VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA" BELAGAVI-590018, KARNATAKA

PROJECT REPORT
On
"DETECTING CROWD DENSITY MAP AND CROWD COUNTING WITH DRONE SURVEILLANCE USING DEEP LEARNING TECHNIQUES"

Submitted in the partial fulfillment of the requirement of the award of

Bachelor of Engineering
In
Computer Science and Engineering

Submitted By

ALDI ROHITH	**1BO19CS004**
BIJJULA JAYA SIMHA REDDY	**1BO19CS025**
CH. MAHESWAR REDDY	**1BO19CS027**
CHINTAPARTHI RETISH REDDY	**1BO19CS031**

Under The Guidance of
Prof. Padmavathi H G
Associate Professor
Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Brindavan College of Engineering

Dwarakanagar, Bagalur Main Road, Yelahanka, Bengaluru – 560063
Affiliated to VTU Belagavi, Approved by AICTE, New Delhi, India
Accredited 'A' level by NAAC

2022-23

Brindavan College of Engineering

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work entitled " **Detecting Crowd Density Map and Crowd counting using Deep Learning Techniques**" is a bonafied work carried out by

ALDI ROHITH	**1BO19CS004**
BIJJULA JAYA SIMHA REDDY	**1BO19CS025**
CH. MAHESWAR REDDY	**1BO19CS027**
CHINTAPARTHI RETISH REDDY	**1BO19CS031**

in partial fulfillment for the award of **Bachelor of Computer Science and Engineering of Visvesvaraya Technological University, Belagavi** during the year 2022-23. It is certified that all corrections and suggestions indicated for the internal assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect to the work prescribed for the Bachelor of Engineering degree.

…….	…….	…….
Signature of Project Guide	**Name & Signature of HoD**	**Signature of the Principal**
Prof. Padmavathi H G	**Prof. Avinash N**	**Dr. Bhagappa**
Associate Professor	**Head of the Department**	**Principal , BrCE**
Department of CSE	**Department of CSE**	

External Viva

Name of the Examiners	**Signature with Date**
1 _____	_____
2 _____	_____

ABSTRACT

Counting objects in images is one of the fundamental tasks in deep learning . Currently, deep learning (DL) methods provide the state-of-the-art performance in digital image processing. However, they require collecting a lot of annotated data, which is usually time consuming and prone to labeling errors. Crowd counting is a challenging problem due to the scene complexity and scale variation. Although deep learning has achieved great improvement in crowd counting, scene complexity affects the judgment of these methods and they usually regard some objects as people mistakenly; causing potentially enormous errors in the crowd counting result. To address the problem, we propose a novel end-to end model called Convolutional Neural Network based Yolo 5 model and three CNN based algorithms such as MobileNet, ResNet and VGG16 deep learning models. Our Yolo 5 model can adaptively assess the importance of a human head at each pixel location by automatically encoding a confidence map. With the guidance of the confidence map, the position of human head in estimated density map gets more attention to encode the final density map, which can avoid enormous misjudgments effectively. The crowd count can be obtained by integrating the final density map. To encode a highly refined density map, the total crowd count of each image is classified in a designed classification task and we first explicitly map the prior of the population-level category to feature maps. To verify the efficiency of our proposed method, extensive experiments are conducted on three highly challenging datasets. Results establish the superiority of our method over many state-of-the-art methods.

ACKNOWLEDGEMENT

First and foremost we would like to thank God Almighty for giving me the opportunity to undertake this Project work. Without his blessings, it would not have been possible to take up this Project work.

We express our sincere thanks to **Padma Shri Dr. B.R. Shetty, Chairman**, Brindavan Group of Institutions, Bengaluru, for providing excellent facilities in the institution.

We wish to express our sincere thanks to **Dr.Venugopal A R, Director,** Brindavan Group of Institutions for his support and motivation.

We express our sincere thanks to **Dr.Bhagappa, Principal**, Brindavan College of Engineering, Bengaluru, for providing necessary facilities and motivation to carry out project work successfully.

We express our heartfelt gratitude and humble thanks to **Prof. Avinash N, Head of Department, Computer Science and Engineering,** Brindavan College of Engineering, for the constant encouragement and help to carry out project work successfully.

We are profoundly grateful to our guide **Prof. Padmavathi H G**, **Associate Professor, Computer Science and Engineering,** Bengaluru for guiding and having facilitated us to complete my project work successfully.

We express our heartfelt gratitude to the Project coordinator **Prof. Padmavathi H G**, **Associate Professor, Department of Computer Science and Engineering**, Brindavan College of Engineering, Bengaluru for the cooperation and coordination in completing Internship work successfully.

We would like to mention our special thanks to all the faculty members of **Computer Science and Engineering**, Brindavan College of Engineering, Bengaluru for their invaluable support and guidance. We finally thank our family and friends who have been encouraging us constantly and inspiring throughout the completion of our project.

ALDI ROHITH	**1BO19CS004**
BIJJULA JAYASIMHA REDDY	**1BO19CS025**
CH. MAHESWAR REDDY	**1BO19CS027**
CHINTAPARTHI RETISH REDDY	**1BO19CS031**

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

As the phenomenon of crowd congestion is becoming serious, safety- and security-oriented tasks– such as public safety control and traffic safety monitoring– face huge challenges. Manual analysis of the degree of crowd aggregation not only cannot achieve high accuracy butalso will perform low efficiently. In contrast, deep-learning-based methods are more applicableat present since their process not only eliminates manual efforts but also can analyze crowd aggregation accurately and quickly. Among them, crowd estimation at the pixel level through the crowd distribution density maps has achieved tremendous progress. A crowd density map is a kind of image label that can reflect the distribution of crowd heads by processing the headcoordinate value through Gaussian convolution.

As the convolutional layer and pooling layer of Convolutional Neural Network (CNN) strengthen the relationship between pattern recognition and the context in the image, the densityestimation methods of CNN are with strong learning ability. They have achieved high accuracyin dense scenes. The accuracy of crowd counting mainly depends on the quality of the estimateddensity map which is limited by the image scale. Since the convolution kernel of CNN owns astatic size, heads of dynamic scales will worsen the network's performance, resulting in misjudgments and missing judgments. To solve this problem, the common methods are as follows: (1) introducing a multicolumn structure to estimate the crowd of different scales]; (2)introducing the idea of dilated convolution in the field from image segmentation . This is a special convolution for extracting feature information of different scales, consisting of a 33 convolution kernel and a dilated parameter. By setting the dilated parameter to replaceredundant branches of different sizes of convolution kernels, the computational cost of multiscale detection can be reduced; (3) applying different detection methods to regions of different scales in the image. To generate a high-quality density map, spatial continuity shouldbe ensured during the generation process so that the adjacent pixels in the output density graphcan transition smoothly.

Crowd counting by computer vision technology plays an important role in safety management,video surveillance, and urban planning. The method of crowd counting can be also extended toother applications, such as cell counting, animal counting, and vehicle counting. However, dueto the severe occlusion, scale variation, and high density in the crowd scene, crowd counting isstill a challenging task.

To address these problems, a lot of efforts have been done in previous works including detection based methods and regression-based methods. Detection-based methods usually detect the instances of each person with pre-trained detectors. In the sparse crowd scene, they count the crowd accurately, while their accuracies are downgraded in the congested scene. Regression-based methods regress the number of the crowd without detecting people. They implement an implicit mapping between low-level features and crowd counts. However, the location information of the crowd is omitted. So that many CNN-based methods with state-of-the-art results are proposed recently. Most of them map the image to a density map that is morerobust than the hand-crafted features. The quantity and location of the crowd at each pixel location are recorded in the density map. The crowd count can be obtained by integrating the density map.

Although CNN-based methods have achieved significant success in crowd counting, we find an important problem that needs to be solved urgently. Due to the complexity of crowd scenes,CNN-based methods usually mistake some objects as the head of people. As shown in Fig. 1, there are no people inside the red box, however, MCNN regards the dense shrubberies as human heads by mistake, which results in enormous errors of crowd counting. To address the above problem, we propose a novel end to-end model called CAT-CNN. An overview of the proposedCAT-CNN, It contains four modules: Multi-information Handling Module, Confidence Module, Density Map Estimation Module, and Fusion Module. The Multi-information Handling Module is utilized to extract robust features for crowd counting. Motivated by [2, 45], we leverage different convolution kernels to encode the input image at the beginning, thenwe fuse rich hierarchies from different convolutional layers, which is significant for extractingmulti-scale features. In addition, the total crowd count of each image is classified in a designedcrowd count group classifier. To the best of our knowledge, we first explicitly map the weightsof predicted class to feature maps to automatically contribute in encoding a highly refined density map. In the Confidence Module, we classify each pixel to obtain the probability of a human head at each pixel location to encode the confidence map. Unfortunately, the ground- truth confidence map is not provided in present crowd counting datasets.

We propose a simple but effective way to obtain the ground-truth confidence map by pasting the ones template on a binary map. The intensive cost of manual labeling is saved. Meanwhile,to address the problem of unbalanced population distribution, we propose the

weighted BinaryCross-Entropy Loss (BCELoss) to encode a robust confidence map for

population distribution. In the Density Map Estimation Module, the estimated density map is encoded.

1.1 Cconvolutional Neural Network (CNN):

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice.

1.2 CNN Architecture

A CNN's architecture is analogous to the connectivity pattern of the human brain. Just like the brain consists of billions of neurons, CNNs also have neurons arranged in a specific way. In fact, a CNN's neurons are arranged like the brain's frontal lobe, the area responsible for processing visual stimuli. This arrangement ensures that the entire visual field is covered, thus avoiding the piecemeal image processing problem of traditional neural networks, which must be fed images in reduced-resolution pieces. Compared to the older networks, a CNN delivers better performance with image inputs, and also with speech or audio signal inputs.

1.3 CNN layers

A deep learning CNN consists of three layers: a convolutional layer, a pooling layer and a fully connected (FC) layer. The convolutional layer is the first layer while the FC layer is the last. From the convolutional layer to the FC layer, the complexity of the CNN increases. It is this increasing complexity that allows the CNN to successively identify larger portions and more complex features of an image until it finally identifies the object in its entirety.

- **Convolutional layer.** The majority of computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter

inside this layer moving across the receptive fields of the image, checking if a feature is present in the image.

Over multiple iterations, the kernel sweeps over the entire image. After each iteration a *dot product* is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map or convolved feature. Ultimately, the image is converted into numerical values in this layer, which allows the CNN to interpret the image and extract relevant patterns from it.

- **Pooling layer.** Like the convolutional layer, the pooling layer also sweeps a kernel or filter across the input image. But unlike the convolutional layer, the pooling layer reduces the number of parameters in the input and also results in some information loss. On the positive side, this layer reduces complexity and improves the efficiency of the CNN.

- **Fully connected layer.** The FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. Here, *fully connected* means that all the inputs or nodes from one layer are connected to every activation unit or node of the next layer.

All the layers in the CNN are not fully connected because it would result in an unnecessarily dense network. It also would increase losses and affect the output quality, and it would be computationally expensive.

1.4 Applications of convolutional neural networks

Convolutional neural networks are already used in a variety of CV and image recognition applications. Unlike simple image recognition applications, CV enables computing systems to also extract meaningful information from visual inputs (e.g., digital images) and then take appropriate action based on this information.

The most common applications of CV and CNNs are used in fields such as the following:

- **Healthcare.** CNNs can examine thousands of visual reports to detect any anomalous conditions in patients, such as the presence of malignant cancer cells.
- **Automotive**. CNN technology is powering research into autonomous vehicles and self-driving cars.

- **Social media.** Social media platforms use CNNs to identify people in a user's photograph and help the user tag their friends.
- **Retail.** E-commerce platforms that incorporate visual search allow brands to recommend items that are likely to appeal to a shopper.
- **Facial recognition for law enforcement.** Generative adversarial networks (GANs) are used to produce new images that can then be used to train deep learning models for facial recognition
- **Audio processing for virtual assistants.** CNNs in virtual assistants learn and detect user-spoken keywords and process the input to guide their actions and respond to the user.

1.5 YOLO5

YOLO (You Only Look Once) is a real-time object detection system that uses deep neural networks for object detection in images and videos. YOLO v5 is the latest version of the YOLO algorithm, developed by Ultralytics.

YOLO v5 is a single-shot detector that can detect multiple objects in an image or video frame in real-time. It is built using a fully convolutional neural network and is trained on the COCO dataset, which contains more than 80 object categories. YOLO v5 is designed to be fast, accurate, and lightweight, making it suitable for a wide range of applications, including autonomous vehicles, robotics, and surveillance systems.

Some of the key features of YOLO v5 include:

- High accuracy: YOLO v5 achieves state-of-the-art accuracy on several object detection benchmarks.
- Fast inference: YOLO v5 can detect objects in real-time with a high frame rate.
- Lightweight: YOLO v5 has a small memory footprint and can run on devices with limited computational resources.

- Customizable: YOLO v5 can be easily adapted to different object detection tasks by fine-tuning the pre-trained models on specific datasets.

Some of the use cases of YOLO v5 include:

Object detection in autonomous vehicles: YOLO v5 can be used to detect objects such as pedestrians, cars, and traffic signs in real-time, enabling safer and more efficient autonomous driving.

