

**A Project Report
on
People Count from Surveillance Video**

**submitted in partial fulfillment of the requirements for the award of the degree
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING**

by

17WH1A0565

Ms. G. NIKHITHA

17WH1A0575

Ms. CH. MANASA

17WH1A05C0

Ms. K. SRAVANI

under the esteemed guidance of

**Dr. L. LAKSHMI
Professor**



**Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090**

May 2021

DECLARATION

We hereby declare that the work presented in this project entitled **“People Count from Surveillance Video”** submitted towards completion of Project Work in IV year of B.Tech., CSE at ‘BVRIT HYDERABAD College of Engineering For Women’, Hyderabad is an authentic record of our original work carried out under the guidance of Dr. L. Lakshmi, Professor, Department of CSE.

Sign. with date:

Ms. G. NIKHITHA

(17WH1A0565)

Sign. with date:

Ms. CH. MANASA

(17WH1A0575)

Sign. with date:

Ms. K. SRAVANI

(17WH1A05C0)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering



Certificate

This is to certify that the Project Work report on **“PEOPLE COUNT FROM SURVEILLANCE VIDEO”** is a bonafide work carried out by Ms. G. NIKHITHA (17WH1A0565) ; Ms. CH. MANASA (17WH1A0575) ; Ms. K. SRAVANI (17WH1A05C0) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. K. Srinivasa Reddy
Professor and HoD,
Department of CSE

Guide
Dr. L. LAKSHMI
Professor

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. k. Srinivasa Reddy, Professor and HoD, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Dr. L. Lakhmi, Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for her constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE Department** who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Ms. G. NIKHITHA
(17WH1A0565)

Ms. CH. MANASA
(17WH1A0575)

Ms. K. SRAVANI
(17WH1A05C0)

Contents

S.No.	Topic	Page No.
	Abstract	i
	List of Figures	ii
1.	Introduction	1
	1.1 Objective	1
	1.2 Methodology	2
	1.2.1 Dataset	1
	1.2.2 The proposed CNN model	4
2.	Theoretical Analysis of the proposed project	8
	2.1 Requirements Gathering	8
	2.1.1 Software Requirements	8
	2.1.2 Hardware Requirements	8
	2.2 Technologies Description	8
3.	Design	12
	3.1 Introduction	12
	3.2 Architecture Diagram	12
	3.3 UML Diagrams	13
	3.3.1 Use Case Diagram	13
	3.3.2 Sequence Diagram	14
	3.3.3 Class Diagram	14
	3.4 Dataset of Images	15
	3.5 Division of dataset for Training and Testing	15
	3.6 Data Preprocessing	15
	3.7 Building the Model	15
4.	Implementation	17
	4.1 Coding	17
	4.2 Results	26
	4.2.1 Model Summary	26
	4.2.2 Dataset Training and Testing	26
	4.2.3 Accuracy of the model	27

	4.2.4 Output Screenshots	28
	4.2.5 Manually Testing for different images	28
5.	Conclusion and Future Scope	30
6.	References	31

ABSTRACT

People Counting is a technique used to count the number of people in a picture. People counting is not an easy task if it is done manually by our hand because we can lost count in the middle of doing this laborious task, especially when dealing with object that intersects with each other or dense crowd. This project automates the counting process by building a machine learning system that can convert a video into frames, then the model will output number of objects in a particular frame. We built the model using **Convolutional Neural Network (CNN)** technique. The system that we built is capable of counting pedestrians in a mall. The frames/images are generated from CCTV that is placed somewhere in the mall. From those frames/images, the system will output how many pedestrians at that particular place in the mall. **VGG16** is used to extract the features of the image and **Structural Similarity Index (SSIM)** to measure the similarity between two images. Then, use the similarity measure as a loss function named Local Pattern Consistency loss + the loss from Euclidean loss. The experimental results show the predicted number of people and exact number of people in the image and the accuracy of the model is 85% to 90%.

LIST OF FIGURES

S.No.	Fig No.	Fig Name	Page No.
1.	1.2.1	Frames from Video Dataset	2
2.	1.2.1.1	Loading Dataset into Colab	3
3.	1.2.1.2	Dataset	4
4.	1.2.2	VGG16	5
5.	1.2.2.1	VGG16 Architecture	5
6.	3.2	Architecture Diagram	12
7.	3.3.1	Use Case Diagram	13
8.	3.3.2	Sequence Diagram	14
9.	3.3.3	Class Diagram	14
10.	4.2.1	Model summary	26
11.	4.2.2	Training the dataset	26
12.	4.2.2.1	Testing and Plotting people	27
13.	4.2.3	Accuracy	27
14.	4.2.4	Output	28
15.	4.2.5	Manually choosing image from device	28
16.	4.2.5.1	Selecting any image for prediction	29
17.	4.2.5.2	Output for the selected image	29

1. INTRODUCTION

People Counting is a technique used to count the number of people in a picture. Counting how many objects in an image is essential for various industries and researchers. Some of the use-cases are:

- Monitoring high traffic roads, or public places.
- Preventing people from entering the forbidden, or dangerous places.
- Giving information about the favorite spots where a high number of people gather.

This shows the usefulness of crowd counting in real life. People counting is not an easy task if it is done manually by our hand because we can lost count in the middle of doing this laborious task, especially when dealing with object that intersects with each other or dense crowd. To solve this problem, we can automate the counting process by building a machine learning system, that can receive an image that contains objects that we want to count as an input and then the model will output number of objects in that image. The system that we build is capable of counting pedestrians in a mall. The images are generated from CCTV that is placed somewhere in the mall. From those images, the system will tell us how many pedestrians at that particular place in the mall. We built the model using **Convolutional Neural Network (CNN)** technique.

1.1 Objective

People counting is not an easy task. The objective of the project is to count the total number of people per frame which is obtained from a video i.e. CCTV footage. This will help us to control the People entering into heavy traffic areas and Monitoring high traffic roads, or Public places. There are so many others use-cases of crowd counting that are not mentioned here. This shows the usefulness of crowd counting in real life. The model takes a video and converts into frames/images. The images are trained using a machine learning model and predict the number of people in each frame/image.

