta donde se encuentran almacenadas se toma como referencia para calcular el peso promedio con el fin de no saturar al dispositivo de almacenamiento.

Tabla 3.3: Peso promedio del conjunto de datos.

Peso promedio					
Subconjunto de imágenes	**Número de imágenes**	**Peso total**	**Promedio**	**Peso mínimo**	**Peso máximo**
Media densidad	316	46536 *kB*	147 *kB*	101 *kB*	187 *kB*
Alta densidad	300	40130 *kB*	134 *kB*	26 *kB*	352 *kB*
Total	616	86666 *kB*	141 *kB*	-	-

En la Tabla 3.3 se observa que del total de imágenes y el peso total se obtiene un promedio de 141 *kB*, aunque el sistema fue entrenado con imágenes de peso mínimo de 26 *kB* y peso máximo de 352 *kB* es recomendable emplear el peso promedio para que el programa final no sature al dispositivo de almacenamiento con los datos acumulados en los registros mencionados en la Sección 2.5.2.

La capacidad de la red neuronal es puesta a prueba a diferentes dimensiones y su efecto en la perdida de información observando el consumo de memoria de la unidad de procesamiento gráfico en dimensiones mínimas y máximas.

Para demostrar el funcionamiento en diferentes relaciones de aspecto, se elige a La Figura 3.1, la cual contiene un peso de 140 *kB* y dimensiones de 1024x768 y la Figura 3.2, la cual contiene un peso de 120 *kB* y dimensiones de 795x547, por lo cual se modifica estas resoluciones con el objetivo de encontrar un formato estandarizado para que la red neuronal convolucional pueda procesar sin ocupar la capacidad total de la unidad de procesamiento gráfico.

Tabla 3.4: Pruebas de dimensiones máximas de imágenes.

Dimensiones máximas				
Nombre	**Relación**	**Dimensiones X:Y**	**Peso**	**Conteo**
Figura 3.1	1:1	1007x1007	154 *kB*	20
	4:3	1200x826	152 *kB*	21
	5:6	900x1080	148 *kB*	19
Figura 3.2	1:1	1007x1007	195.8 *kB*	691
	4:3	1200x826	193.3 *kB*	686
	5:6	900x1080	191.3 *kB*	752

En la Tabla 3.4 se indica las dimensiones máximas en los ejes $(X,\ Y)$, por lo cual, no se debe sobrepasar por el elevado consumo de la unidad de procesamiento gráfico, se puede aumentar la dimensión de un eje, pero en consecuencia se debe reducir la dimensión del otro eje, proporcionalmente.

Tabla 3.5: Pruebas de dimensiones mínimas de imágenes.

Dimensiones mínimas				
Nombre	**Relación**	**Dimensiones X:Y**	**Peso**	**Conteo**
Figura 3.1	1:1	9x9	337 B	0
	4:3	12x9	354 B	0
	3:4	9x12	352 B	0
Figura 3.2	1:1	9x9	327 B	0
	4:3	12x9	341 B	0
	3:4	9x12	336 B	0

En la Tabla 3.5 se indica las dimensiones mínimas pero la arquitectura de la red neuronal convolucional no puede leer la información que contiene la imagen de entrada. Por lo tanto, con los resultados obtenidos en las dimensiones de las imágenes se demuestra que variar la dimensión original es equivalente a la perdida de información para la estimación de personas.

Tabla 3.6: Dimensiones promedio del conjunto de datos.

Dimensiones promedio					
Subconjunto de imágenes	**Número de imágenes**	**Eje X mínimo**	**Eje Y mínimo**	**Eje X máximo**	**Eje Y máximo**
Media densidad	316	1024	768	1024	768
Alta densidad	300	293	200	1024	1024

En la Tabla 3.6 se presenta las dimensiones de los ejes X, Y de las imágenes del conjunto de datos, siendo la dimensión de 1024x768 píxeles quien predomina en los modelos entrenados, Aunque el sistema puede procesar diferentes resoluciones, se recomienda emplear estas dimensiones promedio, el cual es un aspecto de imagen estandarizado que está disponible en dispositivos de fotografía.

Establecida la dimensión estandarizada de 1024x768 se realizó pruebas con imágenes RGB obtenidas con diferentes dispositivos de fotografía ubicados en diferentes ángulos y altura respecto al nivel del suelo, todas estas condiciones se detallan en la Tabla 3.7 y Tabla 3.8.

Tabla 3.7: Condiciones de las imágenes.

Figuras	Dispositivo	Procesador	Ángulo de inclinación	Megapíxeles	Altura
Figura 3.3	Samsung A6+	Snapdragon 450	Ángulo picado (5°)	16	2.10 metros
Figura 3.4	Samsung A6+	Snapdragon 450	Ángulo picado (5°)	16	2.10 metros
Figura 3.5	Samsung A10s	MediaTek Helio P22	Ángulo picado (15°)	13	6.30 metros
Figura 3.6	Samsung A10s	MediaTek Helio P22	Ángulo picado (10°)	13	6.30 metros
Figura 3.7	Nikon Coolpix	EXPEED	Ángulo picado (10°)	8	2.90 metros
Figura 3.8	Nikon Coolpix	EXPEED	Ángulo picado (10°)	8	2.90 metros

En la Tabla 3.7 se menciona al ángulo de inclinación y la altura a nivel de suelo desde donde se realizó la captura de la fotografía, los dispositivos mencionados cuentan con la dimensión estándar de 1024x768, por lo que no se redimensionó la imagen de entrada.

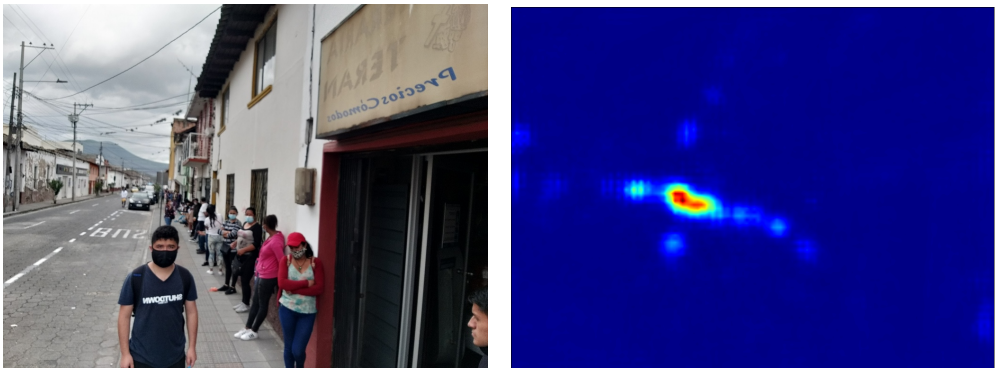
Tabla 3.8: Condición climática.

Figuras	Zona horaria (GMT-5)	Condición climática
Figura 3.3	10:15 a.m.	Nublado
Figura 3.4	10:20 a.m.	Nublado
Figura 3.5	11:30 a.m.	Nubosidad parcial
Figura 3.6	12:00 a.m.	Nubosidad parcial
Figura 3.7	13:10 a.m.	Cielo despejado
Figura 3.8	13:15 a.m.	Cielo despejado

En la Tabla 3.8 se menciona a la condición ambiental del lugar de donde se realizó la captura de la fotografía, para este trabajo de grado se omitió las condiciones de lluvia y fotografiás nocturnas debido a que hasta la actualidad no hay arquitecturas neuronales que puedan detectar personas en esas condiciones.

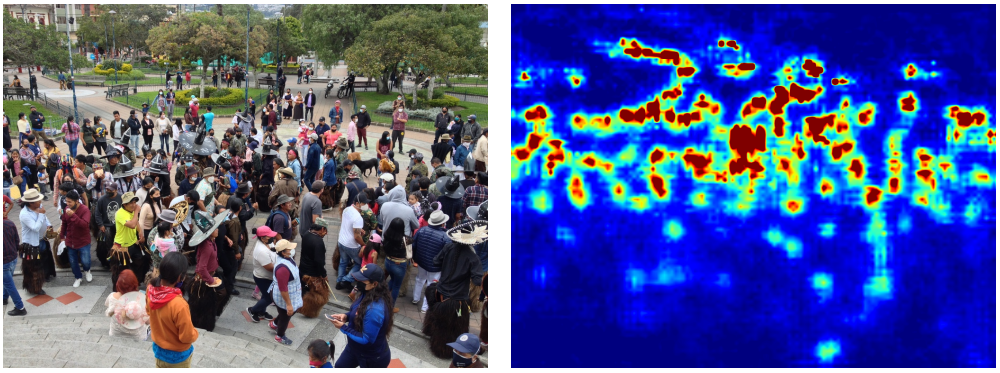
a) b)

Figura 3.3: Prueba 1. a) Imagen de prueba, b) Mapa de densidad.