Surveillance systems: YOLO v5 can be used for real-time object detection in video streams, enabling security personnel to monitor large areas for potential threats.

Robotics: YOLO v5 can be used to detect objects and obstacles in real-time, enabling robots to navigate complex environments.

However, there are also some limitations to YOLO v5, including:

- Limited accuracy for small objects: YOLO v5 may struggle to detect small objects in images or videos due to its coarse-grained feature maps.
- Limited ability to handle occlusion: YOLO v5 may struggle to detect objects that are partially occluded by other objects in the image or video.
- Limited ability to handle complex scenes: YOLO v5 may struggle to detect objects in complex scenes with a high degree of clutter or background noise.

1.6 OpenCV

OpenCV (Open Source Computer Vision) is an open-source library of programming functions mainly aimed at real-time computer vision applications. It was originally developed by Intel and is now maintained by the OpenCV community. OpenCV provides a wide range of image and video processing functions, including object detection, image segmentation, feature detection, and optical flow.

Uses:

OpenCV is widely used in a variety of applications, including:

- Object detection and tracking: OpenCV provides a wide range of object detection and tracking algorithms, including Haar cascades, HOG+SVM, and deep learning-based object detection.
- Image and video processing: OpenCV provides a wide range of image and video processing functions, including color space conversion, filtering, morphological operations, and feature extraction.

- Robotics and automation: OpenCV can be used for robot vision applications, including robot navigation, object detection, and localization.
- Medical image analysis: OpenCV can be used for medical image analysis, including segmentation, registration, and feature extraction.
- Augmented reality: OpenCV can be used for creating augmented reality applications, including face and object recognition, and marker-based tracking.

Limitations:

- Despite its many uses, OpenCV has some limitations, including:
- Steep learning curve: OpenCV is a complex library with many functions and algorithms, and it can take time to learn how to use it effectively.
- Performance issues: OpenCV can be computationally expensive, especially when processing large images or video streams.
- Limited deep learning support: Although OpenCV has some deep learning-based algorithms for object detection and classification, its deep learning support is limited compared to other deep learning frameworks such as TensorFlow or PyTorch.
- Limited hardware support: OpenCV's performance can be limited by hardware constraints, and it may not be optimized for some specialized hardware, such as GPUs or FPGAs.
- Limited support for 3D vision: OpenCV is primarily focused on 2D image and video processing and does not have extensive support for 3D vision applications.

CHAPTER 2

LITERATURE SURVEY

Title: Crowd Counting and Density Estimation by Trellis Encoder-Decoder Networks

Year: 2019

Publications: Computer Vision and Pattern Recognition

Description:

In this paper, we have presented a new deep learning architecture, called the trellis encoder-decoder network (TEDnet) for crowd counting. It consists of a multi-scale encoder and a multi-path decoder to generate high-quality density estimation maps. It preserves the localization precision in the encoded feature maps, upon which a multi-path decoder with dense skip connections is adopted to achieve thorough aggregation and fusion of multi-scale features.

The TEDnet is trained with the distributed supervision implemented with the proposed combinatorial loss. Experiments on four benchmarks show that the TEDnet achieves new state-of-the-art performance in terms of both density map quality and crowd counting accuracy.

Crowd counting has recently attracted increasing interest in computer vision but remains a challenging problem. In this paper, we propose a trellis encoder-decoder network (TEDnet) for crowd counting, which focuses on generating high-quality density estimation maps. The major contributions are four-fold. First, we develop a new trellis architecture that incorporates multiple decoding paths to hierarchically aggregate features at different encoding stages, which improves the representative capability of convolutional features for large variations in objects. Second, we employ dense skip connections interleaved across paths to facilitate sufficient multi-scale feature fusions, which also helps TEDnet to absorb the supervision information. Third, we propose a new combinatorial loss to enforce similarities in local coherence and spatial correlation between maps. By distributedly imposing this combinatorial loss on intermediate outputs, TEDnet can improve the back-propagation process and alleviate the gradient vanishing problem. Finally, on four widely-used benchmarks, our TEDnet achieves the best overall performance in terms of both density map quality and counting accuracy, with an improvement up to 14% in MAE metric. These results validate the effectiveness of TED net for crowd counting.

Title: Active Crowd Counting with Limited Supervision

Year: July 2020

Publications: Computer Vision and Pattern Recognition

Description:

To learn a reliable people counter from crowd images, head center annotations are normally required. Annotating head centers is however a laborious and tedious process in dense crowds. In this paper, we present an active learning framework which enables accurate crowd counting with limited supervision: given a small labelling budget, instead of randomly selecting images to annotate, we first introduce an active labelling strategy to annotate the most informative images in the dataset and learn the counting model upon them. The process is repeated such that in every cycle we select the samples that are diverse in crowd density and dissimilar to previous selections. In the last cycle when the labelling budget is met, the large amount of unlabeled data are also utilized: a distribution classifier is introduced to align the labelled data with unlabelled data; furthermore, we propose to mix up the distribution labels and latent representations of data in the network to particularly improve the distribution alignment in-between training samples. We follow the popular density estimation pipeline for crowd counting. Extensive experiments are conducted on standard benchmarks i.e. ShanghaiTech, UCF CC 50, MAll, TRANCOS, and DCC. By annotating limited number of images (e.g. 10% of the dataset), our method reaches levels of performance not far from the state of the art which utilize full annotations of the dataset.

We present an active learning framework for accurate crowd counting with limited supervision. Given a counting dataset, instead of annotating every image, we introduce a partition-based sample selection with weights to label only a few most informative images and learn a crowd regression network upon them. This process is iterated till the labelling budget is reached. Next, rather than learning from only labelled data, the abundant unlabelled data are also exploited: we introduce a distribution alignment branch with latent Mix Up in the network.

Experiments conducted on standard benchmarks show that labeling only 10% of the entire set, our method already performs close to recent state-of-the-art.

Title: Crowd Detection Using Deep Learning

Year: August 2021

Publications: International Research Journal of Engineering and Technology (IRJET)

Description:

This way of life makes life easier for people and increases the use of public services in cities.

We present a CNN-MRF-based method for counting people in still images from various scenes. Crowd density is well represented by the features derived from the CNN model trained for other computer vision tasks. The neighboring local counts are strongly correlated when using the overlapping patches separated strategies. The MRF may use this connection to smooth adjacent local counts for a more accurate overall count. We divide the dense crowd visible image into overlapping patches, and then extract features from each patch image using a deep convolutional neural network, followed by a completely connected neural network to regress the local patch crowd count. Since the local patches overlap, there is a strong connection between the crowd counts of neighboring patches. We smooth the counting effects of the local patches using this connection and the Markov random field.

We present a CNN based method for counting people in still images from various scenes. Crowd density is well represented by the features derived from the CNN model trained for other computer vision tasks. The neighboring local counts are strongly correlated when using the overlapping patches separated strategies. The feature extraction may use this connection to smooth adjacent local counts for a more accurate overall count. Experimental findings show that the proposed method outperforms other recent related methods.

Title: Crowd Density Estimation Using Image Processing: A Survey

Year:2018

Publications: International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 9

Description:

People counting is a crucial subject in video surveillance application. Factors such as severe occlusions, scene perspective distortions in real time application make this task a bit more challenging. The use of Infra-Red (IR) sensors and Channel State Information (CSI) of the WIFI network, which are the classical methods, give the count but have their own range constraints and its limited applicability to controlled environment. Video-surveillance

systems are one of the advanced technologies used to estimate the density of people in a place for security reasons and to obtain the human statistics. The vision based techniques works well when people are in motion and when a high resolution image with clear background are available. This paper presents the state of the art of such image processing algorithms which are used for crowd estimation and their related applications.

This paper presented the crowd density estimation methods which have provided the most satisfactory results. Marana and team has explained about the real time crowd estimation based on texture features and similarly the other authors have provided their ways of dealing the small and tight crowds. The characteristics obtained from the CNN model trained showed a strong capacity to count the crowd, and principles of GrC are used to gestate crowd segmentation issue at different granular levels and other such algorithms of estimation. Further, these methods can be revised to form new algorithms with multiple advantages and can be implemented for a specific application like monitoring the crowd in shopping malls, in uncontrolled environments like bus stands and railway stations thereby preventing congestions and provide comfort.

Title: A Survey of Recent Advances in CNN-based Single Image Crowd Counting and Density Estimation

Year: 2017

Publications: Elsevier**,** Pattern Recognition Letters

Description:

Estimating count and density maps from crowd images has a wide range of applications such as video surveillance, traffic monitoring, public safety and urban planning. In addition, techniques developed for crowd counting can be applied to related tasks in other fields of study such as cell microscopy, vehicle counting and environmental survey. The task of crowd counting and density map estimation is riddled with many challenges such as occlusions, non-uniform density, intra-scene and inter-scene variations in scale and perspective. Nevertheless, over the last few years, crowd count analysis has evolved from earlier methods that are often limited to small variations in crowd density and scales to the current state-of-the-art methods that have developed the ability to perform successfully on a wide range of scenarios. The success of crowd counting methods in the recent years can be largely attributed to deep learning and publications of challenging datasets. In this paper, we provide a comprehensive survey of recent Convolutional Neural Network (CNN) based

approaches that have demonstrated significant improvements over earlier methods that rely largely on hand-crafted representations. First, we briefly review the pioneering methods that use hand-crafted representations and then we delve in detail into the deep learning-based approaches and recently published datasets. Furthermore, we discuss the merits and drawbacks of existing CNN-based approaches and identify promising avenues of research in this rapidly evolving field.

This article presented an overview of recent advances in CNN-based methods for crowd counting and density estimation. In particular, we summarized various methods for crowd counting into traditional approaches (that use hand-crafted features) and CNN-based approaches. The CNN-based approaches are further categorized based on the training process

and the network property. Obviously all the literature on crowd counting cannot be covered, hence, we have chosen a representative subset of the latest approaches for a detailed analysis and review. We also reviewed the results demonstrated by various traditional and CNN-based approaches to conclude that CNN based methods are more adept at handling large density crowds with variations in object scales and scene perspective. Additionally, we observed that incorporating scale and contextual information in the CNN-based methods drastically improves the estimation error. Finally, we identified some of the most compelling challenges and issues that confront research in crowd counting and density estimation using computer vision and machine leaning approaches.

Title: Near Real-time Crowd Counting using Deep Learning Approach

Year: 2020

Publications: Elsevier, Procedia Computer Science

Description:

In the current digital era, at many places crowd counting mechanisms still rely on old-fashioned methods such as maintaining registers, making use of people counters and sensors based counting at entrance. These methods fail in the places where the movement of people is completely random, highly variable and dynamic. These methods are time consuming and tedious. The proposed system is developed for situations where emergency evacuations are required such as fire outbreaks, calamitous events, etc. and making informed decisions on the basis of the number of people such as food, water, detecting congestion, etc. A deep convolution neural network (DCNN) based system can be used for

near real-time crowd counting. The system uses NVIDIA GPU processor to exploit the parallel computing framework to achieve swift and agile processing of the video feed taken through a camera. This work contributes towards constructing a model to detect heads captured by CCTV camera. The model is trained extensively by providing several scenarios such as overlapping heads, partial visibility of heads etc. This system provides significant accuracy in estimating the head count in dense population in reasonably less amount of time. This work presents an approach which will be effective for near real time crowd counting using DCNN. The benefits of the applications includes High Performance Computing through the use NVIDIA GPU parallel framework, a swift and agile method for processing of the video feed taken through a camera with an innovative solutions that can be deployed for disaster management, emergency evacuation without having to configure explicit systems for the same. The proposed system performs admirably in situations where manual counting is simply not possible. Deep learning also enables the system to perform in versatile environments and continuously learn from new inputs. The Experimental results reveal that the proposed methodology achieves promising crowd count predictions almost as good as ground truth. Another major advantage of using the end-to-end application is that no external configurations are required for achieving crowd-count except for the video feed of the particular area.

Title: Crowd Density Estimation in Spatial and Temporal Distortion Environment Using Parallel Multi-Size Receptive Fields and Stack Ensemble Meta-Learning

Year: August 2022

Publications: Symmetry, MDPI.

Description:

The estimation of crowd density is crucial for applications such as autonomous driving, visual surveillance, crowd control, public space planning, and warning visually distracted drivers prior to an accident. Having strong translational, reflective, and scale symmetry, models for estimating the density of a crowd yield an encouraging result. However, dynamic scenes with perspective distortions and rapidly changing spatial and temporal domains still present obstacles. The main reasons for this are the dynamic nature of a scene and the difficulty of representing and incorporating the feature space of objects of varying sizes into a prediction model. To overcome the aforementioned issues, this paper proposes a parallel multi-size receptive field units framework that leverages the majority of the CNN

layer's features, allowing for the representation and participation in the model prediction of the features of objects of all sizes. The proposed method utilizes features generated from lower to higher layers. As a result, different object scales can be handled at different framework depths, and various environmental densities can be estimated. However, the inclusion of the vast majority of layer features in the prediction model has a number of negative effects on the prediction's outcome. Asymmetric non-local attention and the channel weighting module of a feature map are proposed to handle noise and background details and re-weight each channel to make it more sensitive to important features while ignoring irrelevant ones, respectively. While the output predictions of some layers have high bias and low variance, those of other layers have low bias and high variance. Using stack ensemble meta-learning, we combine individual predictions made with lower-layer features and higher-layer features to improve prediction while balancing the tradeoff between bias and variance. The UCF CC 50 dataset and the ShanghaiTech dataset have both been subjected to extensive testing. The results of the experiments indicate that the proposed method is effective for dense distributions and objects of various sizes.