1.2 Methodology

To predict the people count from surveillance video the video from CCTV placed anywhere in public place is required, from that video different number of frames/images are extracted and that frames/images are used as dataset. In this section the methodology followed is discussed in detail.

1.2.1 Dataset

The dataset that is used in the model is images that are generated from a single CCTV that is placed somewhere in a mall of the same spot, which contains pedestrians who walks around the CCTV. Each image has different number of persons. The images are generated from video with given time rate i.e. frame rate. The video is of 100 seconds and frame rate is 0.1 second i.e. 1000 images/frames are generated in the .jpg format from the video. The 1000 images extracted from video is used as dataset.

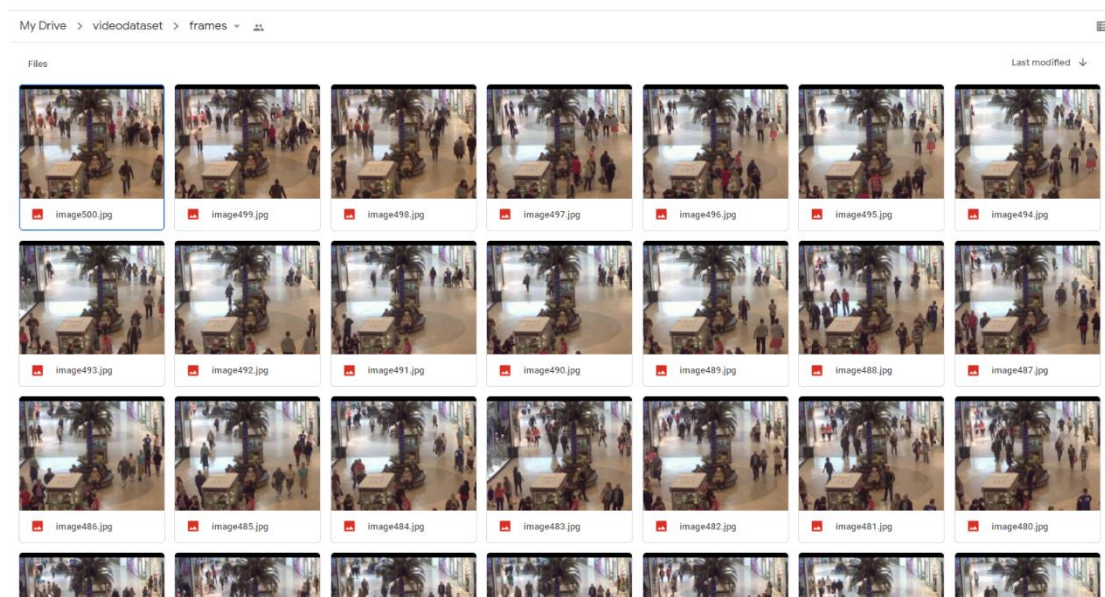


Fig 1.2.1: Frames Generated from CCTV Video

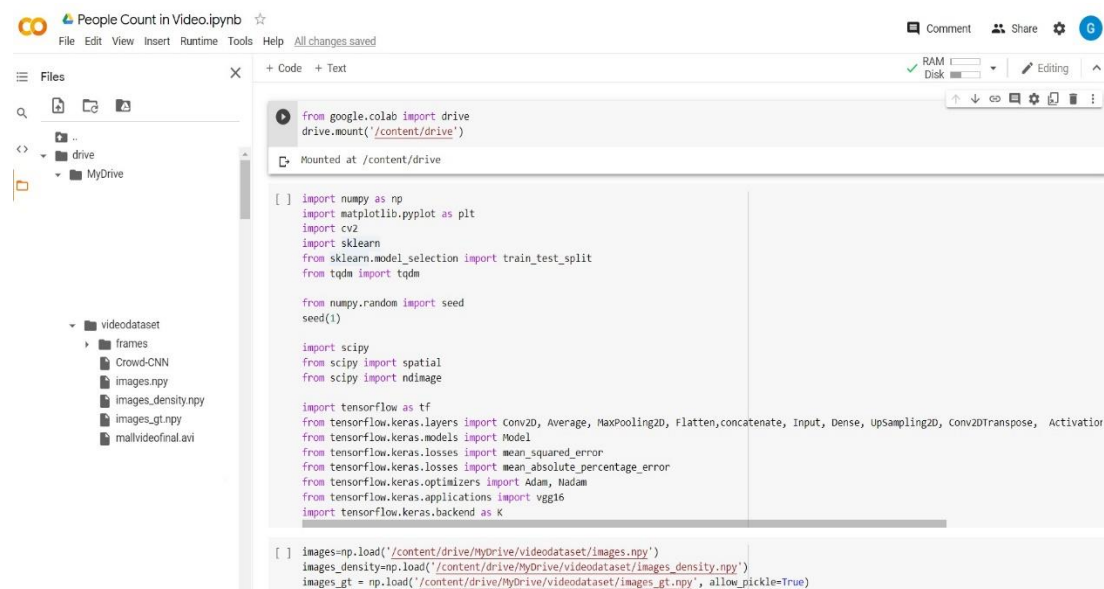
Using extracted Images to predict People count

The dataset now consists of 1000 images in .jpg format. The dataset is converted in to NumPy array. The array holds and represents any image data in a structured way i.e. Images are converted into NumPy Array in Height, Width, Channel format. Ground truth which means a set of measurements that is known to be much more accurate is used in the model and The density maps was generated with Geometry-adaptive kernel.

We generated it from images extracted from video. The images dataset are normalized into the interval $[0,1]$. The dataset is divided into training and validation set with 85% : 15% ratio \ X (input) is images and Y (label) is its corresponding density map. VGG16 is used to train the images in the model. The training process is done with 2 epochs without train the VGG16 layers and 2 epoch with trainable VGG16 layers.