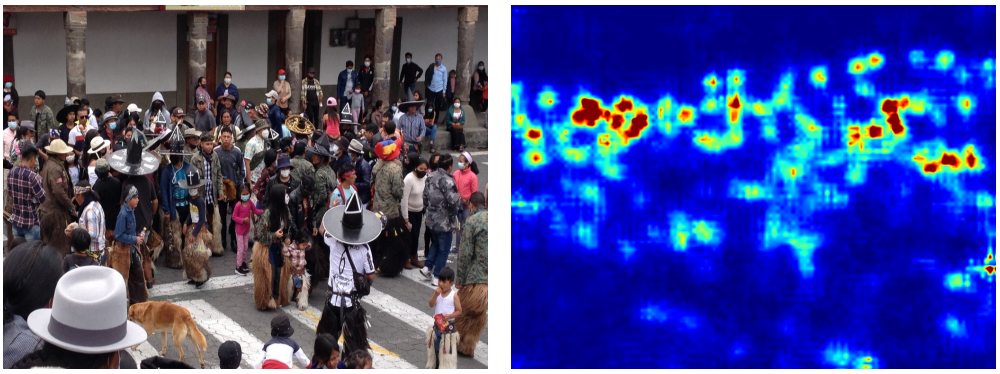
a) b)

Figura 3.4: Prueba 2. a) Imagen de prueba, b) Mapa de densidad.



a) b)

Figura 3.5: Prueba 3. a) Imagen de prueba, b) Mapa de densidad.

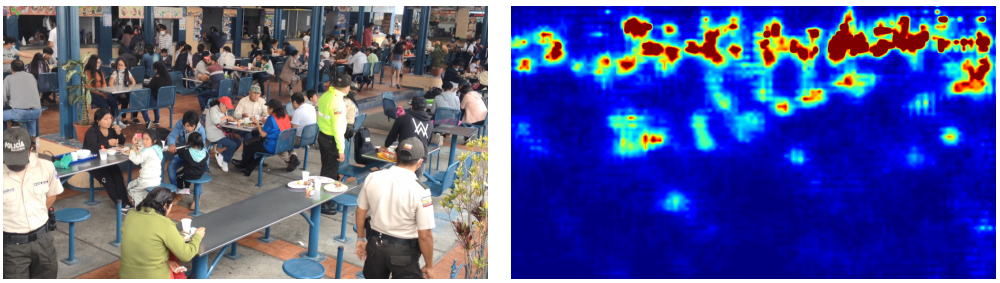


a) b)

Figura 3.6: Prueba 4. a) Imagen de prueba, b) Mapa de densidad.

a) b)

Figura 3.7: Prueba 5. a) Imagen de prueba, b) Mapa de densidad.



a) b)

Figura 3.8: Prueba 6. a) Imagen de prueba, b) Mapa de densidad.

Tabla 3.9: Resultados de las pruebas.

Figuras	Predicción del modelo	Conteo real	Diferencia de conteo	Modelo
Figura 3.3	15	22	7	Multitudes dispersas
Figura 3.4	16	20	4	Multitudes dispersas
Figura 3.5	178	160	18	Multitudes densas
Figura 3.6	112	105	7	Multitudes densas
Figura 3.7	112	120	30	Multitudes densas
Figura 3.8	113	120	30	Multitudes densas

En la Tabla 3.9 se observa los diferentes resultados de las imágenes anteriormente expuestas, donde se puede visualizar que las zonas más alejadas de la cámara al no poder detectar a las personas generan en el mapa de densidad más saturación y también al estar las personas sin distanciamiento físico se puede interpretar a esta saturación como una posible zona aglomerada.

Capítulo 4

Conclusiones y Trabajo Futuro

Este capítulo describe las conclusiones del presente proyecto y manifiesta las posibles líneas para un trabajo futuro.

4.1. Conclusiones

Se identificó la arquitectura CSRNet como la red neuronal convolucional para la estimación de densidad de multitudes de personas y la generación de mapas de densidad con salida de 1/8 respecto a las dimensiones de la imagen de entrada.

Con base en el análisis realizado, uno de los pilares fundamentales de este trabajo de grado fue definir la funcionalidad de la arquitectura CSRNet, de esta forma se establece la capacidad y requerimientos funcionales del sistema, a partir de las necesidades básicas descritas en el alcance de este trabajo de grado.

El sistema se desarrolló en software libre empleando PyTorch para la implementación de la arquitectura CSRNet, en imágenes de multitudes densas y dispersas. Por el gran número de herramientas que posee PyTorch, este sistema puede ser adaptable para tecnología de código abierto.

El sistema de estimación de densidad de personas consta de tres subfunciones, la primera subfunción corresponde al procesamiento de las imágenes, la segunda subfunción tiene como propósito visualizar el mapa de densidad y el conteo de personas, y la tercera subfunción corresponde a guardar en una base de datos la información obtenida en los datos de salida.

Finalmente, mediante las pruebas realizadas al sistema, se garantiza el funcionamiento de este proyecto empleando la dimensión estándar de 1024 x 768 e imágenes a color con posición de la cámara en ángulo en picado o ángulo en cenital. Además, el peso de la imagen debe estar aproximado a 138 kB, de esta forma da validez al sistema propuesto en este trabajo de grado.

4.2. Trabajo futuro

En el caso de buscar una solución para la estimación de densidad de personas en vídeos, se debería mejorar la capacidad de la unidad de procesamiento gráfico, para que logre procesar imágenes en menor tiempo sin saturación de memoria, además, implementar arquitecturas más robustas que mejoren las métricas de rendimiento para la aproximación de personas.

Si se requiere implementar este sistema en un lugar predeterminado, se debe crear un nuevo conjunto de datos, utilizando imágenes de una cámara instalada y fija, este nuevo conjunto de datos debe contener el conteo real e imágenes con diversas iluminaciones en diferentes instantes de tiempo.

La arquitectura implementada en el presente trabajo permite contar vehículos y generar un mapa de densidad en situaciones de congestión vehicular, por lo cual, se debe realizar un entrenamiento del conjunto de datos *TRANCOS*, la cual puede ser implementada en cámaras de videovigilancia ya existentes.

Capítulo 5

Listado de códigos

Este apéndice incluye el sistema desarrollado en el proyecto. Sólo se han anexado el archivo más importante. Todo el código utilizado en este trabajo se encuentra en el servidor GitHub en **`https://github.com/DanielZambranoA/Trabajo_De_Grado`**.