Title: People Counting in High Density Crowds from Still Images
Year: October 16, 2015.
Publications: International Journal of Computer and Electrical Engineering
Description:
They present a method of estimating the number of people in high density crowds (hundreds to thousands of individuals) from still images. Unlike most existing works our method uses only still images to estimate the count. At this scale, we cannot rely on just one set of features for count estimation. We, therefore, use a fusion of multiple sources, viz. interest points (SIFT), Fourier analysis, wavelet decomposition, GLCM features and head detections, to estimate the counts. Each of these sources gives a separate estimate of the count along with confidences and other statistical measures which are then combined to obtain the final estimate. We tested our method on an existing dataset of fifty images containing over 64000 individuals. Further, we added another fifty annotated images of crowds and tested on the complete dataset of hundred images containing over 87000 individuals.
They considered a method for estimating the number of people in extremely dense crowds from still images. The counting problem at this scale has barely been tackled before. We

presented a method that uses information from multiple sources of information (head detections, interest points based counting and texture analysis methods) to estimate the count in an image. Each of these constituent parts gives an independent estimate of the count, along with confidences and other features, which are then fused to give a final estimate. We presented results of extensive tests and experiments we performed. We also introduced a new dataset of still images along with annotations which can complement the existing UCF dataset. The results are very promising and, since the model is extremely simple, it can be applied for real-time counting in critical areas like pilgrimage sites.

Title: A Survey on Human detection in Crowd Density Estimation for Video Surveillance
Year: December, 2021
Publications: International Journal of Mechanical Engineering
Description:
Crowds can be seen in numerous day-to-day life situations and it'll be engaging to recognize, dissect and break the challenges involved in crowd density estimation. The density of a crowd is a vital parameter in several operations like operation of crowd for safety and surveillance for law enforcement, development of public transport structure which have been divided using automated or semi-automated computer vision ways. Mortal discovery in a videotape surveillance system has vast operation areas including suspicious event discovery and mortal exertion recognition. In the current terrain of our society suspicious event discovery is a burning issue. For that reason, this paper proposes a frame for detecting humans in different appearances and acts by generating a mortal point vector. Originally, every pixel of a frame is represented as an objectification of several Gaussians and use a probabilistic system to refurbish the representation. These Gaussian representations are also estimated to classify the background pixels from focus pixels. Shadow regions are excluded from focus by exercising a Hue-Intensity difference value between background and current frame. Also morphological operation is used to remove discontinuities in the focus uprooted from the shadow elimination process. Partial occlusion running is employed by color correlogram to marker objects within a group.

Title: Utilization of Deep Learning-Based Crowd Analysis for Safety Surveillance and Spread Control of COVID-19 Pandemic
Year: 2022

Publications: Intelligent Automation & Soft Computing

Description:

Crowd monitoring analysis has become an important challenge in academic researches ranging from surveillance equipment to people behavior using different algorithms. The crowd counting schemes can be typically processed in two steps, the images ground truth density maps which are obtained from ground truth density map creation and the deep learning to estimate density map from density map estimation. The pandemic of COVID-19 has changed our world in few months and has put the normal human life to a halt due to its rapid spread and high danger. Therefore, several precautions are taken into account during COVID-19 to slowdown the new cases rate like maintaining social distancing via crowd estimation. This manuscript presents an efficient detection model for the crowd counting and social distancing between visitors in the two holy mosques, Al Masjid Al Haram in Mecca and the Prophet's Mosque in Medina. Also, the manuscript develops a secure crowd monitoring structure based on the convolutional neural network (CNN) model using real datasets of images for the two holy mosques. The proposed framework is divided into two procedures, crowd counting and crowd recognition using datasets of different densities. To confirm the effectiveness of the proposed model, some metrics are employed for crowd analysis, which proves the monitoring efficiency of the proposed model with superior accuracy. Also, it is very adaptive to different crowd density levels and robust to scale changes in several places.

Title: Crowd Surge: A Crowd Density Monitoring Solution Using Smart Video Surveillance with Security Vulnerability Assessment

Year: 2022

Publications: Journal of Advances in Information Technology Vol. 13, No. 2, April 2022

Description:

Overcrowding and crowd density monitoring in various places and establishments are being implemented since the pandemic, which helps observe social distancing. This study is about the development of a crowd density solution by utilizing YOLOv4 and Closed-Circuit Television (CCTV) called CrowdSurge. The practice of CCTV has been around for so many years with proven benefits. This has been combined with the state-of-the-art YOLOv4 algorithm that provides high video analytics and object detection performance. With the combination of the said technology and algorithm, it will serve as a smart

surveillance system. A system and mobile application have been developed, and the YOLOv4 deep learning detection model was used to detect various set of scenarios considered to assess if the model executes according to the actions assigned in the experimental set-up. The browser-based application was tested using CVSS or Common Vulnerability Scoring system, which shows that the severity level of most vulnerability is low and has a minor impact on the system. Based on the overall usability testing and statistical results, the respondents are satisfied with both surveillance system and mobile applications developed in terms of functionality, usefulness, and aesthetics. Therefore, using the developed system in real-time surveillance can aid in crowd density reduction in an area.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Existing System

In recent years, crowd counting has drawn much attention and various methods have been proposed, especially in deep learning. Next, we will give these methods some introductions.

1. Traditional detection-based algorithms such as Haar wavelets , HOG , and LBP occupy an important position in early works.
2. Regression-based methods learn a mapping between high level features and crowd counts.
3. CNN-based methods
4. Image based Methods

3.2 Aim & Objective

- **Aim:**

The aim of the project is to propose a novel end-to end model called Crowd Attention Convolutional Neural Network (CAT-CNN) and other deep learning algorithms VGG16, ResNet50 and MobileNet that can adaptively judge the position of a human head at each pixel location by automatically encoding a density map and count the number of Humans.

- **Objective:**

Crowd counting is a challenging problem due to the scene complexity and scale variation. Although

Deep learning has achieved great improvement in crowd counting, scene complexity affects the judgment of these methods and they usually regard some objects as people mistakenly; causing potentially enormous errors in the crowd counting result.

1. The Crowd Dataset is collected from Machine Learning Repository
2. CAT-CNN and other deep learning techniques Crowd Attention Convolutional Neural Network (CAT-CNN) and other deep learning algorithms VGG16, ResNet50 and MobileNet that can adaptively assess the importance of a human head at each pixel location to avoid enormous misjudgments in crowd counting.
3. Density Map Estimator to create high-dimensional feature maps

4.Design a novel classification model that can take input of arbitrary size for training in crowd counting.

5.And we first explicitly map the prior information of the population-level category of images to feature maps to automatically contribute in encoding a highly refined density map.

6.Predicting the Human count based on the density map.

3.3 Proposed System

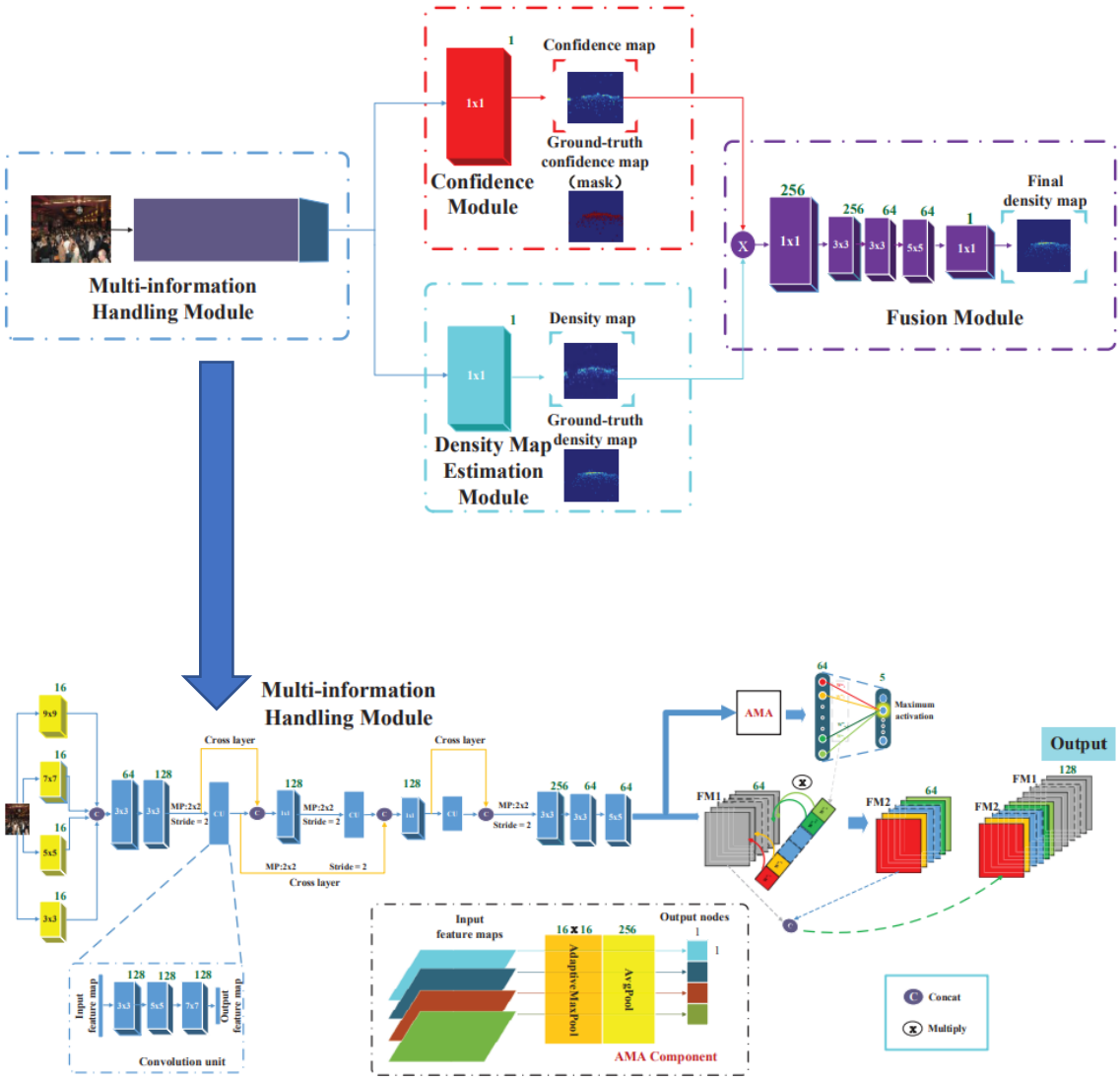

Figure 3.1: The proposed architecture of our CAT-CNN.

An overview of the proposed CAT-CNN is shown in Fig. 1. Our CAT-CNN is composed of three stages. The first stage contains the first module where the features which can automatically adapt different scales and different crowd count groups are extracted. The second stage consists of two modules in the middle to encode confidence map and estimated density map respectively. The third stage contains the final module. With the guidance of the confidence map, final density map is encoded from the estimated density map in this stage. Next, we will elaborate these modules in each stage.

CHAPTER 4

SYSTEM REQUIREMENT & SPECIFICATIONS

Requirements analysis is critical for project development. Requirements must be documented, actionable, measurable, testable and defined to a level of detail sufficient for system design. Requirements can be architectural, structural, behavioral, functional, and functional.

A software requirements specification (SRS) is a comprehensive description of the intended purpose and the environment for software under development.

4.1 Functional Requirements

The tools to execute the Python programs can be many, among that we can go with Visual Studio, Anaconda Navigator (Jupyter Notebook) or any IDLE based on Python. The online tool from Google can be an effective solution towards the execution of Python coding.

4.1.1. Approach 1: Jupyter Notebook (Anaconda Navigator)

This tool is also known as IPython Notebook, and it is Open-Source Distribution Software and provides the platform for development of web applications, computational interactive and specific environment for the users to create notebook documentations. It support for individual code execution , browser based interoperability, can plot various graphs using python libraries and also support for many open source libraries like Bootstrap, JQuery, Tornado, Matplotlib , Seaborn and others.

The features of Jupyter Notebook can be listed as:

- Flexible Notebook Interface
- Useful tool in Machine learning, Deep learning and Ai based Application and model Design.
- Creating and sharing the computational Documents.