Loading the dataset, ground truth, density map

To built the model first it is required to load the dataset. The density map generated is named as images density which is a npy file. Ground truth of the images is named as images gt and the images which are in jpg format converted to npy files is named as images npy file. It is required to load these files into Google Collaboratory by mounting the drive.



```

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[ ] import numpy as np
import matplotlib.pyplot as plt
import cv2
import sklearn
from sklearn.model_selection import train_test_split
from tqdm import tqdm

from numpy.random import seed
seed(1)

import scipy
from scipy import spatial
from scipy import ndimage

import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Average, MaxPooling2D, Flatten, concatenate, Input, Dense, Upsampling2D, conv2DTranspose, Activation
from tensorflow.keras.models import Model
from tensorflow.keras.losses import mean_squared_error
from tensorflow.keras.losses import mean_absolute_percentage_error
from tensorflow.keras.optimizers import Adam, Nadam
from tensorflow.keras.applications import vgg16
import tensorflow.keras.backend as K

[ ] images=np.load('/content/drive/mydrive/videodataset/images.npy')
images_density=np.load('/content/drive/mydrive/videodataset/images_density.npy')
images_gt = np.load('/content/drive/mydrive/videodataset/images_gt.npy', allow_pickle=True)

```

Fig 1.2.1.1: Loading Dataset into Collab

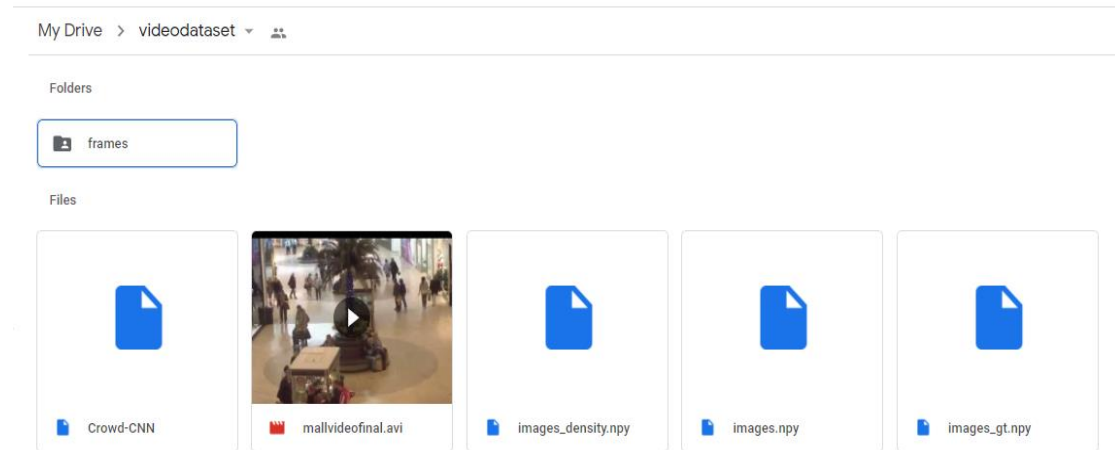


Fig 1.2.1.2: Dataset

1.2.2 The proposed CNN model.

VGG16 - Convolutional Network for Classification and detection

VGG16 is a convolutional neural network model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. Three convolution blocks with ReLU activation is created and BatchNormalization layer inside this block to reduce internal covariate shift of the model is added. This will cause the model to train more faster. AveragePooling layer to reduce the dimension of the feature map is also added. Reducing the dimension of the feature map will decrease the number of parameters in the model, so the model will be much more easy to train. Then, Dropout layer to prevent overfitting is added. At the end of the model, GlobalMaxPooling to reduce the depth of the feature map is added and connected it to final activation ReLU layer. MeanSquaredError loss for this task and Adam method to perform the optimization is used. Also MeanAbsoluteError for the metrics is used.

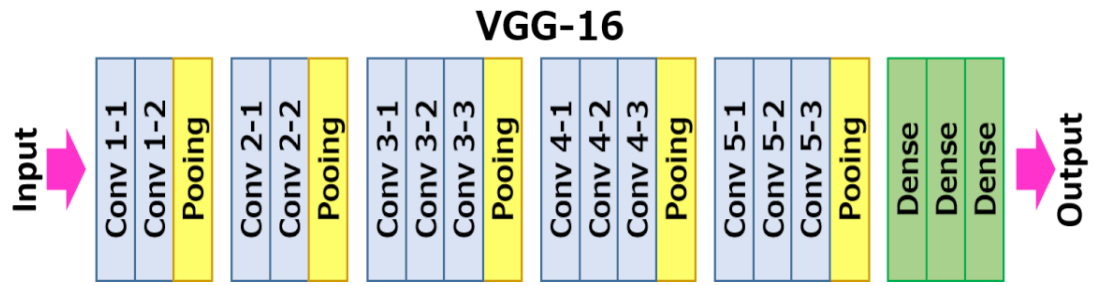


Fig 1.2.2: VGG16

VGG16 – Architecture

The following image describes our model. Where 'CONV a' denotes Convolution with kernel size $a \times a$ and 'CONV T' means Transposed Convolutional layer.