Programa 5.1: Diseño del sistema de estimación de densidad de multitudes de personas

```
# -*- coding: utf-8 -*-
# !/usr/bin/env python3

# ----------------------------------------------------------------------------
# Estimaci\'on de Densidad de Personas a trav\'es de Visi\'on por Computador
# Archivo:      main.py
# Autor:        Daniel David Zambrano Andrade
# Creado:       25 de Septiembre 2020
# Modificado:   04 de Junio 2021
# Copyright:    Libre
# License:      Libre
# ----------------------------------------------------------------------------

import cv2
import sys
from time import strftime
from IPython.external.qt_for_kernel import QtCore
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog, QMessageBox,
    QDesktopWidget, QTableWidget, QToolBar, \
    QStatusBar, QAction, QTableWidgetItem, QDialog, QVBoxLayout, QLineEdit, QPushButton, \
    QHeaderView, QAbstractItemView, QDialogButtonBox, QLabel, QGridLayout
from PyQt5.QtCore import Qt
from PyQt5.uic import *
from PyQt5.QtGui import QPixmap, QIcon, QTextDocument, QTextCursor, QIntValidator
import PIL.Image as Image
import numpy as np
from PyQt5.QtCore import *
from qtpy.uic import loadUi
from skimage.transform import rescale
from model import CSRNet
import torch
from torchvision import transforms
from matplotlib import pyplot as plt
import sqlite3
import os
```

```
from arrow import utcnow
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch, mm
from reportlab.lib.pagesizes import letter
from reportlab.platypus import Paragraph, SimpleDocTemplate, Spacer, Table, TableStyle
from reportlab.lib.enums import TA_LEFT, TA_CENTER
from reportlab.lib.colors import black, purple, white
from reportlab.pdfgen import canvas
from xlsxwriter.workbook import Workbook
import locale
from shutil import rmtree
global fecha
global use_cuda
locale.setlocale(locale.LC_ALL, '')


class VentanaPrincipal(QMainWindow):
    def __init__(self):
        try:
            global use_cuda
            use_cuda = torch.cuda.is_available()
            if use_cuda:
                super(VentanaPrincipal, self).__init__()
                loadUi('Interfaz.ui', self)
                self.BotonImagenes.clicked.connect(self.abrirVentanaImagenesConsulta)
                self.actionHistorial_de_Consultas.triggered.connect(self.historial)
                self.center()
                self.setFixedSize(1020, 720)
                self.multitud = ""
                help_menu = self.menuBar().addMenu("&Ayuda")
                about_ayuda = QAction(QIcon("Recursos/ayuda.png"), "Ayuda", self)
                about_ayuda.triggered.connect(self.ayuda)
                help_menu.addAction(about_ayuda)
                ver_gpu = QAction(QIcon("Recursos/NVIDIA.png"), "Verificaci\'on CUDA", self)
                ver_gpu.triggered.connect(self.CUDA)
                help_menu.addAction(ver_gpu)
                about_action = QAction(QIcon("Recursos/Yo.png"), "Cr\'editos", self)
                about_action.triggered.connect(self.about)
                help_menu.addAction(about_action)
            else:
                self.CUDA()
        except (ValueError, Exception):
            pass

    def abrirVentanaImagenesConsulta(self):
        self.hide()
        self.multitud = str(self.comboBox.currentText())
        multitud = str(self.multitud)
        VentanaImagenesConsulta(self, multitud)

    def closeEvent(self, event):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Salir")
        messageBox.setIcon(QMessageBox.Question)
        messageBox.setWindowIcon(QIcon("Recursos/exit.png"))
        messageBox.setText("?`Desea salir del programa?")
        Si = messageBox.addButton("Si", QMessageBox.YesRole)
        No = messageBox.addButton("No", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()
        if messageBox.clickedButton() == Si:
            event.accept()
        elif messageBox.clickedButton() == No:
            event.ignore()
```

```
    def center(self):
        qr = self.frameGeometry()
        cp = QDesktopWidget().availableGeometry().center()
        qr.moveCenter(cp)
        self.move(qr.topLeft())

    def historial(self):
        window = BaseDeDatos(self)
        window.show()
        window.loaddata()

    @staticmethod
    def about():
        dlg = AboutDialog()
        dlg.exec_()

    @staticmethod
    def ayuda():
        help_1 = Ayuda()
        help_1.exec_()

    @staticmethod
    def CUDA():
        ver_cuda = VerificacionGPU()
        ver_cuda.exec_()


class VentanaImagenesConsulta(QMainWindow):
    def __init__(self, parent=None, multitud=None):
        super(VentanaImagenesConsulta, self).__init__(parent)
        self.ventana = loadUi("ImagenesConsulta.ui", self)
        self.setWindowFlags(QtCore.Qt.Window | QtCore.Qt.CustomizeWindowHint | QtCore.Qt.
            WindowTitleHint | QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowStaysOnTopHint
            )
        self.multitudes = multitud
        self.get_image_file()
        self.base = ""
        self.name = ""
        self.nameFile = ""
        self.file = ""
        self.dir_F = ""
        self.rutaAlmacenamiento = ""
        self.conn = ""
        self.c = ""
        self.den = ""
        self.Rden = ""
        self.conteo = 0
        self.center()
        self.setFixedSize(1800, 900)
        self.totalCnts = ""
        self.Zona = ""
        self.imagen_O = ""
        self.Img_O = ""
        self.Zona_N = ""
        self.Zona_B = ""
        self.Zona_C = ""
        self.imagen = ""
        self.imagen_2 = ""
        self.imagen_3 = ""
        self.IMG_CV2JET = ""
        self.Number_2.setStyleSheet("QLabel""{""color : #ffffff;""background : red;""}")

    def get_image_file(self):
```

```
        try:
            file_name, _ = QFileDialog.getOpenFileName(self, 'Open Image File', r"", "Image
                files (*.jpg *.jpeg *.png)")
            self.name = os.path.abspath(file_name)
            self.ImagenOriginal.setPixmap(QPixmap(file_name))
            self.predecir()
            self.ventana.show()
        except (ValueError, Exception):
            self.abrirVentanaPrincipal()

    def predecir(self):
        try:
            global fecha
            fecha = strftime("%B-%d-%Y_%H_%M_%S")
            self.base = os.path.basename(self.name)
            self.file = os.path.splitext(self.base)[0]
            try:
                os.stat("Modelos Entrenados")
            except (ValueError, Exception):
                os.mkdir("Modelos Entrenados")
            if self.multitudes == "Multitudes Dispersas":
                modeloentrenado = 'Modelos Entrenados/Multitudes_Dispersas.pth.tar'
            elif self.multitudes == "Multitudes Densas":
                modeloentrenado = "Modelos Entrenados/Multitudes_Densas.pth.tar"
            elif self.multitudes == "Multitudes Combinadas":
                modeloentrenado = 'Modelos Entrenados/Multitudes_Densas-Dispersas.pth.tar'
            else:
                modeloentrenado = 'Modelos Entrenados/Multitudes_Dispersas.pth.tar'
            transform = transforms.Compose(
                [transforms.ToTensor(), transforms.Normalize(mean=[0.485, 0.456, 0.406], std
                    =[0.229, 0.224, 0.225]), ])
            model = CSRNet()
            model = model.cuda()
            checkpoint = torch.load(modeloentrenado)
            model.load_state_dict(checkpoint['state_dict'])
            img = transform(Image.open(self.name).convert('RGB')).cuda()
            output = model(img.unsqueeze(0))
            self.conteo = str(int(output.detach().cpu().sum().numpy()))
            MapaDensidad = np.asarray(
                output.detach().cpu().reshape(output.detach().cpu().shape[2], output.detach().
                    cpu().shape[3]))
            ROOT_DIR = os.path.dirname(__file__)
            try:
                os.stat(ROOT_DIR + "/Mapas_De_Densidad/")
            except (ValueError, Exception):
                os.mkdir(ROOT_DIR + "/Mapas_De_Densidad/")
            self.dir_F = ROOT_DIR + "/Mapas_De_Densidad/" + str(str(self.file)) + "_" + str(
                fecha).capitalize()
            self.Rden = self.dir_F + "/" + str(str(self.file))
            self.den = self.Rden + "_Densidad" + ".png"
            os.mkdir(self.dir_F)
            MapaDensidad = rescale(MapaDensidad, 8, anti_aliasing=True)
            plt.imsave(self.den, MapaDensidad, cmap='jet', vmin=0, vmax=0.07)
            self.Number.setText(self.conteo)
            rojoBajo1 = np.array([0, 50, 20], np.uint8)
            rojoAlto1 = np.array([30, 255, 255], np.uint8)
            rojoBajo2 = np.array([150, 50, 20], np.uint8)
            rojoAlto2 = np.array([180, 255, 255], np.uint8)
            self.imagen = cv2.imread(self.den)
            self.imagen_2 = cv2.imread(self.den)
            self.imagen_O = cv2.imread(self.name)
            bordersize = 30
            self.imagen = cv2.copyMakeBorder(self.imagen, top=bordersize, bottom=bordersize,
                left=bordersize,
```

```
                                                right=bordersize, borderType=cv2.BORDER_CONSTANT,
                                                    value=[127, 0, 0])
            self.imagen_2 = cv2.copyMakeBorder(self.imagen_2, top=bordersize, bottom=
                bordersize, left=bordersize,