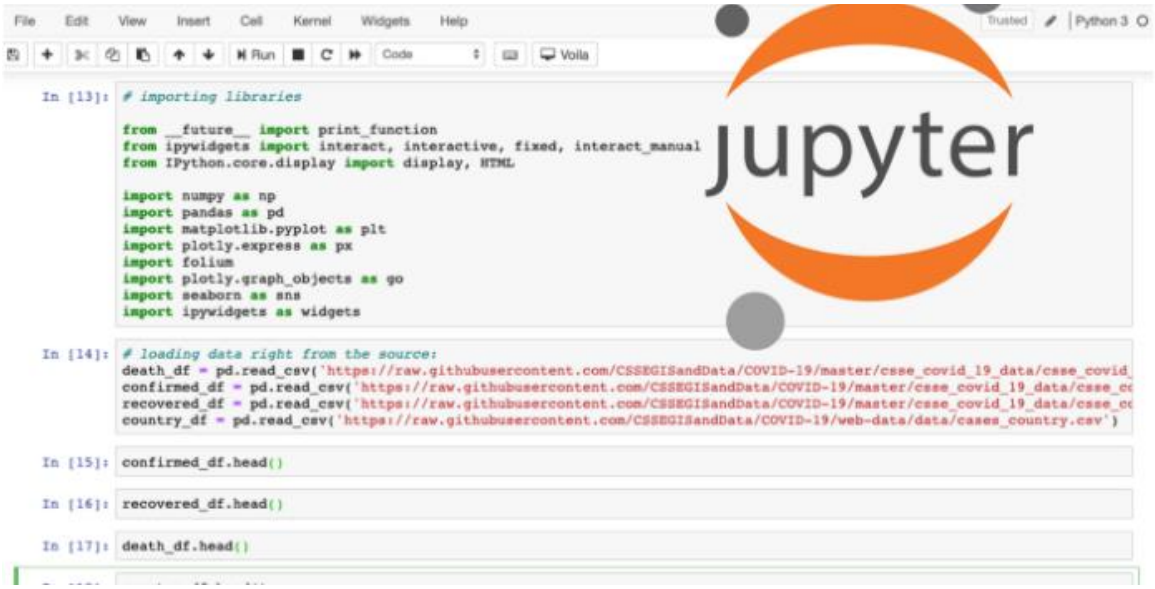

Figure 4.1 Jupyter Notebook Dashboards

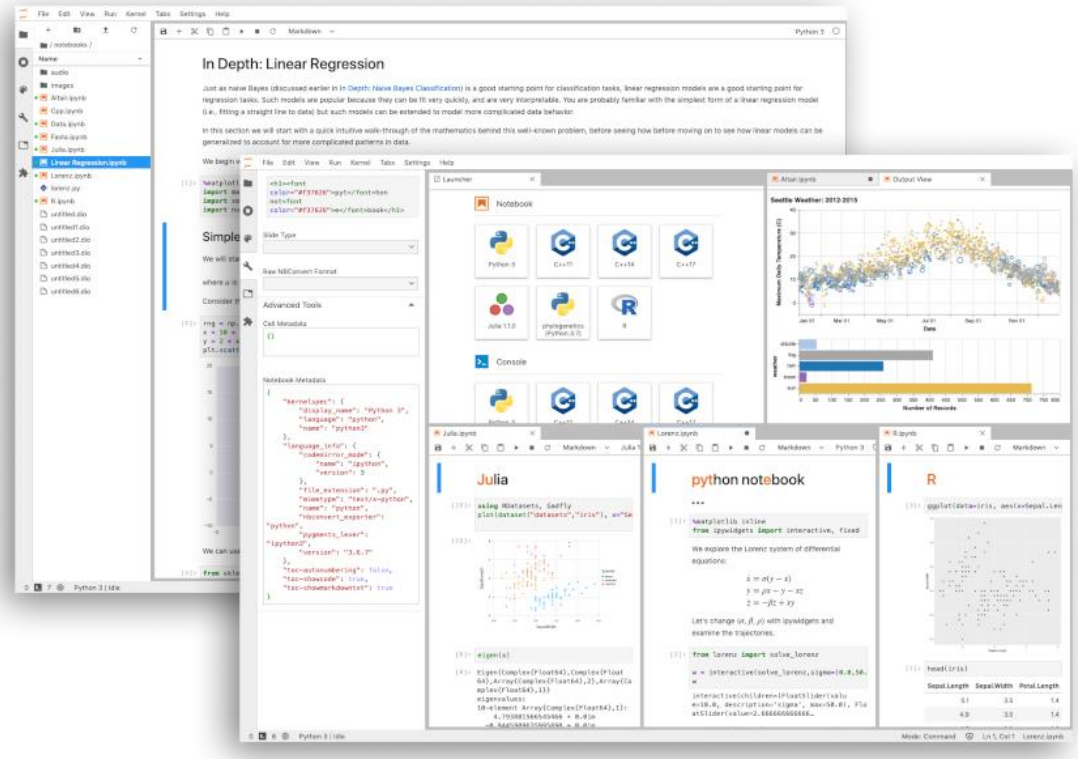

Figure 4.2: Notebook support for plotting

4.1.2 Approach 2: Python IDLE

Python IDLE (Python Integrated Development and Learning Environment) help is writing the code very effectively and efficiently and helpful tool to the Python learning who wants to start from the scratch and beginners can have an advantage to execute the code easily..

This is a powerful interpreter and compiler to run the code.

It's an Interactive Interpreter also known as shell, which executes the python written code, reads the input, evaluate the statements and print the output on the standard output screen provided.

File Editor Help to edit the code, save the program in text files and store as .py file.

Figure 4.3: Python IDLE Download Page

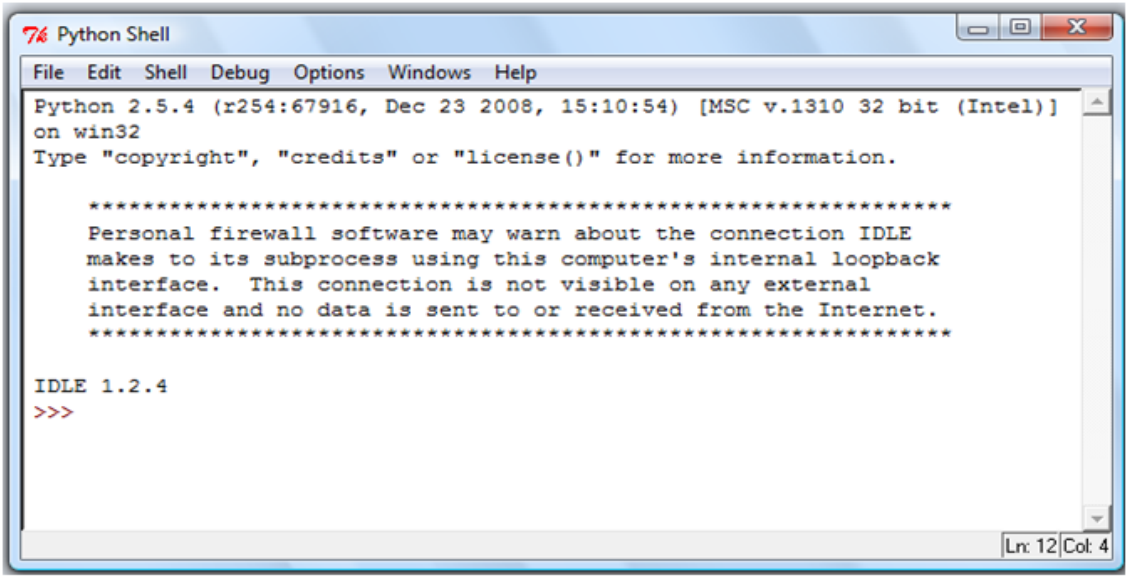

Figure 4.4: Python IDLE prompt to write and execute code.

4.1.3 Approach 3: Google Colab

Google Colab, Also called as Colab in short is a powerful Machine Learning, Deep Learning and Data Analysis Tool that allows mixing the Python script along with text document. Rich support for Plotting the graphs, Diagram, Charts, Import Images, HTML Tags Support and LATEX format API conversions. Additional functional is it works on cloud model where document can be accessed and run on any platform independent of framework design and operating system. The runtime support for Virtual Hard Disk space and 12GB of RAM to execute the application is very excited feature of Colab. The uploading of files is very easy in this application so that it connects to the runtime.

Some of the important feature is:

- Remote Desktop Connection
- Runtime Environment
- Dataset Upload Features
- I/O operations and Operating System API Support
- General Processing Unit (GPU) availability

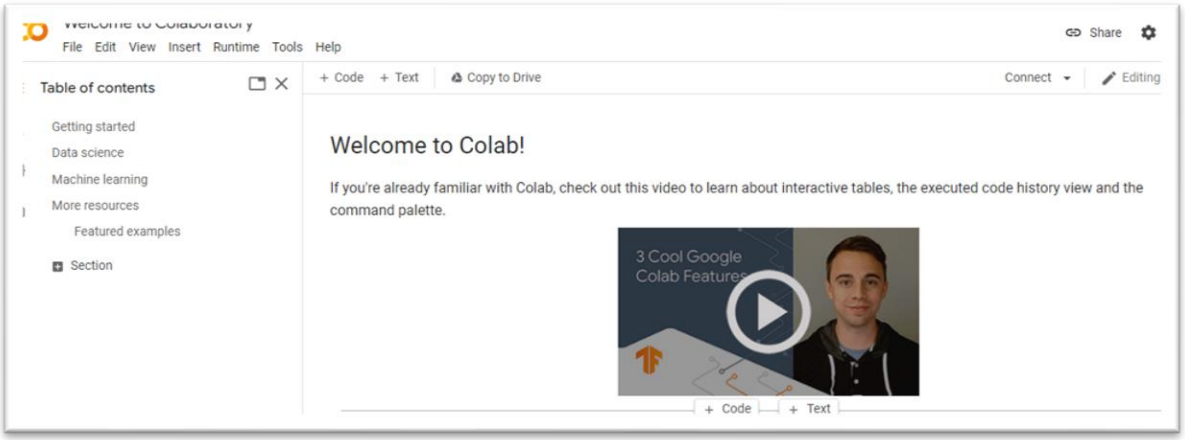

Figure 4.5: Welcome page of Google Colab

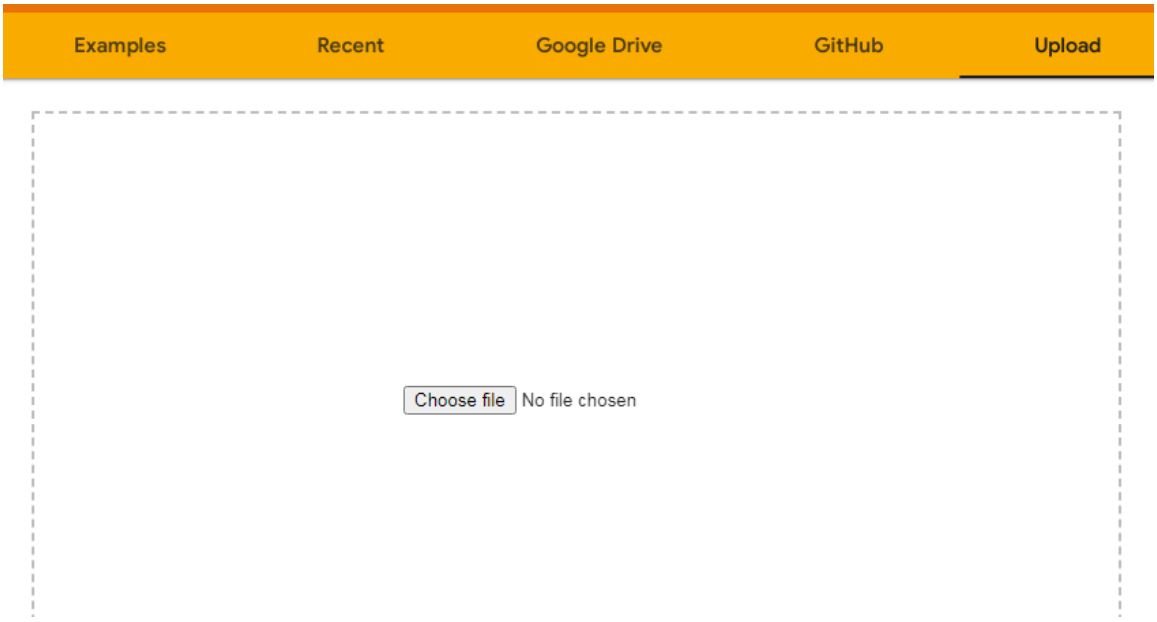

Figure 4.6: Upload the Notebook File

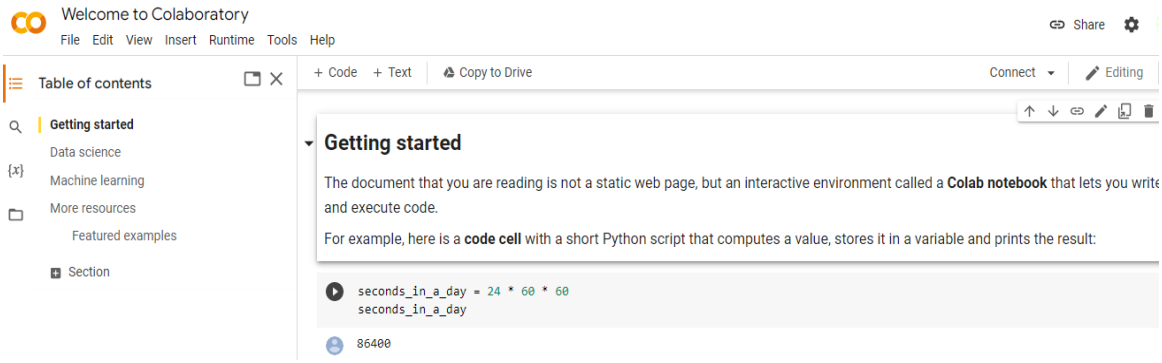

Figure 4.7 Start the Application Page

Requirements analysis is critical for project development. Requirements must be documented, actionable, measurable, testable and defined to a level of detail sufficient for system design. Requirements can be architectural, structural, behavioural, functional, and functional.