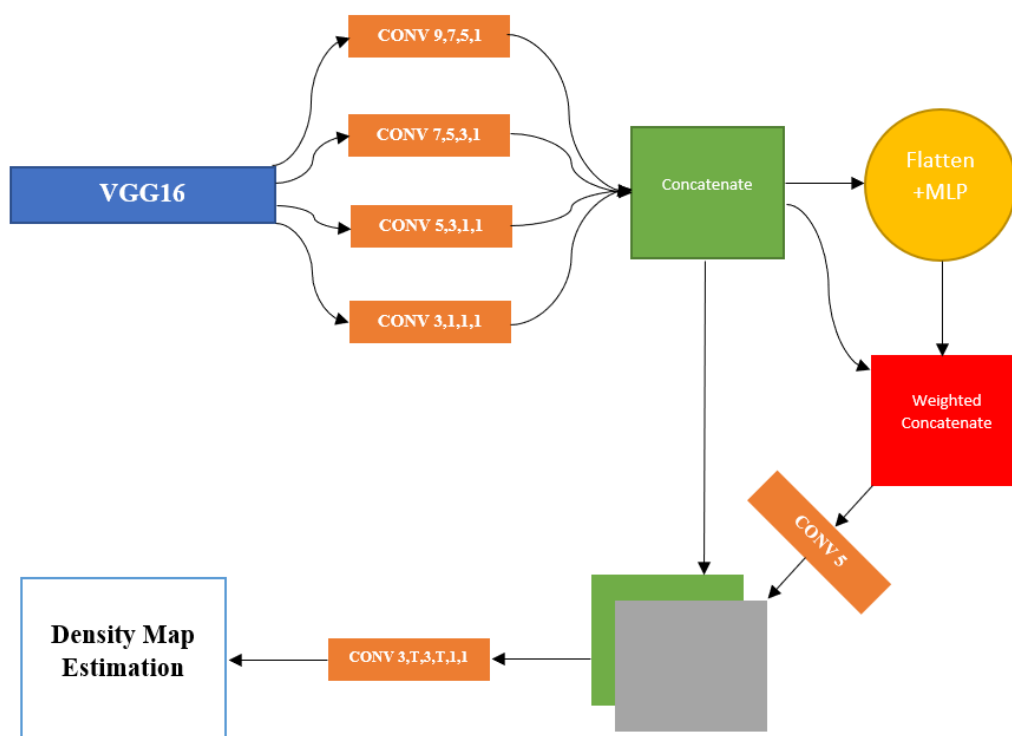


Fig 1.2.2.1: Network Architecture

Suppose we have an image with a size of 96 x 128.

- First, the first ten layers of VGG16 is used to extract the features of the image. After features with the first ten layers of VGG16 are extracted, we got an output with size 24 x 32 which is a quarter of the original size.
- Second, we fed our output from VGG16 to four filters with different sizes. We didn't use the pooling layer in the set of convolutional layers, because using many pooling layers will affect the loss of spatial information from the feature map.
- Third, we used feature enhancement layer, where we concatenate our output (x_{conct} , contain 4 filter from the previous layers) and used flatten+MLP with softmax function in the output layer to get the weight for each input filters. Thus, the model will learn to give a high weight for the filter which best represents the image.
- Last, we need to up sampling the image with size 24 x 32 to 96 x 128. Transposed Convolutional layer for the up sampling method is used. Concatenated the x_{conct} with the filter that generated by convolution the weighted x_{conct} is also used. In the last layer, we set the filter size with 1, and that filter represents the predicted density map.

For each convolutional layer, we used batch-normalization and ReLu as the activation function.

Structural Similarity Index (SSIM) and Local Pattern Consistency loss

- The Structural Similarity Index is used to measure the similarity between 2 images. Then, use the similarity measure as a loss function named Local Pattern Consistency loss + the loss from Euclidean loss.
- For each predicted density map and actual density map, the similarity between a small patch of the image with a gaussian kernel of 12x16 as the weight is counted

- The model loss function is defined as follow: Euclidean Loss (MSE) + α * Local Pattern Consistency Loss.
- Local pattern consistency loss helps model to learn similarity between small patches of the image.

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language: Python 3.7

Dataset : Surveillance video from mall

Packages : TensorFlow, Keras, NumPy, Matplotlib, Scikit-learn

Tool : Google Collab

2.1.2 Hardware Requirements

Operating System: Windows 10

Processor : Intel Core i3

CPU Speed : 2.30 GHz

Memory : 4 GB (RAM)

2.2 Technologies Description

Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive – you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does

say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras contains numerous implementations of commonly used neural-network building blocks such as

layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

In addition to standard neural networks, It supports convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, Google colab, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the [sample plots](#) and [thumbnail gallery](#).

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc., via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console

- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis
- Extensions or modules for SciPy are conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

Google Colaboratory

Google Colaboratory or ‘Colab’ for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPU.

Colab allows to write and execute Python in the browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Colab can make work easier. Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them.

Colab can harness the full power of popular Python libraries to analyze and visualize data. Colab can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine.

3. DESIGN

3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

3.2 Architecture Diagram

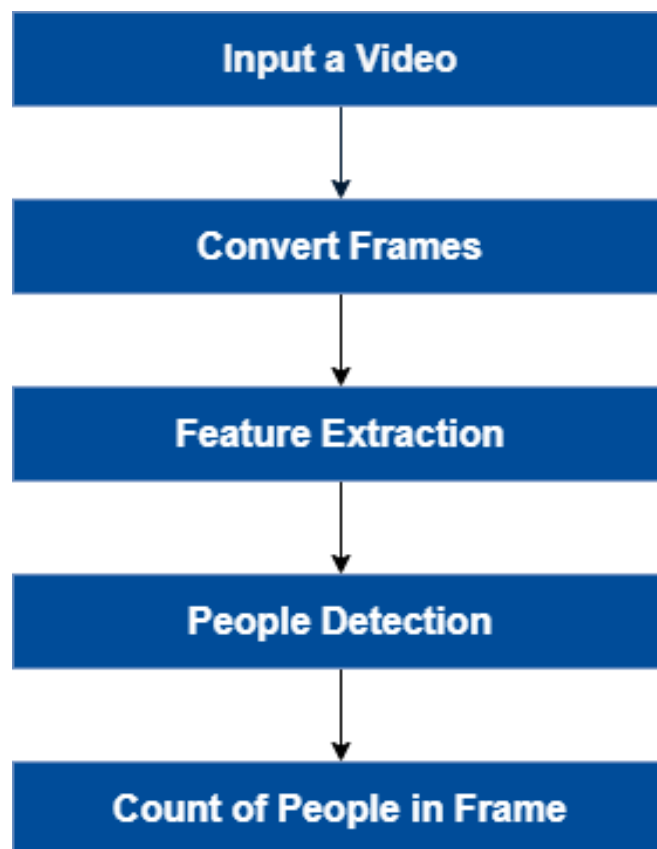


Fig 3.2: Architecture Diagram

3.3 UML Diagrams

A UML diagram shows the unified visual presentation of the UML (Unified Modeling Language) system intending to let developers or business owners understand, analyze, and undertake the structure and behaviors of their system.

In UML, there are some diagrams available, in that

- Use case diagram
- Sequence diagram
- Class diagram are few types of UML diagrams

3.3.1 Use Case Diagram

Use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

To model the entire system, a number of use case diagrams are used.