                                                    right=bordersize, borderType=cv2.
                                                        BORDER_CONSTANT, value=[127, 0, 0])
            imagenHSV = cv2.cvtColor(self.imagen, cv2.COLOR_BGR2HSV)
            maskRojo1 = cv2.inRange(imagenHSV, rojoBajo1, rojoAlto1)
            maskRojo2 = cv2.inRange(imagenHSV, rojoBajo2, rojoAlto2)
            maskRojo = cv2.add(maskRojo1, maskRojo2)
            contornosRojo = cv2.findContours(maskRojo, cv2.RETR_EXTERNAL, cv2.
                CHAIN_APPROX_SIMPLE)[0]
            self.dibujarContorno(contornosRojo)
            self.totalCnts = len(contornosRojo)
            self.imagen_3 = cv2.applyColorMap(self.imagen_2, cv2.COLORMAP_JET)
            self.IMG_CV2JET = self.Rden + "_Mapa_Calor" + ".png"
            self.Img_O = self.Rden + ".png"
            self.Zona_N = self.Rden + "_Numerada" + ".png"
            self.Zona_C = self.Rden + "_Contorno" + ".png"
            self.Zona_B = self.Rden + "_Binaria" + ".png"
            cv2.imwrite(self.Zona_B, maskRojo)
            cv2.imwrite(self.Zona_N, self.imagen)
            cv2.imwrite(self.Zona_C, self.imagen_2)
            cv2.imwrite(self.Img_O, self.imagen_O)
            cv2.imwrite(self.IMG_CV2JET, self.imagen_3)
            self.MapaDeDensidad_2.setPixmap(QPixmap(self.Zona_C))
            self.Number_2.setText(str(self.totalCnts))
            self.AlmacenarDatos()
            pdf()
            sqliteExcel()
        except (ValueError, Exception):
            self.errorModelo()
            self.ventana.show().quit()

    def errorModelo(self):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Error")
        messageBox.setIcon(QMessageBox.Critical)
        messageBox.setWindowIcon(QIcon("Recursos/Error.png"))
        messageBox.setText(
            "Falla en el sistema, compruebe las dimensiones de la imagen y el modelo entrenado
                 de la red neuronal convolucional.")
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()

    def dibujarContorno(self, contornos):
        for (i, c) in enumerate(contornos):
            M = cv2.moments(c)
            if M["m00"] != 0:
                x = int(M["m10"] / M["m00"])
                y = int(M["m01"] / M["m00"])
            else:
                x, y = 0, 0
            cv2.drawContours(self.imagen, [c], 0, (0, 0, 255), 2)
            cv2.drawContours(self.imagen_2, [c], 0, (0, 255, 255), 4)
            cv2.putText(self.imagen, str(i + 1), (x, y), 5, 1, (0, 0, 0), 2)

    def AlmacenarDatos(self):
        ImagenOriginal = os.path.basename(self.Img_O)
        Aproximacion = self.conteo
        Zona = self.totalCnts
        Modelo = self.multitudes
        Fecha = strftime("%H:%M:%S")
```

```
        Year = strftime("%B %d, %Y").capitalize()
        MapaDeDensidad = os.path.basename(self.den)
        ImagenBinaria = os.path.basename(self.Zona_B)
        ImagenNumerada = os.path.basename(self.Zona_N)
        ImagenContorno = os.path.basename(self.Zona_C)
        MapaCV2 = os.path.basename(self.IMG_CV2JET)
        Ruta = self.dir_F
        try:
            try:
                os.stat("Database")
            except (ValueError, Exception):
                os.mkdir("Database")
            self.conn = sqlite3.connect("Database/Consultas.db")
            self.c = self.conn.cursor()
            self.c.execute(
                "CREATE TABLE IF NOT EXISTS historial(roll INTEGER PRIMARY KEY AUTOINCREMENT
                    NOT NULL ,ImagenOriginal TEXT,"
                "Aproximacion TEXT,Zona TEXT, Modelo TEXT, Fecha TEXT, Year Text,
                    MapaDeDensidad TEXT, ImagenBinaria TEXT,ImagenNumerada TEXT,
                    ImagenContorno TEXT, MapaCV2 TEXT, Ruta TEXT)")
            self.c.execute(
                "INSERT INTO historial (ImagenOriginal, Aproximacion, Zona, Modelo, Fecha,
                    Year,  MapaDeDensidad, ImagenBinaria,ImagenNumerada, ImagenContorno,
                    MapaCV2, Ruta) VALUES (?,?,?,?,?,?,?,?,?,?,?,?)",
                (ImagenOriginal, Aproximacion, Zona, Modelo, Fecha, Year, MapaDeDensidad,
                    ImagenBinaria, ImagenNumerada,
                 ImagenContorno, MapaCV2, Ruta))
            self.conn.commit()
            self.c.close()
            self.conn.close()
            file = open(Ruta + "/Resumen.txt", "w")
            file.write("Nombre de la imagen: " + ImagenOriginal + os.linesep)
            file.write("Aproximaci\'on de personas: " + str(Aproximacion) + os.linesep)
            file.write("Zonas de posibles aglomeraciones: " + str(Zona) + os.linesep)
            file.write("Modelo seleccionado: " + Modelo + os.linesep)
            file.write("Hora de consulta: " + Fecha + os.linesep)
            file.write("Fecha de consulta: " + Year + os.linesep)
            file.write("Nombre del mapa de densidad: " + MapaDeDensidad + os.linesep)
            file.write("Nombre de la imagen binaria: " + ImagenBinaria + os.linesep)
            file.write("Nombre de la imagen numerada: " + ImagenNumerada + os.linesep)
            file.write("Nombre de la imagen con contorno: " + ImagenContorno + os.linesep)
            file.write("Nombre del mapa de densidad de openCV: " + MapaCV2 + os.linesep)
            file.write("Ruta de la carpeta contenedora: " + Ruta + os.linesep)
            file.close()
            self.confirmacionConsulta()
        except (ValueError, Exception):
            self.errorConsultas()

    def confirmacionConsulta(self):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Mensaje")
        messageBox.setIcon(QMessageBox.Information)
        messageBox.setWindowIcon(QIcon("Recursos/database.png"))
        messageBox.setText("Consulta agregada a historial de consultas")
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()

    def errorConsultas(self):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Error")
        messageBox.setIcon(QMessageBox.Warning)
        messageBox.setWindowIcon(QIcon("Recursos/database.png"))
        messageBox.setText("Consulta no agregada a historial. Reinicie el programa.")
```

```
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()

    def abrirVentanaPrincipal(self):
        self.parent().show()
        self.close()

    def closeEvent(self, event):
        self.abrirVentanaPrincipal()

    def center(self):
        qr = self.frameGeometry()
        cp = QDesktopWidget().availableGeometry().center()
        qr.moveCenter(cp)
        self.move(qr.topLeft())


class BaseDeDatos(QMainWindow):
    def __init__(self, *args, **kwargs):
        super(BaseDeDatos, self).__init__(*args, **kwargs)
        self.setWindowIcon(QIcon("Recursos/database.png"))
        self.connection = ""
        self.conn = sqlite3.connect("Database/Consultas.db")
        self.c = self.conn.cursor()
        self.c.execute(
            "CREATE TABLE IF NOT EXISTS historial(roll INTEGER PRIMARY KEY AUTOINCREMENT NOT
                NULL ,ImagenOriginal TEXT,"
            "Aproximacion TEXT,Zona TEXT, Modelo TEXT,Fecha TEXT, Year Text, MapaDeDensidad
                TEXT, ImagenBinaria TEXT,ImagenNumerada TEXT, ImagenContorno TEXT, MapaCV2
                TEXT, Ruta)")
        self.c.close()
        self.center()
        file_menu = self.menuBar().addMenu("&Archivos")
        self.setWindowTitle("Historial de Consultas")
        self.setMinimumSize(1345, 740)
        self.setMaximumSize(1350, 740)
        self.tableWidget = QTableWidget()
        self.setCentralWidget(self.tableWidget)
        self.tableWidget.setAlternatingRowColors(True)
        self.tableWidget.setColumnCount(12)
        self.tableWidget.horizontalHeader().setCascadingSectionResizes(False)
        self.tableWidget.horizontalHeader().setSortIndicatorShown(False)
        self.tableWidget.horizontalHeader().setStretchLastSection(False)
        self.tableWidget.verticalHeader().setVisible(False)
        self.tableWidget.verticalHeader().setCascadingSectionResizes(False)
        self.tableWidget.verticalHeader().setStretchLastSection(False)
        self.tableWidget.setHorizontalHeaderLabels(("No. Consulta", "Nombre de la Imagen", "No
            . Aproximado de personas",
                                                    "Zonas Posibles de Aglomeraci\'on", "
                                                        Modelo Seleccionado",
                                                    "Hora de Consulta", "Fecha de Consulta", "
                                                        Mapa de Densidad",
                                                    "Imagen Binaria",
                                                    "Imagen Numerada", "Imagen Contorno", "
                                                        Mapa CV2", "Ruta"))
        self.tableWidget.verticalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
        for indice, ancho in enumerate((100, 300, 225, 225, 175, 150, 150, 300, 300, 300, 300,
             300), start=0):
            self.tableWidget.setColumnWidth(indice, ancho)
        self.tableWidget.setColumnHidden(7, True)
        self.tableWidget.setColumnHidden(8, True)
        self.tableWidget.setColumnHidden(9, True)
```

```
        self.tableWidget.setColumnHidden(10, True)
        self.tableWidget.setColumnHidden(11, True)
        self.tableWidget.setColumnHidden(12, True)
        toolbar = QToolBar()