A software requirements specification (SRS) is a comprehensive description of the intended purpose and the environment for software under development.

4.2 Software Requirements

Scripting language : Python Programming

Scripting Tool : Anaconda Navigator (Jupyter Notebook) or Google Colab

Operating System : Microsoft Windows 8/ 10 or 11

Dataset : Crowd Dataset

Deep Learning Packages : Numpy, Pandas, Matplotlib, Seaborn Packages etc..

4.3 Hardware Requirements

Processor : 3.0 GHz and Above

Output Devices : Monitor (LCD)

Input Devices : Keyboard

Hard Disk : 1 TB

RAM : 16GB or Above

Graphics : 2GB or Higher

CHAPTER 5
SYSTEM DESIGN

System design is the phase that bridges the gap between problem domain and the existing system in a manageable way. This phase focuses on the solution domain, i.e. “how to implement? “ It is the phase where the SRS document is converted into a format that can be implemented and decides how the system will operate.

In this phase, the complex activity of system development is divided into several smaller sub-activities, which coordinate with each other to achieve the main objective of system development.

System design is the process of defining the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system. It is meant to satisfy specific needs and requirements of a business or organization through the engineering of a coherent and well-running system.

System design gives the following outputs −

- Infrastructure and organizational changes for the proposed system.
- A data schema, often a relational schema.
- Metadata to define the tables/files and columns/data-items.
- A function hierarchy diagram or web page map that graphically describes the program structure.
- Actual or pseudocode for each module in the program.
- A prototype for the proposed system.

5.1 Architecture Diagram

It is also known as high level design that focuses on the design of system architecture. It describes the structure and behavior of the system. It defines the structure and relationship between various modules of system development process.

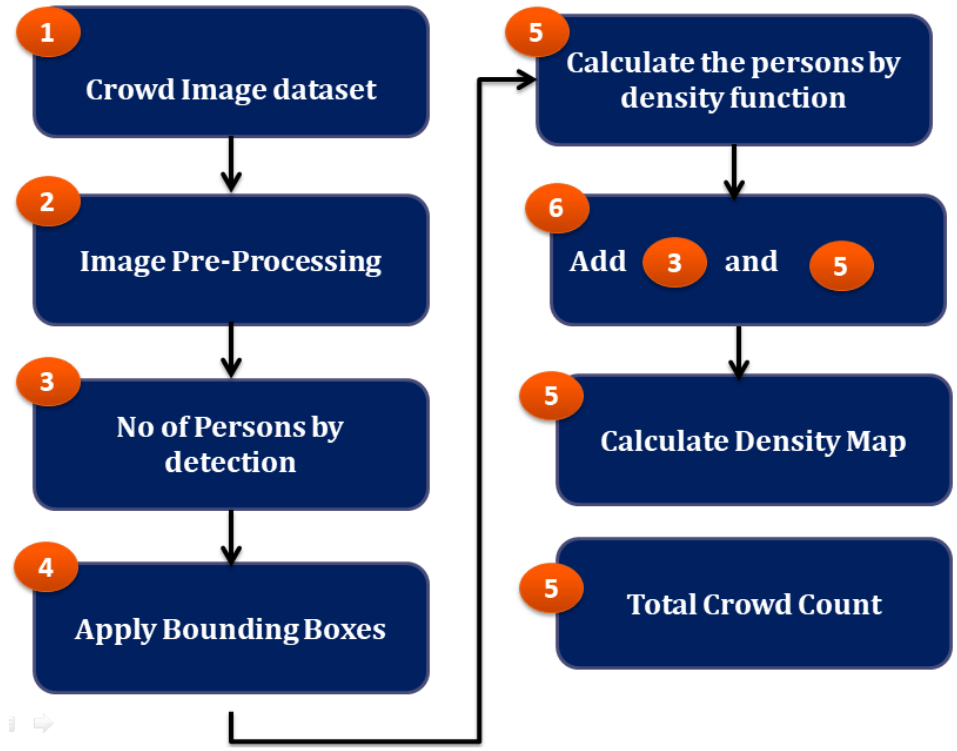

Figure 5.1 Architecture Diagram

Architectural diagrams are used to communicate the design and structure of a system to various stakeholders, including developers, architects, project managers, and clients. They help in understanding the system's overall organization, key components, and their interactions.

5.2 Sequence Diagram

A sequence diagram is a type of interaction diagram in Unified Modeling Language (UML) that illustrates the flow of messages and interactions between objects or components of a system over time. It represents the dynamic behavior of a system by depicting the sequence of actions and the order in which those actions occur. UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

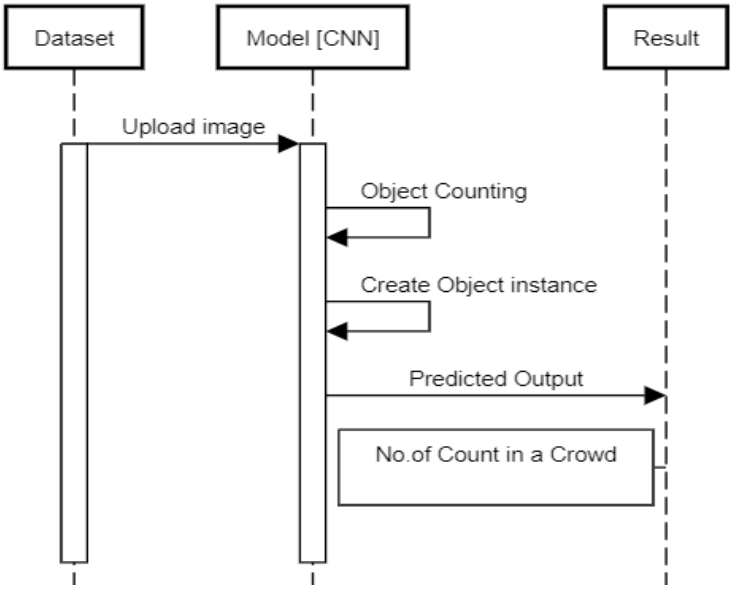

Figure 5.2 Sequence Diagram

5.3 Dataflow Diagram

A flowchart is a diagram that shows an overview of a program. Flowcharts normally use standard symbols to represent the different types of instructions. These symbols are used to construct the flowchart and show the step-by-step solution to the problem. Flowcharts are sometimes known as flow diagrams.

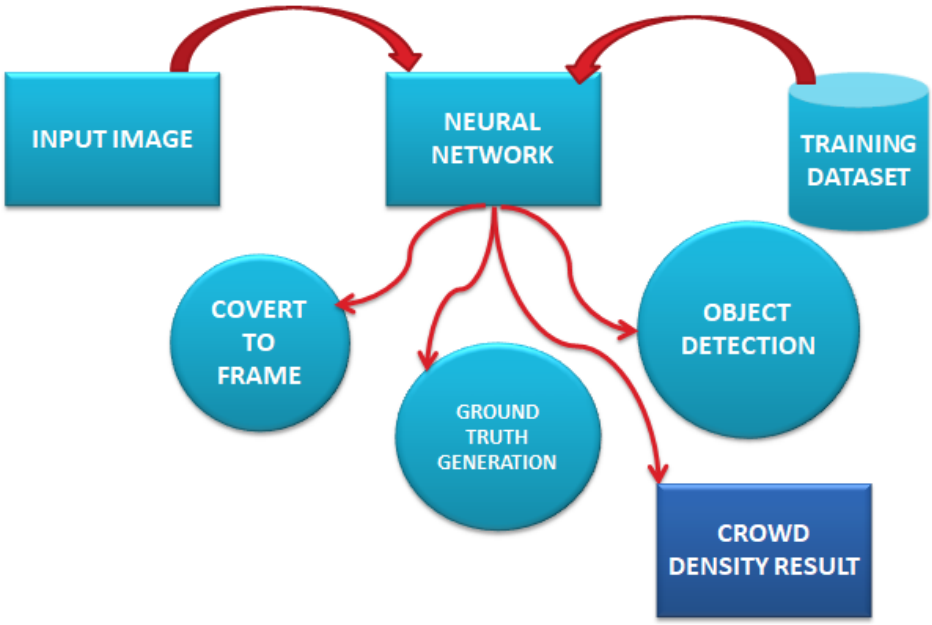

Figure 5.3 Data Flow Diagram

5.4 Class Diagram

A class diagram in the Unified Modeling Language (UML) **is a type of static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

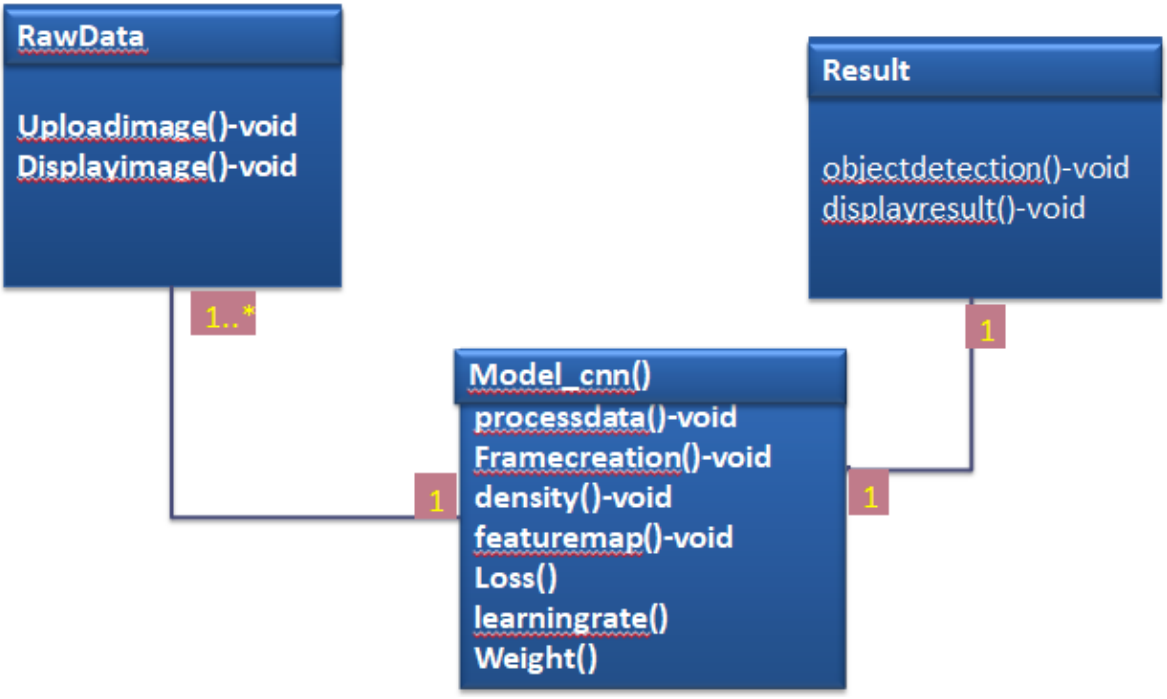

Figure 5.4 Class Diagram

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

System Implementation is basically taken the input from system definition, the system definitions are the actual requirements of the project. System definitions consist of System requirements (Functional and Non-Functional requirements), the architectural Design, Design Features, Specifications and complete blueprint of development. The System Implementation is the process where user has to implement the application with the help of all components, defined at design phase of the project. The implementation phase is the final phase of project development where the system is executed in a Real-time environment.

Convolutional Neural Networks (CNNs) can be used for detecting image forgery by analyzing the visual features of the image and identifying inconsistencies that suggest tampering. Here are the general steps that a CNN model may follow to detect fake or real images:

- Data preparation: A dataset of real and fake images is collected, and each image is labeled as either real or fake. The dataset is then split into training, validation, and testing sets.
- Preprocessing: The images are preprocessed to normalize the pixel values and resize them to a standard size that can be processed efficiently by the CNN.
- CNN architecture: A CNN model is designed and trained on the training set to learn the visual features that distinguish real and fake images. The architecture may include convolutional layers, pooling layers, and fully connected layers.
- Training: The CNN is trained on the labeled dataset using backpropagation and stochastic gradient descent to minimize the classification error.
- Validation: The validation set is used to monitor the performance of the CNN during training and to prevent overfitting.
- Testing: The testing set is used to evaluate the final performance of the CNN on unseen data.

- Postprocessing: The output of the CNN is postprocessed to identify the regions of the image that are likely to be forged.

Overall, CNNs can be effective in detecting image forgery by learning the visual features that distinguish real and fake images. However, the performance of the model may be affected by factors such as the quality of the training data and the complexity of the forgery techniques used. Therefore, it is important to carefully design and evaluate the CNN model for each specific application.