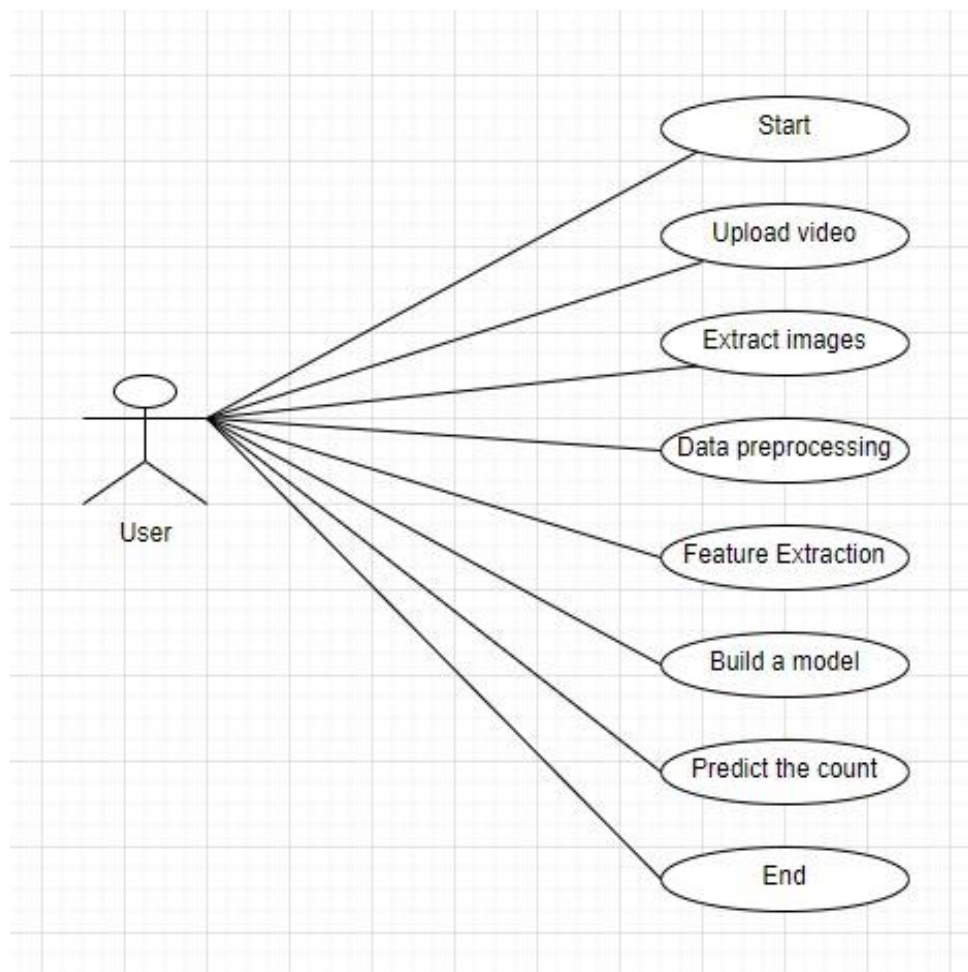


Fig 3.3.1: Use Case Diagram

3.3.2 Sequence Diagram

A sequence diagram generally shows the interaction between objects in a sequential order. It is for users to document and understand requirements in a new system. In software development, this type of diagram is used to represent the architecture of a system.

The Sequence Diagram of the model is as follows.

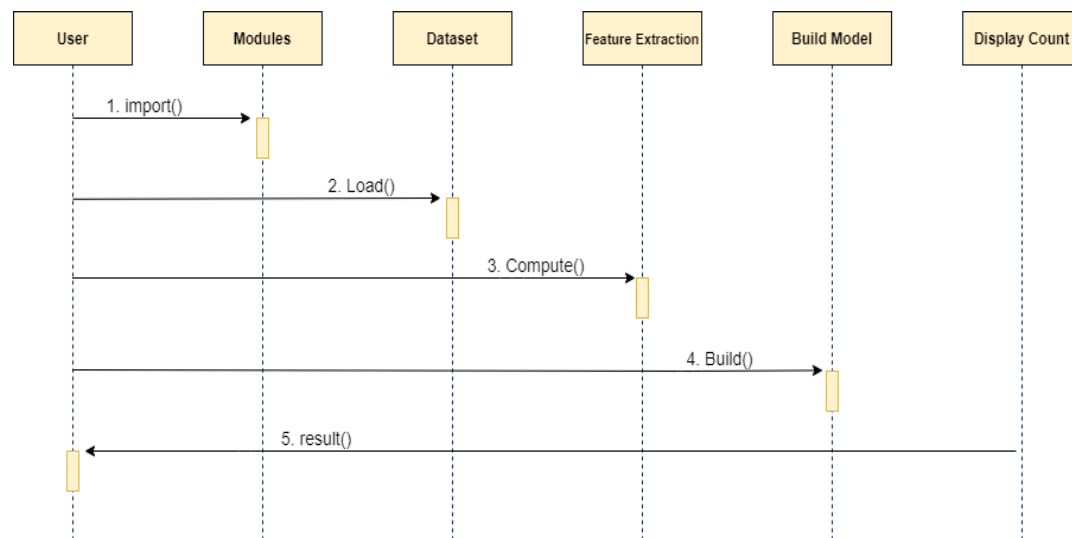


Fig 3.3.2: Sequence Diagram

3.3.3 Class Diagram

It is the most widely used UML diagram sub-category. The Class diagram is the building block of all object-oriented software systems. Users can depict the static structure and identify classes relationship of a system by checking system's classes and attributes. Each class has three basic elements: the class name at the top, the class attributes in the middle, and the class behaviors at the bottom.

Below is the class diagram of the model.