        toolbar.setMovable(False)
        self.addToolBar(toolbar)
        statusbar = QStatusBar()
        self.setStatusBar(statusbar)
        btn_ac_refresh = QAction(QIcon("Recursos/r3.png"), "Actualizar Reporte/Base de Datos",
             self)  # refresh icon
        btn_ac_refresh.triggered.connect(self.loaddata)
        btn_ac_refresh.setStatusTip("Refresh Table")
        toolbar.addAction(btn_ac_refresh)
        btn_ac_search = QAction(QIcon("Recursos/s1.png"), "Visualizar", self)  # search icon
        btn_ac_search.triggered.connect(self.search)
        btn_ac_search.setStatusTip("Search User")
        toolbar.addAction(btn_ac_search)
        btn_ac_delete = QAction(QIcon("Recursos/d1.png"), "Borrar", self)
        btn_ac_delete.triggered.connect(self.delete)
        btn_ac_delete.triggered.connect(self.loaddata)
        btn_ac_delete.setStatusTip("Borrar Consulta")
        toolbar.addAction(btn_ac_delete)
        searchuser_action = QAction(QIcon("Recursos/s1.png"), "Visualizar", self)
        searchuser_action.triggered.connect(self.search)
        file_menu.addAction(searchuser_action)
        deluser_action = QAction(QIcon("Recursos/d1.png"), "Borrar", self)
        deluser_action.triggered.connect(self.delete)
        deluser_action.triggered.connect(self.loaddata)
        file_menu.addAction(deluser_action)
        help_menu = self.menuBar().addMenu("&Ayuda")
        about_ayuda = QAction(QIcon("Recursos/ayuda.png"), "Ayuda ", self)
        about_ayuda.triggered.connect(self.ayuda)
        help_menu.addAction(about_ayuda)
        ver_gpu = QAction(QIcon("Recursos/NVIDIA.png"), "Verificaci\'on CUDA", self)
        ver_gpu.triggered.connect(self.CUDA)
        help_menu.addAction(ver_gpu)
        about_action = QAction(QIcon("Recursos/Yo.png"), "Cr\'editos ", self)  # info icon
        about_action.triggered.connect(self.about)
        help_menu.addAction(about_action)

    def loaddata(self):
        self.connection = sqlite3.connect("Database/Consultas.db")
        query = "SELECT * FROM historial"
        pdf()
        sqliteExcel()
        result = self.connection.execute(query)
        self.tableWidget.setRowCount(0)
        for row_number, row_data in enumerate(result):
            self.tableWidget.insertRow(row_number)
            for column_number, data in enumerate(row_data):
                item = QTableWidgetItem(str(data))
                item.setTextAlignment(Qt.AlignHCenter)
                self.tableWidget.setItem(row_number, column_number, item)
        self.connection.close()

    def handlePaintRequest(self, printer):
        document = QTextDocument()
        cursor = QTextCursor(document)
        model = self.table.model()
        table = cursor.insertTable(model.rowCount(), model.columnCount())
        for row in range(table.rows()):
            for column in range(table.columns()):
                cursor.insertText(model.item(row, column).text())
                cursor.movePosition(QTextCursor.NextCell)
```

```
            document.print_(printer)

        @staticmethod
        def delete():
            dlg = DeleteDialog()
            dlg.exec_()

        @staticmethod
        def search():
            dlg = SearchDialog()
            dlg.exec_()

        def center(self):
            qr = self.frameGeometry()
            cp = QDesktopWidget().availableGeometry().center()
            qr.moveCenter(cp)

        @staticmethod
        def about():
            dlg = AboutDialog()
            dlg.exec_()

        @staticmethod
        def ayuda():
            help_1 = Ayuda()
            help_1.exec_()

        @staticmethod
        def CUDA():
            ver_cuda = VerificacionGPU()
            ver_cuda.exec_()


    class SearchDialog(QDialog):
        def __init__(self, *args, **kwargs):
            super(SearchDialog, self).__init__(*args, **kwargs)
            self.conn = ""
            self.c = ""
            self.serachresult = ""
            self.QBtn = QPushButton()
            self.QBtn.setText("Visualizar")
            self.setWindowTitle("Visualizar Consulta")
            self.setWindowIcon(QIcon("Recursos/s1.png"))
            self.setFixedWidth(300)
            self.setFixedHeight(100)
            self.QBtn.clicked.connect(self.buscarConsulta)
            layout = QVBoxLayout()
            self.setWindowFlags(QtCore.Qt.Window | QtCore.Qt.CustomizeWindowHint | QtCore.Qt.
                WindowTitleHint | QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowStaysOnTopHint
                )
            self.searchinput = QLineEdit()
            self.onlyInt = QIntValidator()
            self.searchinput.setValidator(self.onlyInt)
            self.searchinput.setPlaceholderText("No. Consulta")
            layout.addWidget(self.searchinput)
            layout.addWidget(self.QBtn)
            self.setLayout(layout)

        def buscarConsulta(self):
            searchrol = self.searchinput.text()
            self.close()
            try:
                self.conn = sqlite3.connect("Database/Consultas.db")
                self.c = self.conn.cursor()
```

```
            result = self.c.execute("SELECT * from historial WHERE roll =" + str(searchrol))
            row = result.fetchone()
            self.serachresult = "Consulta: " + str(row[0]) + '\n' + "Nombre: " + str(
                row[1]) + '\n' + "Aproximaci\'on: " + str(row[2]) + '\n' + "Zonas con
                    Aglomeraci\'on: " + str(
                row[3]) + '\n' + "Modelo: " + str(row[4]) + '\n' + "Hora de Consulta: " + str(
                row[5]) + '\n' + "Fecha de Consulta: " + str(row[6])
            img_o = str(row[12]) + "/" + str(row[1])
            img_d = str(row[12]) + "/" + str(row[7])
            img_b = str(row[12]) + "/" + str(row[8])
            img_n = str(row[12]) + "/" + str(row[9])
            img_c = str(row[12]) + "/" + str(row[10])
            img_mp = str(row[12]) + "/" + str(row[11])
            imagenMP = cv2.imread(img_mp)
            cv2.imshow('Imagen Mapa CV2', imagenMP)
            imagenN = cv2.imread(img_n)
            cv2.imshow('Imagen Numerada', imagenN)
            imagenC = cv2.imread(img_c)
            cv2.imshow('Imagen Contorno', imagenC)
            imagenB = cv2.imread(img_b)
            cv2.imshow('Imagen Binaria', imagenB)
            imagenMD = cv2.imread(img_d)
            cv2.imshow('Mapa de Densidad', imagenMD)
            imagenO = cv2.imread(img_o)
            cv2.imshow('Imagen Original', imagenO)
            self.visualizar()
            self.conn.commit()
            self.c.close()
            self.conn.close()
        except (ValueError, Exception):
            self.errorVisualizacion()

    def visualizar(self):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Visualizaci\'on de Im\'agenes")
        messageBox.setIcon(QMessageBox.Information)
        messageBox.setWindowIcon(QIcon("Recursos/database.png"))
        messageBox.setText(self.serachresult)
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()

    @staticmethod
    def errorVisualizacion():
        messageBox = QMessageBox()
        messageBox.setWindowTitle("Error")
        messageBox.setIcon(QMessageBox.Warning)
        messageBox.setWindowIcon(QIcon("Recursos/database.png"))
        messageBox.setText("Consulta no encontrada.")
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()


class DeleteDialog(QDialog):
    def __init__(self, *args, **kwargs):
        super(DeleteDialog, self).__init__(*args, **kwargs)
        self.conn = ""
        self.c = ""
        self.serachresultdel = ""
        self.QBtn = QPushButton()
        self.QBtn.setText('Borrar')
        self.setWindowTitle("Borrar Consulta")
        self.setWindowIcon(QIcon("Recursos/d1.png"))
```

```
        self.setFixedWidth(300)
        self.setFixedHeight(100)
        self.QBtn.clicked.connect(self.deleteconsulta)
        layout = QVBoxLayout()
        self.setWindowFlags(
            QtCore.Qt.Window | QtCore.Qt.CustomizeWindowHint | QtCore.Qt.WindowTitleHint |
                QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowStaysOnTopHint)
        self.deleteinput = QLineEdit()
        self.onlyInt = QIntValidator()
        self.deleteinput.setValidator(self.onlyInt)
        self.deleteinput.setPlaceholderText("No. Consulta")
        layout.addWidget(self.deleteinput)
        layout.addWidget(self.QBtn)
        self.setLayout(layout)

    def deleteconsulta(self):
        delrol = self.deleteinput.text()
        self.close()
        try:
            self.conn = sqlite3.connect("Database/Consultas.db")
            self.c = self.conn.cursor()
            result = self.c.execute("SELECT * from historial WHERE roll =" + str(delrol))
            row = result.fetchone()
            rmtree(str(row[12]))
            self.c.execute("DELETE from historial WHERE roll=" + str(delrol))
            self.conn.commit()
            pdf()
            self.c.close()
            self.conn.close()
            self.confirmacionEliminacion()
            self.close()
        except (ValueError, Exception):
            self.errorEliminacion()

    def confirmacionEliminacion(self):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Listo")
        messageBox.setIcon(QMessageBox.Information)
        messageBox.setWindowIcon(QIcon("Recursos/database.png"))