The modules in System Implementation are:

- Data Collection (image)
- Data pre-processing
- Data Visualization
- Building Deep Learning model and prediction

6.2 Code Implementation

6.2.1 Importing the Package

Here importing the necessary packages

```
import os
import cv2
import ctypes
import logging

from time import sleep

from tkinter import *
import tkinter.messagebox as tkMessageBox
import tkinter.filedialog as filedialog

from PIL import ImageTk, Image


from apscheduler.schedulers.background import BackgroundScheduler
```

[a]

```
import logging
logging.basicConfig()
logging.getLogger('apscheduler').setLevel(logging.ERROR)

from nwpu.nwpu_count import nwpu_count
import torch
from nwpu.config import cfg
from nwpu.models.CC import CrowdCounter

from classifier.classifier import predict_density
```

[b]

Figure 6.1: Importing the Package

In figure 6.1, here importing required packages such as pandas, Numpy, Matplotlib etc.

6.2.2 Initialize the path & model config to cpu

```
model_path = './nwpu/exp/MCNN-all_ep_907_mae_218.5_mse_700.6_nae_2.005.pth'
model = CrowdCounter(cfg.GPU_ID, 'MCNN')

model.load_state_dict(torch.load(model_path,map_location=torch.device('cpu')))
```

Figure 6.2: Initialization.

```
window = Tk()
window.title("Crowd Counting")

user32 = ctypes.windll.user32
user32.SetProcessDPIAware()
[w, h] = [user32.GetSystemMetrics(0), user32.GetSystemMetrics(1)]
w = w-100
h = h-150
window.geometry(f"{w}x{h}+{25}+{25}")
#window.resizable(0,0)

loading_frame = Frame(window, bg="#FFFFFF", width=w, height=h)
select_frame = Frame(window, bg="#FFFFFF", width=w, height=h)
select_image_frame = Frame(window, bg="#FFFFFF", width=w, height=h)
select_video_frame = Frame(window, bg="#FFFFFF", width=w, height=h)
select_stream_frame = Frame(window, bg="#FFFFFF", width=w, height=h)
select_image_result = Frame(window, bg="#FFFFFF", width=w, height=h)
capture_image_result = Frame(window, bg="#FFFFFF", width=w, height=h)
select_video_result = Frame(window, bg="#FFFFFF", width=w, height=h)
capture_video_result = Frame(window, bg="#FFFFFF", width=w, height=h)
capture_stream_result = Frame(window, bg="#FFFFFF", width=w, height=h)
```

Figure 6.3: Loading the window

```
def CaptureStreamResult(stream):
    global result, count_result
    result, count_result = "", 0
    capture_stream_result.place(x=0, y=0)

    image_frame = Frame(window, width=1080, height=720, borderwidth=4, bg='black')
    image_frame.place(x=370, y=80)
    densiity_label = Label(capture_stream_result, text=f"Density : {result}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    densiity_label.place(x=550, y=15)
    count_label = Label(capture_stream_result, text=f"Count : {int(count_result)}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    count_label.place(x=1050, y=15)

    cap = cv2.VideoCapture(stream)
    def show_frame():
        try:
            ret, frame = cap.read()
            if ret:
                global frame_copy
                frame_copy = frame.copy()
                cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
                img = Image.fromarray(cv2image)
                img = img.resize((1080, 720))
```

```
                display1.imgtk = imgtk
                display1.configure(image=imgtk)
                window.after(10, show_frame)
        except Exception as e:
            print(e)

display1 = Label(image_frame)
display1.grid(row=1, column=0)

show_frame()

if result == "":
    result = predict_density(frame_copy)
    densiity_label.config(text=f"Density : {result}")
    print(result)
```

[b]

```
def start_detection():
    count_result = nwpu_count(frame_copy, model)
    print(count_result)
    count_label.config(text=f"Count : {int(count_result)}")

scheduler = BackgroundScheduler()
scheduler.add_job(start_detection, 'interval', seconds=3)
scheduler.start()

def back():
    result = ""
    count_result = 0
    cap.release()
    image_frame.place_forget()
    capture_stream_result.place_forget()
    scheduler.shutdown()
    densiity_label.place_forget()
    count_label.place_forget()
    SelectStreamScreen()

Button(capture_stream_result, text = "BACK", font = ("Agency FB", 14, "bold"), relief = FLAT, bd = 0, width=20, fg="#FFFFFF",
       bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:back()).place(x=25, y = 25)
```

[c]

```
def SelectStreamScreen():
    select_stream_frame.place(x=0, y=0)

    Label(select_stream_frame, text="Crowd Counting", font=("Comic Sans MS", 75, "bold"), bg="#FFFFFF").place(x=150, y=200)

    def capture_stream_ip():
        stream_ip = stream_ip_entry.get()
        if len(stream_ip) == 0:
                result = tkMessageBox.showinfo("Crowd Counting", "Enter Stream IP!", icon= "warning")
        else:
            CaptureStreamResult(stream_ip)

    stream_ip_label = Label(select_stream_frame, text="Stream IP", font = ("Agency FB",20,"bold"),relief = FLAT, fg="black", bg="#FFFFFF")
    stream_ip_label.place(x = 100, y = 225)
    stream_ip_entry = Entry(select_stream_frame, font = ("Agency FB",20,"normal"), highlightthickness=2,
                            bg="#FFFFFF", fg="black", highlightcolor="#006EFF", selectbackground="black", width=30)
    stream_ip_entry.place(x=200, y=225)

    Button(select_stream_frame, text = "Start Stream", font = ("Agency FB", 28, "bold"), relief = FLAT, bd = 0,
           width=20, fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF",
           command=lambda:capture_stream_ip()).place(x=25, y = 10)
```

[d]

Figure 6.4: Loading the streaming code

```
def SelectVideoResult():
    select_video_result.place(x=0, y=0)
    global result, count_result
    result, count_result = "", 0

    selected_video = filedialog.askopenfilename(title="Select file", filetypes=( ("Video Files",(".mp4",".avi")),("All Files", "*.*")))
    if selected_video == "":
        select_video_result.place_forget()
        SelectVideoScreen()

    image_frame = Frame(window, width=1080, height=720, borderwidth=4, bg='black')
    image_frame.place(x=370, y=80)
    densiity_label = Label(select_video_result, text=f"Density : {result}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    densiity_label.place(x=550, y=15)
    count_label = Label(select_video_result, text=f"Count : {int(count_result)}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    count_label.place(x=1050, y=15)

    cap = cv2.VideoCapture(selected_video)
```

[a]

```
def show_frame():
    try:
        ret, frame = cap.read()
        if ret:
            global frame_copy
            frame_copy = frame.copy()
            frame = cv2.flip(frame, 1)
            cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
            img = Image.fromarray(cv2image)
            img = img.resize((1080, 720))
            imgtk = ImageTk.PhotoImage(image=img)

            display1.imgtk = imgtk
            display1.configure(image=imgtk)
            window.after(10, show_frame)
    except Exception as e:
        print(e)

display1 = Label(image_frame)
display1.grid(row=1, column=0)

show_frame()
```

[b]

```
def start_detection():
    count_result = nwpu_count(frame_copy, model)
    print(count_result)
    count_label.config(text=f"Count : {int(count_result)}")


scheduler = BackgroundScheduler()
scheduler.add_job(start_detection, 'interval', seconds=3)
scheduler.start()

def back():
    result = ""
    count_result = 0
    cap.release()
    scheduler.shutdown()
    image_frame.place_forget()
    select_video_result.place_forget()
    densiity_label.place_forget()
    count_label.place_forget()
    SelectVideoScreen()

Button(select_video_result, text = "BACK", font = ("Agency FB", 14, "bold"),
       relief = FLAT, bd = 0, width=20, fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF",
       command=lambda:back()).place(x=25, y = 25)
```

[c]

```
def CaptureVideoResult():
    global result, count_result
    result, count_result = "", 0
    capture_video_result.place(x=0, y=0)

    image_frame = Frame(window, width=1080, height=720, borderwidth=4, bg='black')
    image_frame.place(x=370, y=80)
    densiity_label = Label(capture_video_result, text=f"Density : {result}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    densiity_label.place(x=550, y=15)
    count_label = Label(capture_video_result, text=f"Count : {int(count_result)}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    count_label.place(x=1050, y=15)

    cap = cv2.VideoCapture(0)

    def show_frame():
        try:
            ret, frame = cap.read()
            if ret:
                global frame_copy
                frame_copy = frame.copy()
                frame = cv2.flip(frame, 1)
                cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
                img = Image.fromarray(cv2image)
                img = img.resize((1080, 720))
                imgtk = ImageTk.PhotoImage(image=img)
```

[d]

```
            display1.imgtk = imgtk
            display1.configure(image=imgtk)
            window.after(10, show_frame)
    except Exception as e:
        print(e)

display1 = Label(image_frame)
display1.grid(row=1, column=0)

show_frame()

if result == "":
    result = predict_density(frame_copy)
    densiity_label.config(text=f"Density : {result}")
    print(result)
```

[e]

Figure 6.5: Loading the detection from Video code

```
def SelectImageResult():
    select_image_result.place(x=0, y=0)

    selected_image = filedialog.askopenfilename(title="Select file", filetypes=( ("Image Files",(".jpg",".png",".jpeg")),("All Files", "*.*")))
    if selected_image == "":
        select_image_result.place_forget()
        SelectImageScreen()

    image = cv2.imread(selected_image)

    cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
    img = Image.fromarray(cv2image)
    img = img.resize((1080, 720))
    imgtk = ImageTk.PhotoImage(image=img)

    image_frame = Frame(window, width=1080, height=720, borderwidth=4, bg='black')
    image_frame.place(x=370, y=80)

    display1 = Label(image_frame)
    display1.grid(row=1, column=0)
    display1.imgtk = imgtk
    display1.configure(image=imgtk)
```

[a]

```
display1 = Label(image_frame)
display1.grid(row=1, column=0)
display1.imgtk = imgtk
display1.configure(image=imgtk)


result = predict_density(image)   # PREDICT DENSITY
density_label = Label(select_image_result, text=f"Density : {result}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
density_label.place(x=550, y=15)

count_result = nwpu_count(image, model)


count_label = Label(select_image_result, text=f"Count : {int(count_result)}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
count_label.place(x=1050, y=15)
```

[b]

```
def back():

    image_frame.place_forget()
    select_image_result.place_forget()
    density_label.place_forget()
    count_label.place_forget()
    SelectImageScreen()

Button(select_image_result, text = "BACK", font = ("Agency FB", 14, "bold"), relief = FLAT, bd = 0, width=20,
       fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:back()).place(x=25, y = 25)

def CaptureImageResult():
capture_image_result.place(x=0, y=0)

image_frame = Frame(window, width=1080, height=720, borderwidth=4, bg='black')
image_frame.place(x=370, y=80)

cap = cv2.VideoCapture(0)
def show_frame():
    try:
        ret, frame = cap.read()
        if ret:
            global frame_copy
            frame_copy = frame.copy()
            frame = cv2.flip(frame, 1)
            cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
```

[c]

```
cap = cv2.VideoCapture(0)
def show_frame():
    try:
        ret, frame = cap.read()
        if ret:
            global frame_copy
            frame_copy = frame.copy()
            frame = cv2.flip(frame, 1)
            cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
            img = Image.fromarray(cv2image)
            img = img.resize((1080, 720))
            imgtk = ImageTk.PhotoImage(image=img)

            display1.imgtk = imgtk
            display1.configure(image=imgtk)
            window.after(10, show_frame)
    except Exception as e:
        print(e)

display1 = Label(image_frame)
display1.grid(row=1, column=0)
```

[d]

```
def capture_image():
    global density_label, count_label
    result = predict_density(frame_copy)   # PREDICT DENSITY
    density_label = Label(capture_image_result, text=f"Density : {result}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    density_label.place(x=550, y=15)

    count_result = nwpu_count(frame_copy, model)

    count_label = Label(capture_image_result, text=f"Count : {count_result}", font=("Comic Sans MS", 25, "bold"), bg="#FFFFFF")
    count_label.place(x=1050, y=15)

    capture_button.place_forget()
    cap.release()

capture_button = Button(capture_image_result, text = "CAPTURE", font = ("Agency FB", 14, "bold"), relief = FLAT, bd = 0, width=20,
                        fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:capture_image())
capture_button.place(x=900, y = 850)

show_frame()
```

[e]

```
    def back():
        cap.release()
        image_frame.place_forget()
        capture_image_result.place_forget()
        density_label.place_forget()
        count_label.place_forget()
        SelectImageScreen()

    Button(capture_image_result, text = "BACK", font = ("Agency FB", 14, "bold"), relief = FLAT, bd = 0, width=20,
           fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:back()).place(x=25, y = 25)


def SelectImageScreen():
    select_image_frame.place(x=0, y=0)

    Label(select_image_frame, text="Crowd Counting", font=("Comic Sans MS", 75, "bold"), bg="#FFFFFF").place(x=50, y=200)

    Button(select_image_frame, text = "Browse Image", font = ("Agency FB", 28, "bold"), relief = FLAT, bd = 0, width=10,
           fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:SelectImageResult()).place(x=150, y = 100)

    Button(select_image_frame, text = "Click Image", font = ("Agency FB", 28, "bold"), relief = FLAT, bd = 0, width=10,
           fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:CaptureImageResult()).place(x=450, y = 100)
```

[f]

Figure 6.6: Loading the detection from Video code.

```
def SelectScreen():
  select_frame.place(x=0, y=0)

  Label(select_frame, text="Crowd Counting", font=("Comic Sans MS", 75, "bold"), bg="#FFFFFF").place(x=150, y=200)

  Button(select_frame, text = "From Image", font = ("Agency FB", 28, "bold"), relief = FLAT, bd = 0, width=15,
         fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:SelectImageScreen()).place(x=70, y = 500)

  Button(select_frame, text = "From Video", font = ("Agency FB", 28, "bold"), relief = FLAT, bd = 0, width=15,
         fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:SelectVideoScreen()).place(x=470, y = 500)

  Button(select_frame, text = "Live Stream", font = ("Agency FB", 28, "bold"), relief = FLAT, bd = 0, width=15,
         fg="#FFFFFF", bg='black', activebackground = "#E3E3E3",activeforeground = "#006EFF", command=lambda:SelectStreamScreen()).place(x=870, y = 500)
```

Figure 6.7: Loading the selection code.

```
def LoadingScreen():
  loading_frame.place(x=0, y=0)

  Label(loading_frame, text="Crowd Counting", font=("Comic Sans MS", 75, "bold"), bg="#FFFFFF").place(x=100, y=200)

  for i in range(28):
    Label(loading_frame, bg="#574D72",width=2,height=1).place(x=(i+4)*50,y=690)

  def play_animation():
    for j in range(28):
      Label(loading_frame, bg= 'black',width=2,height=1).place(x=(j+4)*50,y=690)
      sleep(0.07)
      loading_frame.update_idletasks()
    else:
      loading_frame.place_forget()
      SelectScreen()

  loading_frame.update()
  play_animation()
```

Figure 6.8: Loading the loading code.

```
LoadingScreen()
# SelectScreen()


window.configure(background='#FFFFFF')
window.mainloop()
```

Figure 6.9: Loading the initialized code.