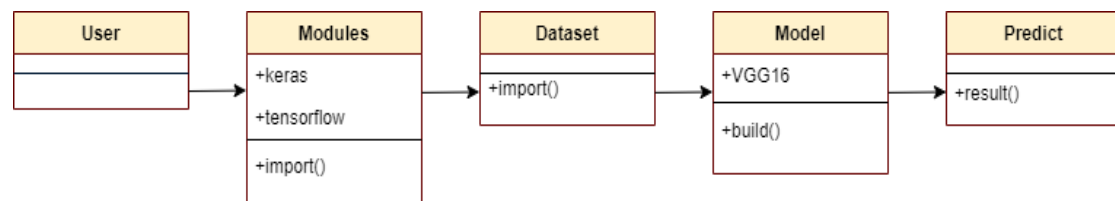


Fig 3.3.3: Class Diagram

3.4 Dataset of Images

- To train the model the images.npy file, images_gt.npy file, images_density.npy file which are mentioned clearly in 1.2.1 Dataset are used.
- 85% images are used for training the model and 15% images are used for testing the model.

3.5 Division of dataset for training and testing

- The dataset is divided into five training batches and one test batch. The test batch contains randomly-selected images from each class.
- The dataset is divided into training and validation set with 85% : 15% ratio \ X (input) is images and Y (label) is its corresponding density map.
- The training process is done with 2 epochs without train the VGG16 layers and 2 epoch with trainable VGG16 layers.

3.6 Data Preprocessing

- Normalize images value
 - To normalize the images value divide the images by 255 so it will have value with range [0,1].
- Create Train and Validation Set
 - The dataset is divided into training and validation with 85%:15% ratio \ X (input) is images and Y (label) is its corresponding density map.
- Add dimension for label
- Implement a function to generate patches of images

3.7 Building the Model

After preprocessing the data model is built in the following procedure

Load the imports

Import all the required libraries to build the model.

Load Dataset

In this part, we load the dataset, ground truth, density map.

Google Drive Mounting

Mount the drive to the colab where the model is to be implemented

Load dataset from file

Dataset is loaded from file by giving the path of the files in the colab.

Import VGG16 as Baseline

Set the input shape for the model\ Creating baseline model with VGG16

Implement the Model

Implementation is actual building of the model. Implementation is briefed in 4.1 coding part.

Evaluation and Result

Evaluating the model implemented and the experimental result is shown as predicted number of people and actual number of people

4. IMPLEMENTATION

4.1 Coding

```

from google.colab import drive
drive.mount('/content/drive')

pip install opencv-python

import cv2
import os.path
path = '/content/drive/MyDrive/videodataset/frames/'
vidcap = cv2.VideoCapture('/content/drive/MyDrive/videodataset/mallvideofinal.avi')
def getFrame(sec):
    vidcap.set(cv2.CAP_PROP_POS_MSEC,sec*1000)
    hasFrames,image = vidcap.read()
    if hasFrames:
        cv2.imwrite(path + 'image'+str(count)+'.jpg' , image)
    return hasFrames
sec = 0
frameRate = 0.2
count=1
success = getFrame(sec)
while success:
    count = count + 1
    sec = sec + frameRate
    sec = round(sec, 2)
    success = getFrame(sec)

%tensorflow_version 1.x

!pip install keras==2.1.5

pip install -U scikit-learn

```

Load the imports

```

import numpy as np
import matplotlib.pyplot as plt
import cv2
import sklearn
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from numpy.random import seed
seed(1)
import scipy
from scipy import spatial
from scipy import ndimage
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Average, MaxPooling2D,
Flatten, concatenate, Input, Dense, UpSampling2D, Conv2DTranspose, Activation,
BatchNormalization, Lambda
from tensorflow.keras.models import Model
from tensorflow.keras.losses import mean_squared_error
from tensorflow.keras.losses import mean_absolute_percentage_error
from tensorflow.keras.optimizers import Adam, Nadam
from tensorflow.keras.applications import vgg16
import tensorflow.keras.backend as K

```

Load dataset from file

```

images=np.load('/content/drive/MyDrive/videodataset/images.npy')
images_density=np.load('/content/drive/MyDrive/videodataset/images_density.npy')
images_gt      =      np.load('/content/drive/MyDrive/videodataset/images_gt.npy',
allow_pickle=True)
img=plt.imshow(images[200])
plt.imshow(images[200])
plt.imshow(images_density[200],cmap='prism',interpolation='bicubic',alpha=0.25)
print('number of people: %4.2f' %np.sum(images_density[200]/1000))
size=(96,128)

```

```
input_shape=(size[0],size[1],3)
baseline_vgg=vgg16.VGG16(weights='imagenet',include_top=False,
input_shape=input_shape)
```

Main Model

```
def Crowd_CNN(input_shape=input_shape):
    input_flow=Input(input_shape)
    base_filter=8
    x_vgg=baseline_vgg.get_layer(index=1)(input_flow)
    for i in range(2,10):
        x_vgg=baseline_vgg.get_layer(index=i)(x_vgg)
    x_1=Conv2D(base_filter*16,9,padding='same',activation='relu')(x_vgg)
    x_1=BatchNormalization()(x_1)
    x_1=Conv2D(base_filter*8,7,padding='same',activation='relu')(x_1)
    x_1=BatchNormalization()(x_1)
    x_1=Conv2D(base_filter*4,5,padding='same',activation='relu')(x_1)
    x_1=BatchNormalization()(x_1)
    x_1=Conv2D(1,1,padding='same',activation='relu')(x_1)
    x_1=BatchNormalization()(x_1)
    x_2=Conv2D(base_filter*8,7,padding='same',activation='relu')(x_vgg)
    x_2=BatchNormalization()(x_2)
    x_2=Conv2D(base_filter*4,5,padding='same',activation='relu')(x_2)
    x_2=BatchNormalization()(x_2)
    x_2=Conv2D(base_filter*2,3,padding='same',activation='relu')(x_2)
    x_2=BatchNormalization()(x_2)
    x_2=Conv2D(1,1,padding='same',activation='relu')(x_2)
    x_2=BatchNormalization()(x_2)
    x_3=Conv2D(base_filter*4,5,padding='same',activation='relu')(x_vgg)
    x_3=BatchNormalization()(x_3)
    x_3=Conv2D(base_filter*2,3,padding='same',activation='relu')(x_3)
    x_3=BatchNormalization()(x_3)
    x_3=Conv2D(base_filter*1,1,padding='same',activation='relu')(x_3)
    x_3=BatchNormalization()(x_3)
```

```

x_3=Conv2D(1,1,padding='same',activation='relu')(x_3)
x_3=BatchNormalization()(x_3)
x_4=Conv2D(base_filter*4,3,padding='same',activation='relu')(x_vgg)
x_4=BatchNormalization()(x_4)
x_4=Conv2D(base_filter*2,1,padding='same',activation='relu')(x_4)
x_4=BatchNormalization()(x_4)
x_4=Conv2D(base_filter*1,1,padding='same',activation='relu')(x_4)
x_4=BatchNormalization()(x_4)
x_4=Conv2D(1,1,padding='same',activation='relu')(x_4)
x_4=BatchNormalization()(x_4)
x_conct=concatenate([x_1,x_2,x_3,x_4])
x_fel=Flatten()(x_conct)
x_fel=Dense(base_filter*8,activation='relu')(x_fel)
x_fel=Dense(4,activation='softmax')(x_fel)
x=Lambda(lambda x: x*(1+x_fel))(x_conct)
x=Conv2D(base_filter*2,5,padding='same',activation='relu')(x)
x=BatchNormalization()(x)
x=concatenate([x,x_conct])
x=Conv2D(base_filter*2,3,padding='same',activation='relu')(x)
x=BatchNormalization()(x)
x=Conv2DTranspose(base_filter*2, kernel_size=(2, 2), strides=(2, 2))(x)
x=BatchNormalization()(x)
x=Conv2D(base_filter*2,3,padding='same',activation='relu')(x)
x=BatchNormalization()(x)
x=Conv2DTranspose(base_filter, kernel_size=(2, 2), strides=(2, 2))(x)
x=BatchNormalization()(x)
x=Conv2D(base_filter,1,padding='same',activation='relu')(x)
x=BatchNormalization()(x)
x=Conv2D(1,1,padding='same',activation='relu')(x)
model=Model(inputs=input_flow,outputs=x)
return model