        messageBox.setText('Consulta eliminada de historial.')
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()

    @staticmethod
    def errorEliminacion():
        messageBox = QMessageBox()
        messageBox.setWindowTitle("Error")
        messageBox.setIcon(QMessageBox.Warning)
        messageBox.setWindowIcon(QIcon("Recursos/database.png"))
        messageBox.setText("Consulta no eliminada de historial.")
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()


class reportePDF(object):
    def __init__(self, datos):
        super(reportePDF, self).__init__()
        self.titulo = "REPORTE DE CONSULTAS"
        self.cabecera = (("ImagenOriginal", "Imagen Original"), ("Aproximacion", "Aproximacion
            de Personas"),
                         ("Zona", "Zonas Posibles de Aglomeraci\'on"), ("Modelo", "Modelo
                             Seleccionado"),
```

```
                    ("Fecha", "Hora"), ("Year", "Fecha"), ("Ruta", "Ruta"))
        self.datos = datos
        try:
            os.stat("Reportes")
        except (ValueError, Exception):
            os.mkdir("Reportes")
        self.nombrePDF = "Reportes/Reporte de Consultas.pdf"
        self.estilos = getSampleStyleSheet()
        self.ancho = ""
        self.alto = ""

    @staticmethod
    def _encabezadoPiePagina(canva, archivoPDF):
        canva.saveState()
        estilos = getSampleStyleSheet()
        alineacion = ParagraphStyle(name="alineacion", alignment=TA_LEFT, parent=estilos["
            Normal"])
        encabezadoNombre = Paragraph(
            "Estimaci\'on de Densidad de Multitudes de Personas a trav\'es de Visi\'on por
                Computador", estilos["Normal"])
        _, altura = encabezadoNombre.wrap(archivoPDF.width, archivoPDF.topMargin)
        encabezadoNombre.drawOn(canva, archivoPDF.leftMargin, 736)
        FechaConsulta = utcnow().to("local").format("dddd, DD  MMMM  YYYY", locale="es")
        Hora = strftime("%H:%M:%S")
        encabezadoFecha = Paragraph(FechaConsulta + " a las " + Hora, alineacion)
        _, altura = encabezadoFecha.wrap(archivoPDF.width, archivoPDF.topMargin)
        encabezadoFecha.drawOn(canva, archivoPDF.leftMargin, 725)
        piePagina = Paragraph("Reporte generado autom\'aticamente.", estilos["Normal"])
        _, altura = piePagina.wrap(archivoPDF.width, archivoPDF.bottomMargin)
        piePagina.drawOn(canva, archivoPDF.leftMargin, 15 * mm + (0.2 * inch))
        canva.restoreState()

    def convertirDatos(self):
        estiloEncabezado = ParagraphStyle(name="estiloEncabezado", alignment=TA_CENTER,
            fontSize=7, textColor=white,
                                          fontName="Helvetica-Bold", parent=self.estilos["
                                              Normal"])
        estiloNormal = self.estilos["Normal"]
        estiloNormal.alignment = TA_CENTER
        estiloNormal.fontSize = 7
        claves, nombres = zip(*[[k, n] for k, n in self.cabecera])
        encabezado = [Paragraph(nombre, estiloEncabezado) for nombre in nombres]
        nuevosDatos = [tuple(encabezado)]
        for dato in self.datos:
            nuevosDatos.append([Paragraph(str(dato[clave]), estiloNormal) for clave in claves
                ])
        return nuevosDatos

    def Exportar(self):
        alineacionTitulo = ParagraphStyle(name="centrar", alignment=TA_CENTER, fontSize=13,
            leading=10,
                                          textColor=purple, parent=self.estilos["Heading1"])
        self.ancho, self.alto = letter
        convertirDatos = self.convertirDatos()
        tabla = Table(convertirDatos, colWidths=(self.ancho - 100) / len(self.cabecera),
            hAlign="CENTER")
        tabla.setStyle(TableStyle([("BACKGROUND", (0, 0), (-1, 0), purple), ("ALIGN", (0, 0),
            (0, -1), "LEFT"),
                                   ("VALIGN", (0, 0), (-1, -1), "MIDDLE"), ("INNERGRID", (0,
                                       0), (-1, -1), 0.50, black),
                                   ("BOX", (0, 0), (-1, -1), 0.25, black), ]))
        historia = [Paragraph(self.titulo, alineacionTitulo), Spacer(1, 0.16 * inch), tabla]
        archivoPDF = SimpleDocTemplate(self.nombrePDF, leftMargin=50, rightMargin=50, pagesize
            =letter,
```

```
                                        title="Reporte de Consultas", author="Daniel Zambrano")
        try:
            archivoPDF.build(historia, onFirstPage=self._encabezadoPiePagina, onLaterPages=
                self._encabezadoPiePagina,
                                canvasmaker=numeracionPaginas)
        except PermissionError:
            self.errorPDF()

    def errorPDF(self):
        messageBox = QMessageBox(self)
        messageBox.setWindowTitle("Error")
        messageBox.setIcon(QMessageBox.Critical)
        messageBox.setWindowIcon(QIcon("Recursos/Error.png"))
        messageBox.setText("Archivo PDF no generado.")
        Si = messageBox.addButton("Aceptar", QMessageBox.AcceptRole)
        messageBox.setDefaultButton(Si)
        messageBox.exec_()


class numeracionPaginas(canvas.Canvas):
    def __init__(self, *args, **kwargs):
        canvas.Canvas.__init__(self, *args, **kwargs)
        self._saved_page_states = []

    def showPage(self):
        self._saved_page_states.append(dict(self.__dict__))
        self._startPage()

    def save(self):
        numeroPaginas = len(self._saved_page_states)
        for state in self._saved_page_states:
            self.__dict__.update(state)
            self.draw_page_number(numeroPaginas)
            canvas.Canvas.showPage(self)
        canvas.Canvas.save(self)

    def draw_page_number(self, conteoPaginas):
        self.drawRightString(204 * mm, 15 * mm + (0.2 * inch),
                             "P\'agina {} de {}".format(self._pageNumber, conteoPaginas))


def pdf():
    try:
        def dict_factory(canv, row):
            d = {}
            for idx, col in enumerate(canv.description):
                d[col[0]] = row[idx]
            return d

        conn = sqlite3.connect("Database/Consultas.db")
        conn.row_factory = dict_factory
        c = conn.cursor()
        c.execute(
            "SELECT ImagenOriginal, Aproximacion, Zona, Modelo, Fecha, Year, MapaDeDensidad,
                ImagenBinaria,ImagenNumerada, ImagenContorno, MapaCV2, Ruta FROM historial")
        datos = c.fetchall()
        conn.commit()
        c.close()
        conn.close()
        reportePDF(datos).Exportar()
    except (ValueError, Exception):
        pass
```

```
def sqliteExcel():
    FechaConsulta = utcnow().to("local").format("dddd, DD - MMMM - YYYY", locale="es")
    fechaReporte = FechaConsulta.replace("-", "de")
    Hora = strftime("%H:%M:%S")
    celdaConteo = 2
    celdaZonas = 3
    n = 3
    m = 4
    workbook = Workbook('Reportes/Reporte de Consultas.xlsx')
    worksheet = workbook.add_worksheet("Base de Datos")
    alineacion = workbook.add_format({'align': 'center', 'valign': 'vcenter'})
    numbersformat = workbook.add_format({'num_format': '#,##0', 'align': 'center', 'valign': '
        vcenter'})
    font = workbook.add_format()
    font.set_font_size(26)
    font2 = workbook.add_format()
    font2.set_font_size(12)
    conn = sqlite3.connect("Database/Consultas.db")
    c = conn.cursor()
    c.execute("SELECT * FROM historial")
    mysel = c.execute("SELECT * FROM historial ")
    connection = sqlite3.connect("Database/Consultas.db")
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM historial")
    recuento = (len(cursor.fetchall()))
    margenTabla = "A" + str(n) + ":" "M" + str(recuento + m)
    titulo = 'Estimaci\'on de Densidad de Personas a trav\'es de Visi\'on por Computador'
    subtitulo = 'Reporte generado autom\'aticamente el dia '
    worksheet.merge_range('A1:M1', "")
    worksheet.merge_range('A2:M2', "")
    worksheet.write('A1:M1', titulo, font)
    worksheet.write('A2:M2', subtitulo + fechaReporte + " a las " + Hora, font2)
    worksheet.set_column('A:A', 25, alineacion)
    worksheet.set_column('B:B', 25, alineacion)
    worksheet.set_column('C:C', 25, alineacion)
    worksheet.set_column('D:D', 35, alineacion)
    worksheet.set_column('E:E', 25, alineacion)
    worksheet.set_column('F:F', 25, alineacion)
    worksheet.set_column('G:G', 25, alineacion)
    worksheet.set_column('H:H', 35, alineacion)
    worksheet.set_column('I:I', 35, alineacion)
    worksheet.set_column('J:J', 35, alineacion)
    worksheet.set_column('K:K', 35, alineacion)
    worksheet.set_column('L:L', 35, alineacion)
    worksheet.set_column('M:M', 100, alineacion)
    worksheet.add_table(margenTabla, {'total_row': 1, 'columns': [{'total_string': 'Total', '
        header': 'No. Consulta'},
                                                                  {'header': 'Nombre de la
                                                                      Imagen',
                                                                   'total_function': 'count',
                                                                   'format': numbersformat},
                                                                  {'header': 'Conteo de
                                                                      Personas',
                                                                   'total_function': 'sum',