1.2Front End Application Design

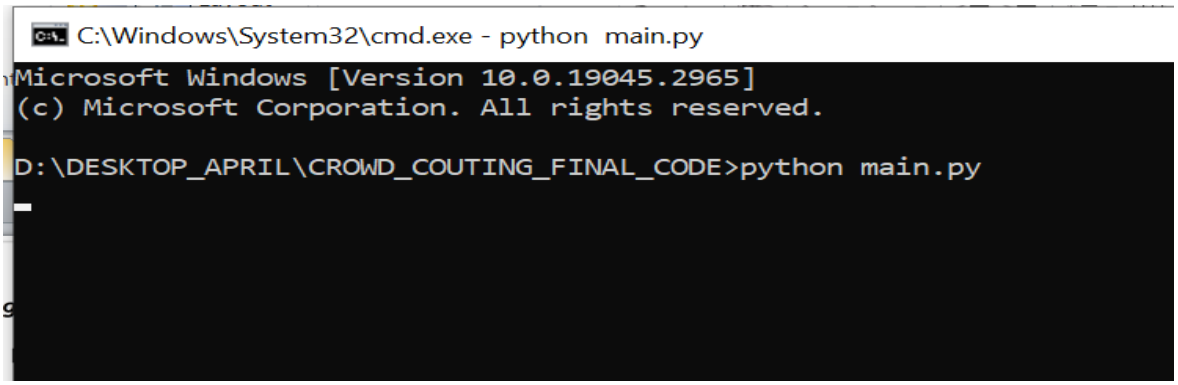

Figure 6.10 Project Initialization

Initialize the application through the command prompt

python main.py

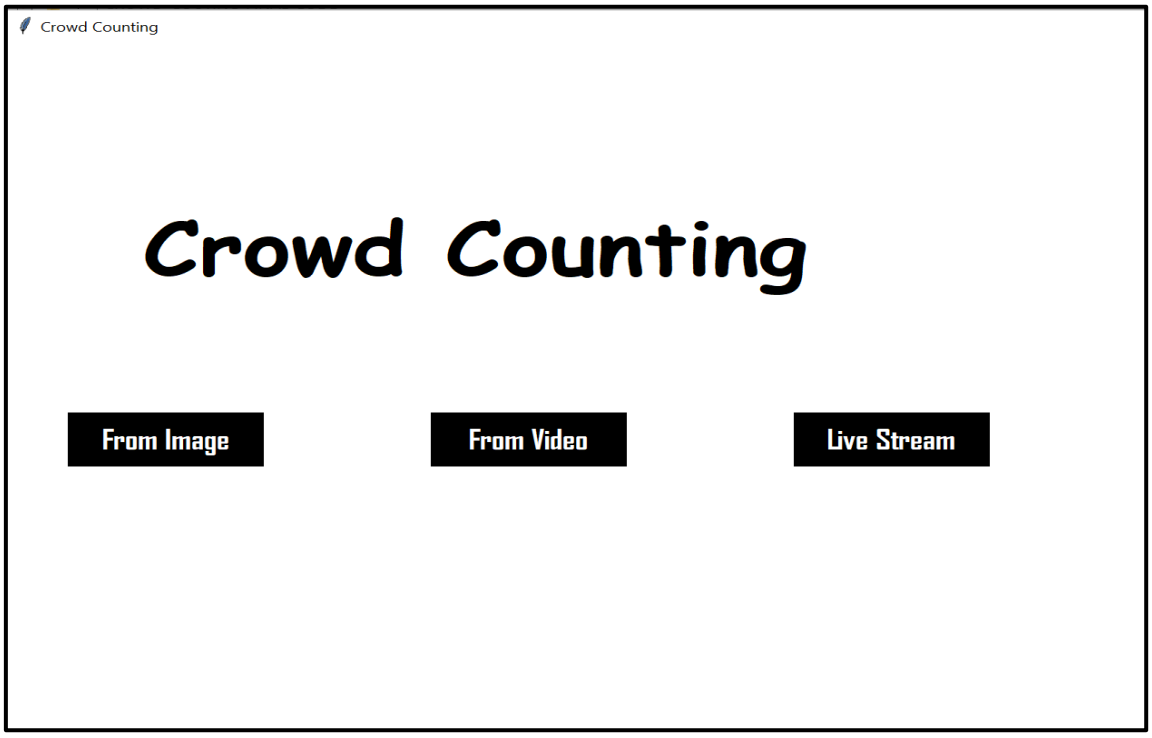

Figure 6.11 Crowd Counting Screenshot

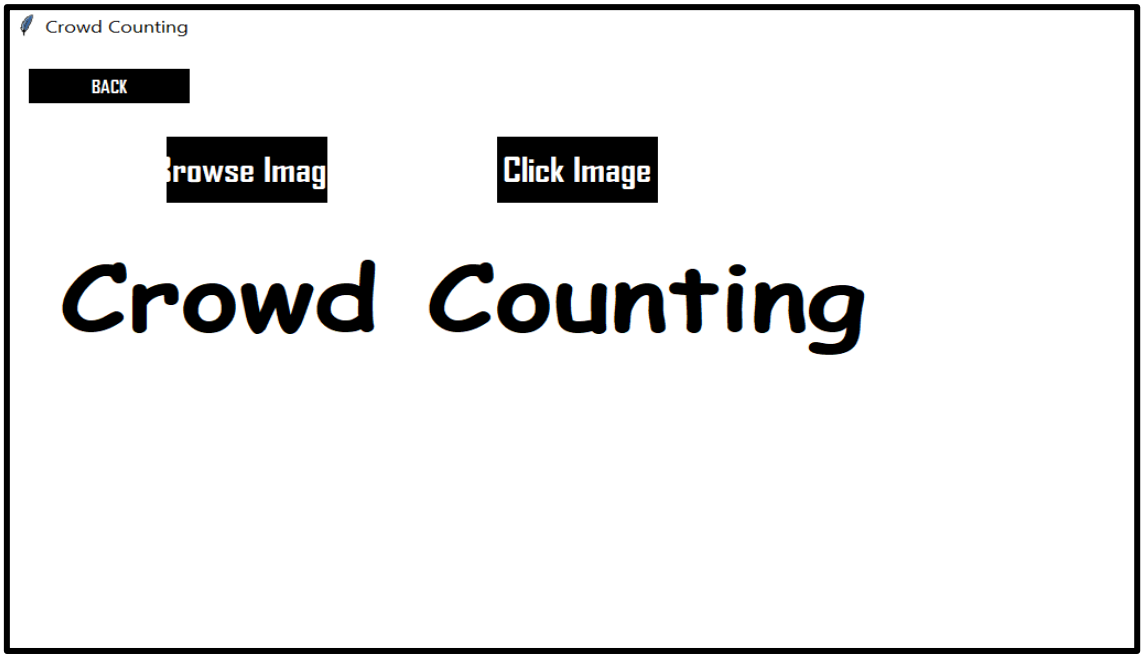

Figure 6.12 Crowd Counting Screenshot

The main application is having three types of crowd counting. 1. From Image 2. From Video and 3. Live Stream.