```

SSIM and Euclidean Loss

```

def ssim_loss(y_true, y_pred, c1=0.01**2, c2=0.03**2):
    weights_initial = np.multiply(
        cv2.getGaussianKernel(12, 1.5),
        cv2.getGaussianKernel(16, 1.5).T
    )
    weights_initial = weights_initial.reshape(*weights_initial.shape, 1, 1)
    weights_initial = K.cast(weights_initial, tf.float32)
    mu_F = tf.nn.conv2d(y_pred, weights_initial, [1, 1, 1, 1], padding='SAME')
    mu_Y = tf.nn.conv2d(y_true, weights_initial, [1, 1, 1, 1], padding='SAME')
    mu_F_mu_Y = tf.multiply(mu_F, mu_Y)
    mu_F_squared = tf.multiply(mu_F, mu_F)
    mu_Y_squared = tf.multiply(mu_Y, mu_Y)
    sigma_F_squared = tf.nn.conv2d(tf.multiply(y_pred, y_pred), weights_initial, [1, 1, 1, 1], padding='SAME') - mu_F_squared
    sigma_Y_squared = tf.nn.conv2d(tf.multiply(y_true, y_true), weights_initial, [1, 1, 1, 1], padding='SAME') - mu_Y_squared
    sigma_F_Y = tf.nn.conv2d(tf.multiply(y_pred, y_true), weights_initial, [1, 1, 1, 1], padding='SAME') - mu_F_mu_Y
    ssim = ((2 * mu_F_mu_Y + c1) * (2 * sigma_F_Y + c2)) / ((mu_F_squared + mu_Y_squared + c1) * (sigma_F_squared + sigma_Y_squared + c2))
    return 1 - tf.reduce_mean(ssim, axis=[1, 2, 3])

def ssim_eucli_loss(y_true, y_pred, alpha=0.0025):
    ssim = ssim_loss(y_true, y_pred)
    eucli = mean_squared_error(y_true, y_pred)
    loss = eucli + alpha * ssim
    return loss

def mae_cnt(labels, preds):
    cnt_label = K.sum(labels)
    cnt_pred = K.sum(preds)
    return K.abs(cnt_label - cnt_pred)

```

Define the Model

```
lr=1e-5
model=Crowd_CNN()
model.load_weights('/content/Crowd-CNN.h5')
Optimizer=Nadam(lr)
model.summary()
model.compile(optimizer=Optimizer,loss=ssim_eucli_loss,metrics=[mae_cnt])
```

Training and Testing validation

```
X_train,.X_val,.Y_train,
Y_val=train_test_split(images,images_density,test_size=0.15,random_state=70)
```

```
Y_train=np.expand_dims(Y_train, -1)
Y_val=np.expand_dims(Y_val,-1)
```

```
def get_patch(X,Y):
    x_patch=np.zeros([3*X.shape[0],size[0],size[0],size[1],3])
    y_patch=np.zeros([3*y.shape[0],size[0],size[0],size[1],1])
```

```
for i in range(X.shape[0]):
    for j in range(3):
        x1=np.random.randint(0,X.shape[1]-size[0]-1)
        y1=np.random.randint(0,X.shape[2]-size[1]-1)
        x2=size[0]+x1
        y2=size[1]+y1
        x_patch[3*i+j]=X[i,x1:x2,y1:y2,:]
        y_patch[3*i+j]=Y[i,x1:x2,y1:y2,:]
    return x_patch,y_patch
```

```
for i in model.layers[:5]:
    i.trainable=False
model.compile(optimizer=Optimizer,loss=ssim_eucli_loss,metrics=[mae_cnt])
```

```

for i in range(1):
    x_patch,y_patch=get_patch(X_train, Y_train)
    x_patch_val,y_patch_val=get_patch(X_val, Y_val)

model.fit(x_patch,y_patch,batch_size=1,epochs=1,validation_data=(x_patch_val,y_patch_val))
from sklearn import model_selection
X_train,X_test,Y_train,Y_test =
model_selection.train_test_split(images,images_density,test_size=0.15,random_state=
7)
model = Crowd_CNN()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result*100.0))