                                                                   'format': numbersformat},
                                                                  {'header': 'Zonas con
                                                                      Aglomeraciones',
                                                                   'total_function': 'sum',
                                                                   'format': numbersformat},
                                                                  {'header': 'Modelo
                                                                      Seleccionado'},
                                                                  {'header': 'Hora'}, {'header
                                                                      ': 'Fecha'},
                                                                  {'header': 'Nombre de Mapa
```

```
                                                                    de Densidad'},
                                                                {'header': 'Nombre de Imagen
                                                                    Binaria'},
                                                                {'header': 'Nombre de Imagen
                                                                    Numerada'},
                                                                {'header': 'Nombre de Imagen
                                                                    con Contorno'},
                                                                {'header': 'Nombre de Mapa
                                                                    de Calor CV2'},
                                                                {'header': 'Ruta de
                                                                    Almacenamiento'}, ]})
    for i, row in enumerate(mysel):
        for j, value in enumerate(row):
            if j == celdaConteo or j == celdaZonas:
                worksheet.write(i + n, j, int(value))
            else:
                worksheet.write(i + n, j, value)
    workbook.close()


class Ayuda(QDialog):
    def __init__(self, *args, **kwargs):
        super(Ayuda, self).__init__(*args, **kwargs)
        loadUi('Ayuda.ui', self)
        self.setWindowFlags(
            QtCore.Qt.Window | QtCore.Qt.CustomizeWindowHint | QtCore.Qt.WindowTitleHint |
                QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowStaysOnTopHint)
        self.setWindowIcon(QIcon("Recursos/ayuda.png"))


class HyperlinkLabel(QLabel):
    def __init__(self, parent=None):
        super().__init__()
        self.setOpenExternalLinks(True)
        self.setParent(parent)


class VerificacionGPU(QDialog):
    def __init__(self, *args, **kwargs):
        super(VerificacionGPU, self).__init__(*args, **kwargs)
        global use_cuda
        self.setWindowFlags(
            QtCore.Qt.Window | QtCore.Qt.CustomizeWindowHint | QtCore.Qt.WindowTitleHint |
                QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowStaysOnTopHint)
        self.setFixedWidth(460)
        self.setFixedHeight(250)
        self.setWindowIcon(QIcon("Recursos/NVIDIA.png"))
        QBtn = QDialogButtonBox.Ok
        self.buttonBox = QDialogButtonBox(QBtn)
        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)
        QBtn2 = QDialogButtonBox.Cancel
        self.buttonBox2 = QDialogButtonBox(QBtn2)
        self.buttonBox2.accepted.connect(self.accept)
        self.buttonBox2.rejected.connect(self.reject)
        self.button = QPushButton('Omitir')
        self.button.setFixedWidth(100)
        self.button.setFixedHeight(25)
        self.button.setDefault(False)
        self.button1 = QPushButton('Salir')
        self.button1.setFixedWidth(100)
        self.button1.setFixedHeight(25)
        self.button1.setDefault(True)
        layout = QGridLayout()
```

```
        self.setWindowTitle("CUDA")
        title = QLabel("Verificacion CUDA")
        font = title.font()
        font.setPointSize(15)
        title.setFont(font)
        label2 = QLabel(self)
        pixmap2 = QPixmap('Recursos/Blanco.png')
        label2.setPixmap(pixmap2)
        label2.resize(pixmap2.width(), pixmap2.height())
        label = QLabel(self)
        pixmap = QPixmap('Recursos/NVIDIA.png')
        label.setPixmap(pixmap)
        label.setOpenExternalLinks(True)
        label.setScaledContents(True)
        label.resize(120, 120)
        label.move(325, 15)
        layout.addWidget(title)
        if use_cuda:
            device = (torch.cuda.current_device())
            nameDevice = torch.cuda.get_device_name(device)
            memoria = torch.cuda.get_device_properties(device).total_memory / (1024 ** 2)
            layout.addWidget(QLabel("CUDA DISPONIBLE"))
            layout.addWidget(QLabel("Tarjeta: " + str(nameDevice)))
            layout.addWidget(QLabel("Memoria Total: " + str(memoria)))
            layout.addWidget(QLabel("Dispositivo: " + str(device)))
            layout.addWidget(self.buttonBox)
            self.setLayout(layout)
        else:
            linkTemplate = 'Instale los drivers de <a href= https://la.nvidia.com/Download/
                index.aspx?lang=la>NVIDIA</a> y <a href=https://developer.nvidia.com/cuda-
                downloads>CUDA</a>.'
            label1 = HyperlinkLabel(self)
            label1.setText(linkTemplate.format())
            layout.addWidget(QLabel())
            layout.addWidget(QLabel())
            layout.addWidget(label1, 5, 0)
            layout.addWidget(self.button, 5, 5)
            layout.addWidget(self.button1, 6, 5)
            layout.addWidget(QLabel("Verifique la disponibilidad de una tarjeta NVIDIA."), 4,
                0)
            layout.addWidget(QLabel("CUDA NO DISPONIBLE"), 2, 0)
            self.setLayout(layout)


class AboutDialog(QDialog):
    def __init__(self, *args, **kwargs):
        super(AboutDialog, self).__init__(*args, **kwargs)
        self.setWindowFlags(
            QtCore.Qt.Window | QtCore.Qt.CustomizeWindowHint | QtCore.Qt.WindowTitleHint |
                QtCore.Qt.WindowCloseButtonHint | QtCore.Qt.WindowStaysOnTopHint)
        self.setFixedWidth(460)
        self.setFixedHeight(250)
        self.setWindowIcon(QIcon("Recursos/Yo.png"))
        QBtn = QDialogButtonBox.Ok
        self.buttonBox = QDialogButtonBox(QBtn)
        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)
        layout = QVBoxLayout()
        self.setWindowTitle("Cr\'editos")
        title = QLabel("Universidad T\'ecnica del Norte")
        font = title.font()
        font.setPointSize(20)
        title.setFont(font)
        label2 = QLabel(self)
```

```
        pixmap2 = QPixmap('Recursos/Blanco.png')
        label2.setPixmap(pixmap2)
        label2.setScaledContents(True)
        label2.resize(pixmap2.width(), pixmap2.height())
        label = QLabel(self)
        pixmap = QPixmap('Recursos/Yo.png')
        label.setPixmap(pixmap)
        label.setScaledContents(True)
        label.resize(120, 120)
        label.move(335, 80)
        layout.addWidget(title)
        layout.addWidget(QLabel("Estimaci\'on de Densidad de Personas a trav\'es de Visi\'on
            por Computador"))
        layout.addWidget(QLabel("Versi\'on: 1.0"))
        layout.addWidget(QLabel("Autor: Daniel David Zambrano Andrade"))
        layout.addWidget(QLabel("Correo: ddzambranoa@utn.edu.ec"))
        layout.addWidget(QLabel("A\~{n}o: 2021"))
        layout.addWidget(self.buttonBox)
        self.setLayout(layout)


stylesheet = """
QScrollArea > QWidget > QWidget
{
    background: none;
    border: 0px;
    margin: 0px 0px 0px 0px;
}

 QScrollBar:vertical
 {
     background-color: #2A2929;
     width: 15px;
     margin: 15px 3px 15px 3px;
     border: 1px transparent #2A2929;
     border-radius: 4px;
 }
 QScrollBar::handle:vertical
 {
     background-color: red;
     min-height: 5px;
     border-radius: 4px;
 }
 QScrollBar::sub-line:vertical
 {
     margin: 3px 0px 3px 0px;
     border-image: url(Recursos/arrow_up.jpg);
     height: 10px;
     width: 10px;
     subcontrol-position: top;
     subcontrol-origin: margin;
 }
 QScrollBar::add-line:vertical
 {
     margin: 3px 0px 3px 0px;
     border-image: url(Recursos/arrow_down.jpg);
     height: 10px;
     width: 10px;
     subcontrol-position: bottom;
     subcontrol-origin: margin;
 }
 QScrollBar::sub-line:vertical:hover,QScrollBar::sub-line:vertical:on
 {
     border-image: url(Recursos/arrow_up.jpg);
```

```
        height: 10px;
        width: 10px;
        subcontrol-position: top;
        subcontrol-origin: margin;
    }
    QScrollBar::add-line:vertical:hover, QScrollBar::add-line:vertical:on
    {
        border-image: url(Recursos/arrow_down.jpg);
        height: 10px;
        width: 10px;
        subcontrol-position: bottom;
        subcontrol-origin: margin;
    }
    QScrollBar::up-arrow:vertical, QScrollBar::down-arrow:vertical
    {
        background: none;
        border-radius: 4px;
        min-height: 5px;
    }
    QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical
    {
        background: none;
        border-radius: 4px;
        min-height: 5px;
    }
"""

app = QApplication(sys.argv)
app.setStyleSheet(stylesheet)
main = VentanaPrincipal()
main.show()
sys.exit(app.exec_())
```