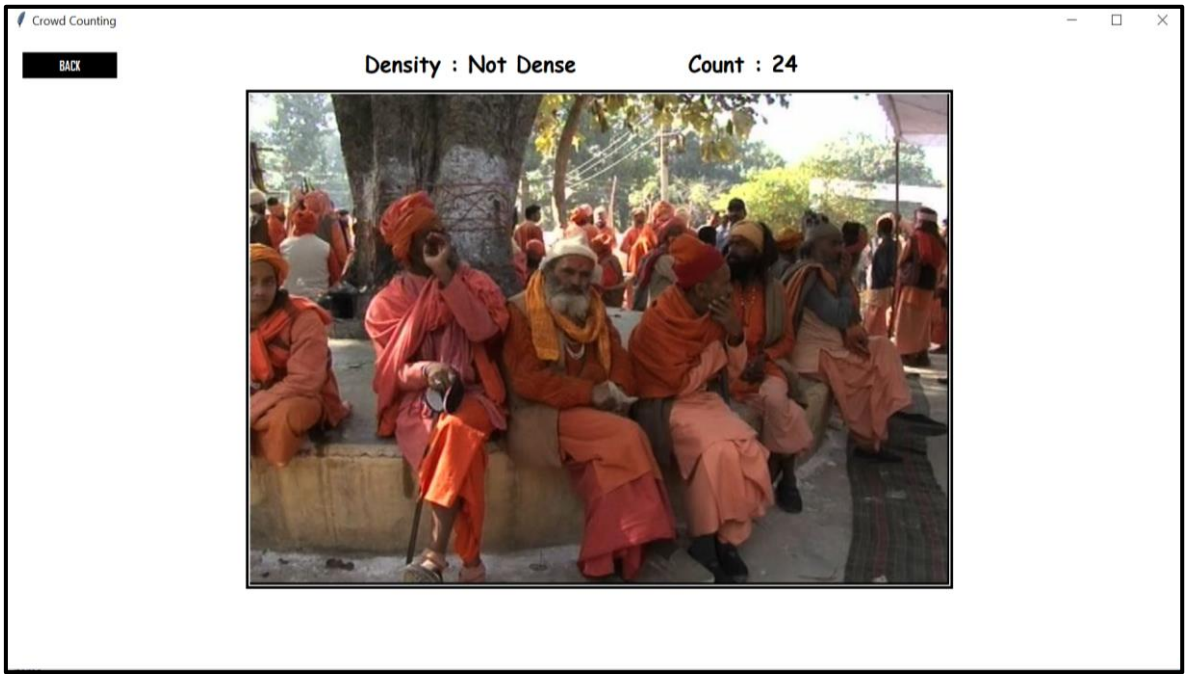

Figure 6.13 Example of image

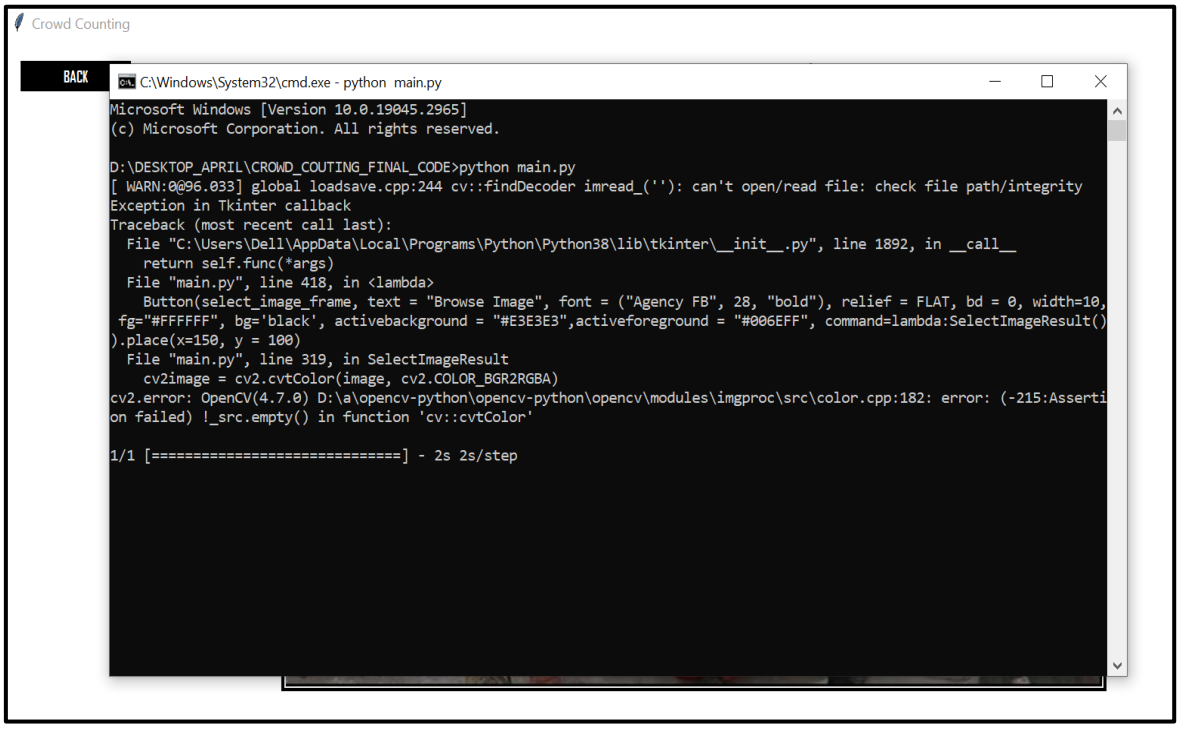

Figure 6.14 Background CNN based model ResNet

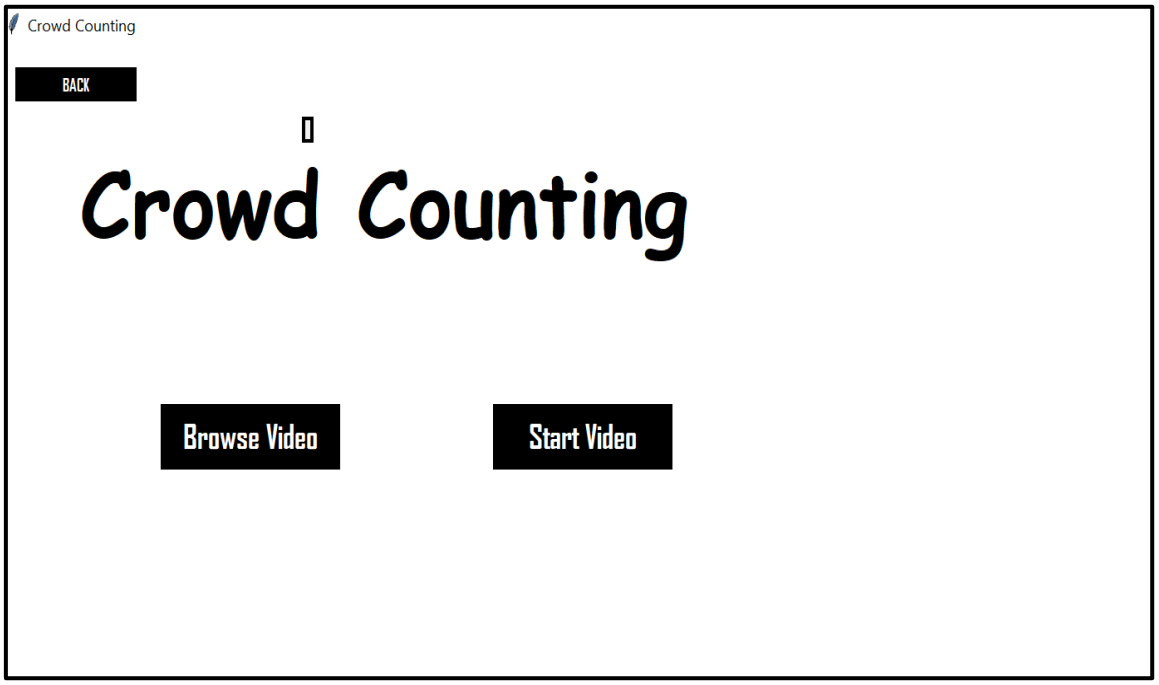

Figure 6.15 Browse Video and Start video

Figure 6.15 shows the Browse Video and Start video are the two options available in this project. These two modules work as per the video file input.

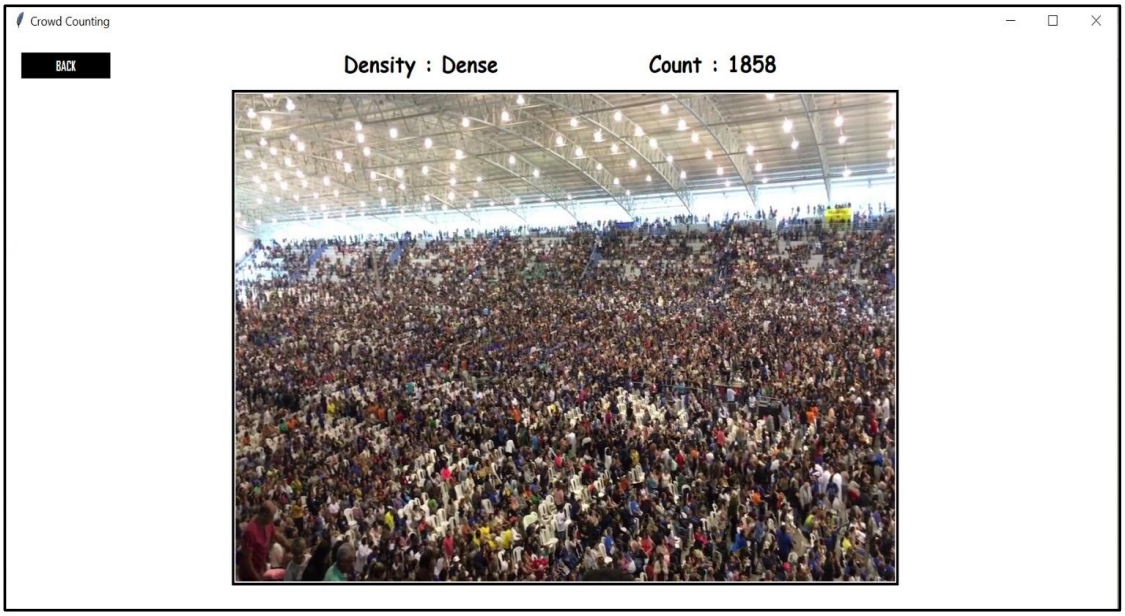

Figure 6.16 Density count

CHAPTER 7

SYSTEM TESTING

Testing is extremely important for quality assurance and ensuring the products reliability. The success of testing for programmer flaws in largely determined by the experience. Testing might be a crucial component in ensuring the proposed systems quality and efficiency in achieving its goal. Testing is carried out at various phases during the system design and implementation process with the goal of creating a system that is visible, adaptable and secure. Testing is an important element of the software development process. The testing procedure verifies whether the generated product meets the requirements for which it was intended.

7.1 Test objectives

• Testing may be a defined as a process of running a programme with the goal of detecting a flaw.

• An honest case is one in which there is a good chance of discovering a mistake that hasn't been detected yet.

• A successful test is one that uncovers previously unknown flaw. If testing is done correctly, problems in the programme will be discovered. Testing cannot reveal whether or not flaws are present. It can only reveal the presence of software flaws.

7.2 Testing principles

A programmer must first grasp the fundamental idea that governs software testing before applying the methodologies to create successful test cases. All testing must be able to be tracked back to the customer's specification.

7.3 Testing design

Any engineering product is frequently put to the test in one of two ways:

7.3.1 White Box Testing

Glass container checking out is every other call for this kind of checking out. By understanding the necessary characteristic that the product has been supposed to do,

checking out is regularly accomplished that proves every characteristic is absolutely operational at the same time as additionally checking for faults in every characteristic. The take a look at case layout technique that leverages the manage shape of the procedural layout to create take a look at instances is used on this take a look at case

7.3.2 Black Box Testing

Tests are regularly finished on this checking out via way of means of understanding the indoors operation of a product to make certain that each one gears mesh, that the indoors operation operates reliably in step with specification, and that each one inner additives had been nicely exercised. It is in most cases worried with the software's practical needs.

7.4 Testing Techniques

A software testing template should be established as a set of stages in which particular test suit design techniques are defined for the software engineering process.
The following characteristics should be included in every software testing strategy:

- Testing begins with the modules and extends to the mixing of the full computer based system.
- At different periods in time, different testing approaches are applicable.
- Testing is carried out by the software's developer and an independent test group. A software developer can use a software testing strategy as a route map. Testing might be a collection of actions that are prepared ahead of time and carried out in a methodical manner. As a result a software testing template should be established as a set of stages in which particular test suit design techniques are defined for the software engineering process. The following characteristics should be included in every software testing strategy: Testing begins at the module level and progresses to entire computer based system are mixing.
- At different periods in time different testing approaches are applicable.
- Testing is carried out by the software's developer and a separate test group.

7.5 Levels of Testing

Testing is frequently omitted at various stages of the SDLC. They are as follows:

7.5.1 Unit Testing

Unit testing checks the tiny piece of software that makes up the module. The white box orientation of the unit test is maintained throughout. Different modules are tested alongside the requirements created throughout the module design process. The aim of unit testing is to inspect the inner logic of the modules, and it is used to verify the code created during development phase. It is usually done by the module's developer. The coding phase is sometimes referred to as coding and unit testing because of its tight association with coding. Unit tests for many modules are frequently run in simultaneously.

7.5.2 Integration Testing

Integration testing is the second level of quality assurance. This type of testing integrates different components in program like modules also to check the interface problems. Many tested modules are combined into subsystems and tested as a result of this. The purpose of this test is to see if all of the modules are properly integrated. Integration testing may be divided into three categories:

• Top-Down Integration:

Top-Down integration is a method of gradually constructing a Programme structures. Modules are connected by working their way down the control Hierarchy, starting with the module having the most control. Bottom-Up Integration:

Construction and testing using autonomous modules begin with Bottom-up integration, as the name suggests.

• Regression Testing:

It is a subset of previously executed tests to ensure that Modifications have not propagated unexpected side effects during this competition of an Integration test strategy.

7.5.3 Functional Testing

The business and technical requirements, system documentation, and user guides all specify that functional tests must be conducted to ensure that the functions being tested are available. The following items are the focus of functional testing.

7.5.4 Validation Testing

Validation may be characterized in a lot of ways; however one easy definition is that validation is a hit whilst software program plays in a manner that clients may fairly expect. The affordable expectation is said with inside the software program requirement specification that is a record that lists all the software program's user-seen attributes. Validation standards are a segment of the specification. The statistics on this component serves as the premise for the validation trying out strategy.

7.5.5 Alpha Testing

Software developer can't know how a customer will utilize a programme ahead of time. Instructions to be utilized could be misconstrued, a peculiar combination of knowledge could be employed on a regular basis, and a result that was clear to the tester could be unclear to a field user. It's impractical to conduct a formal acceptance test with all users if the programmed is designed as a product that will be used by many people. Most software developers utilize alpha and beta testing to detect bugs that only the most experienced users seem to be aware of. At the developer's premises, a customer does the trial.

CONCLUSION

Crowd counting using deep learning models has emerged as a promising approach for accurately estimating the number of people in crowded scenes. Deep learning models, particularly convolutional neural networks (CNNs) and their variants have shown significant success in addressing the challenges of crowd counting, such as occlusion, scale variations, and complex crowd dynamics.The use of deep learning models for crowd counting typically involves two key stages: density map estimation and counting. In the density map estimation stage, the deep learning model learns to generate a density map that assigns higher values to regions with more people. This map acts as a continuous representation of the crowd density within an image. In the counting stage, the total number of people is obtained by summing the values of the density map.

This project has presented a CNN -based VGG16 & RESNET50 with Yolo 5 Pre-trained Model density estimation and crowd counting models. Here we make the crowd count by using images and videos.

One major challenge is the lack of labeled training data, as annotating large-scale crowd datasets is a time-consuming and labor-intensive process. Moreover, variations in viewpoint, lighting conditions, and camera perspectives pose additional difficulties for accurate crowd counting. Crowd counting using the ResNet50 deep learning model has proven to be effective and efficient in estimating the number of people in crowded scenes.

REFERENCES

[1] Babu Sam, D., Sajjan, N.N., Venkatesh Babu, R., Srinivasan, M., 2018. Divide and grow: Capturing huge diversity in crowd images with incrementally growing cnn, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3618–3626.

[2] Boominathan, L., Kruthiventi, S.S., Babu, R.V., 2016. Crowdnet: A deep convolutional network for dense crowd counting, in: Proceedings of the 24th ACM international conference on Multimedia, ACM. pp. 640–644.

[3] Chan, A.B., Liang, Z.S.J., Vasconcelos, N., 2008. Privacy preserving crowd monitoring: Counting people without people models or tracking, in: 2008 IEEE Conference on Computer Vision and Pattern Recognition, IEEE. pp. 1–7.

[4] Chan, A.B., Vasconcelos, N., 2009. Bayesian poisson regression for crowd counting, in: Computer Vision, 2009 IEEE 12th International Conference on, IEEE. pp. 545–551.

[5] Collobert, R., Weston, J., 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning, in: Proceedings of the 25th international conference on Machine learning, ACM. pp. 160–167.

[6] Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection, in: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, IEEE. pp. 886– 893.

[7] Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 315–323.

[8] He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034.

[9] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

[10] Hossain, M., Hosseinzadeh, M., Chanda, O., Wang, Y., 2019. Crowd counting using scale-aware attention networks, in: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE. pp. 1280–1288.