Bibliografía

[1] A. Echeverría, "Covid-19 una nueva pandemia", Sociedad Ecuatoriana de Reanimación Cardiopulmonar, Quito, pp.1-11, 2020.

[2] Organización Mundial de la Salud (WHO, por sus siglas en inglés), 2020, [En línea]. Disponible en: https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public/. [Último acceso: Mayo 13, 2020].

[3] Ministerio de Salud Pública del Ecuador, "Lineamientos de prevención y control para casos sospechosos o confirmados de SARS CoV-2/COVID-19", pp. 1-45, 2020.

[4] Servicio Integrado de Seguridad ECU-911, 2020, [En línea]. Disponible en: https://www.ecu911.gob.ec/objetivos/. [Último acceso: mayo 24, 2020].

[5] D. Vera, E. Martínes, "Conteo de personas en imagen y video mediante la técnica de Viola-Jones a través de clasificadores Haar utilizando software libre", Paraguay, 2014.

[6] H. Idrees, I. Saleemi, C. Seibert, and M. Shah, "Multi-Sources Multi-Scale Counting in Extremely Dense Crowd Images", IEEE CVPR, pp. 2547-2554, 2013.

[7] G. Gao, J. Gao, Q. Liu, Q. Wang, Y. Wang, "CNN-based Density Estimation and Crowd Counting: A Survey", IEEE, pp. 1-25, 2020.

[8] R. Kaplan, M. Yu, "3D Person Tracking in Retail Stores", 2016, [En línea]. Disponible en: https://web.stanford.edu/class/cs231a/prev_projects_2016/3d-person-tracking.pdf. [Último acceso: Mayo 26, 2020].

[9] L. Yuhong, Z, Xiaofan, C. Deming, "CSRNet: Dilated Convolutional Neural Networks Understanding the Highly Congested Scenes", pp. 1-16, 2018.

[10] C. Lon, E. Garnett, J. Miles, "Physical Science: What the Technology Professional Needs to Know", pp. 47, 2000.

[11] N. Dalal, B. Triggs, "Histograms of oriented gradients for human detection", In Computer Vision and Pattern Recognition, vol 1, pp. 886–893, 2005.

[12] A. Chan, N. Vasconcelos, "Bayesian poisson regression for crowd counting", In Computer Vision, pp. 545–551, 2009.

[13] V. Lempitsky, A. Zisserman, “Learning to count objects in images”, In Advances in Neural Information Processing Systems, pp. 1324–1332, 2010.

[14] J. Montaño, “Redes Neuronales Artificiales aplicadas al Análisis de Datos”, Tesis Doctoral, España, pp. 1-275, 2002.

[15] J. Bapu Ahire, “The Artificial Neural Networks Handbook", 2018, [En línea]. Disponible en: https://dzone.com/articles/the-artificial-neural-networks-handbook-part-4. [Último acceso: Junio 1, 2020].

[16] D. Francisco, A. Matías, “Pronóstico del tipo de cambio USD/MXN con redes neuronales de retropropagación", ISSN 1870-4069, pp. 97-110, México, 2017.

[17] H. Gregory, “Digital Image Processing: Principles and Applications”, New York, pp. 1-480, 1994.

[18] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning (Adaptive Computation and machine Learning series)”, pp. 1-716, 2016.

[19] K. Simonyan, A. Zisserman, “Very Deep Convolutional Networks For Large-Scale Image Recognition”, ICLR, pp. 1-14, 2015.

[20] Y. Zhang, D. Zhou, S. Chen, S. Gao, Y. Ma, “Single-Image Crowd Counting via Multi-Column Convolutional Neural Network”, IEEE, vol. 1, pp. 589-597, 2016. [En línea]. Disponible en: https://www.kaggle.com/tthien/shanghaitech-with-people-density-map. [Último acceso: Julio 10, 2020].

[21] Z. Yan, Y. Yuan, W. Zuo, T. Xiao, Y. Wang, S. Wen, and E. Ding, “Perspective-guided convolution networks for crowd counting”, in ICCV, 2019.

[22] Z. Cheng, J. Li, Q. Dai, X. Wu, A. Hauptmann, “Learning spatial awareness to improve crowd counting”, ICCV, 2019.

[23] S. Bai, Z. He, Y. Qiao, “Adaptive dilated network with self-correction supervision for counting”, IEEE CVPR, pp. 4593–4602, 2020.

[24] Biblioteca de Aprendizaje profundo PyTorch, 2021, [En línea]. Disponible en: https://pytorch.org/get-started/locally/. [Último acceso: Diciembre 18, 2020].

[25] R. Farber, M. Kaufmann, “CUDA Application Design and Development”, pp. 241 - 264, 2012, [En línea]. Disponible en: https://developer.nvidia.com/cuda-downloads. [Último acceso: Enero 4, 2021].

[26] Librería de Visión por Computador de Código Abierto, 2021, [En línea]. Disponible en: https://github.com/opencv/opencv. [Último acceso: Enero 5, 2021].