```

Testing

```

test_size = 2
idx_test = np.random.randint(images.shape[0], size=test_size)
idx_test
X = images[idx_test]/255
y = images_density[idx_test]
plt.title("Image for data 0")
plt.imshow(X[0])
plt.imshow(y[0],cmap='prism',interpolation='bicubic',alpha=0.25)
print('number of people: %4.2f' %np.sum(y[0]/1000))
plt.imshow(X[1])
plt.imshow(y[1],cmap='prism',interpolation='bicubic',alpha=0.25)
print('number of people: %4.2f' %np.sum(y[1]/1000))
y = np.expand_dims(y,-1)
y.shape

```

```

def full_eval(x,smooth=False):
    y=np.zeros([480,640])

```

```

for i in range(0,480,size[0]):
    for j in range(0,640,size[1]):
        y[i:i+size[0],j:j+size[1]]+=model.predict(x[:,i:i+size[0],j:j+size[1],:])[0,:,:,0]
    if smooth:
        y=ndimage.filters.gaussian_filter(y, 2, mode='constant')
    return y
for i in range(0, test_size):
    y_predict = full_eval(X[i:i+1],smooth=True)
    print("Prediction for data { }".format(i))
    print('Predicted Number of people: %4.2f' %(np.sum(y_predict)/1000))
    print('Exact Number of people: %4.2f' %(np.sum(y[i])/1000))
    print()

e=0
y_predict=full_eval(X[e:e+1],smooth=True)
plt.figure()
f, axarr = plt.subplots(1,2,figsize=(20, 20))
axarr[0].title.set_text('predicted density map')
axarr[0].imshow(X[e,:,:,])
axarr[0].imshow(y_predict, alpha=0.7)
axarr[1].title.set_text('actual density map')
axarr[1].imshow(X[e,:,:,])
axarr[1].imshow(y[e,:,:,0])
print('Predicted Number of people: %4.2f' %(np.sum(y_predict)/1000))
print('Exact Number of people: %4.2f' %(np.sum(y[e])/1000))

e=1
y_predict=full_eval(X[e:e+1],smooth=True)
plt.figure()
f, axarr = plt.subplots(1,2,figsize=(20, 20))
axarr[0].title.set_text('predicted density map')
# axarr[0].imshow(X[e,:,:,])
axarr[0].imshow(y_predict)

```



```

axarr[1].title.set_text('actual density map')
# axarr[1].imshow(X[e,:,:,:])
axarr[1].imshow(y[e,:,:,:])
print('Predicted Number of people: %4.2f' %(np.sum(y_predict)/1000))
print('Exact Number of people: %4.2f' %(np.sum(y[e])/1000))

```

Manually Testing for different images

```

from google.colab import files
from tensorflow.keras.preprocessing import image
uploaded = files.upload()
for fn in uploaded.keys():
    path = '/content/' + fn
    img = image.load_img(path, target_size=(480, 640))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    images = np.vstack([x])
    images[0] = images[0] / 255
    y_predict = full_eval(images,smooth=True)
    print("\nPrediction for data {}".format(fn))
    print('Predicted Number of people: %4.2f\n' %(np.sum(y_predict)/1000))

```

4.2 Results

4.2.1 Model Summary

People Count in Video.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
model.summary()
```

batch_normalization_50 (BatchNorm)	(None, 24, 32, 16)	64	conv2d_48[0][0]
batch_normalization_54 (BatchNorm)	(None, 24, 32, 8)	32	conv2d_52[0][0]
batch_normalization_58 (BatchNorm)	(None, 24, 32, 8)	32	conv2d_56[0][0]
conv2d_45 (Conv2D)	(None, 24, 32, 1)	33	batch_normalization_46[0][0]
conv2d_49 (Conv2D)	(None, 24, 32, 1)	17	batch_normalization_50[0][0]
conv2d_53 (Conv2D)	(None, 24, 32, 1)	9	batch_normalization_54[0][0]
conv2d_57 (Conv2D)	(None, 24, 32, 1)	9	batch_normalization_58[0][0]
batch_normalization_47 (BatchNorm)	(None, 24, 32, 1)	4	conv2d_45[0][0]
batch_normalization_51 (BatchNorm)	(None, 24, 32, 1)	4	conv2d_49[0][0]
batch_normalization_55 (BatchNorm)	(None, 24, 32, 1)	4	conv2d_53[0][0]
batch_normalization_59 (BatchNorm)	(None, 24, 32, 1)	4	conv2d_57[0][0]
concatenate_4 (Concatenate)	(None, 24, 32, 4)	0	batch_normalization_47[0][0] batch_normalization_51[0][0] batch_normalization_55[0][0] batch_normalization_59[0][0]
lambda_2 (Lambda)	(None, 24, 32, 4)	0	concatenate_4[0][0]
conv2d_58 (Conv2D)	(None, 24, 32, 16)	1616	lambda_2[0][0]
batch_normalization_60 (BatchNorm)	(None, 24, 32, 16)	64	conv2d_58[0][0]
concatenate_5 (Concatenate)	(None, 24, 32, 20)	0	batch_normalization_60[0][0]

Fig 4.2.1: Summary of model

4.2.2 Dataset Training and Testing

People Count in Video.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
non_trainable_params = 16004
```

```
[20] model.compile(optimizer=Optimizer,loss=ssim_eucli_loss,metrics=[mae_cnt])
```

```
[21] X_train, X_val, Y_train, Y_val=train_test_split(images,images_density,test_size=0.15,random_state=70)
```

```
[22] Y_train=np.expand_dims(Y_train, -1)
      Y_val=np.expand_dims(Y_val,-1)
```

```
def get_patch(X,Y):
    x_patch=np.zeros([3*X.shape[0],size[0],size[0],size[1],3])
    y_patch=np.zeros([3*Y.shape[0],size[0],size[0],size[1],1])

    for i in range(X.shape[0]):
        for j in range(3):

            x1=np.random.randint(0,X.shape[1]-size[0]-1)
            y1=np.random.randint(0,X.shape[2]-size[1]-1)
            x2=size[0]+x1
            y2=size[1]+y1

            x_patch[3*i+j]=X[i,x1:x2,y1:y2,:]
            y_patch[3*i+j]=Y[i,x1:x2,y1:y2,:]

    return x_patch,y_patch
```

Fig 4.2.2: Training the dataset

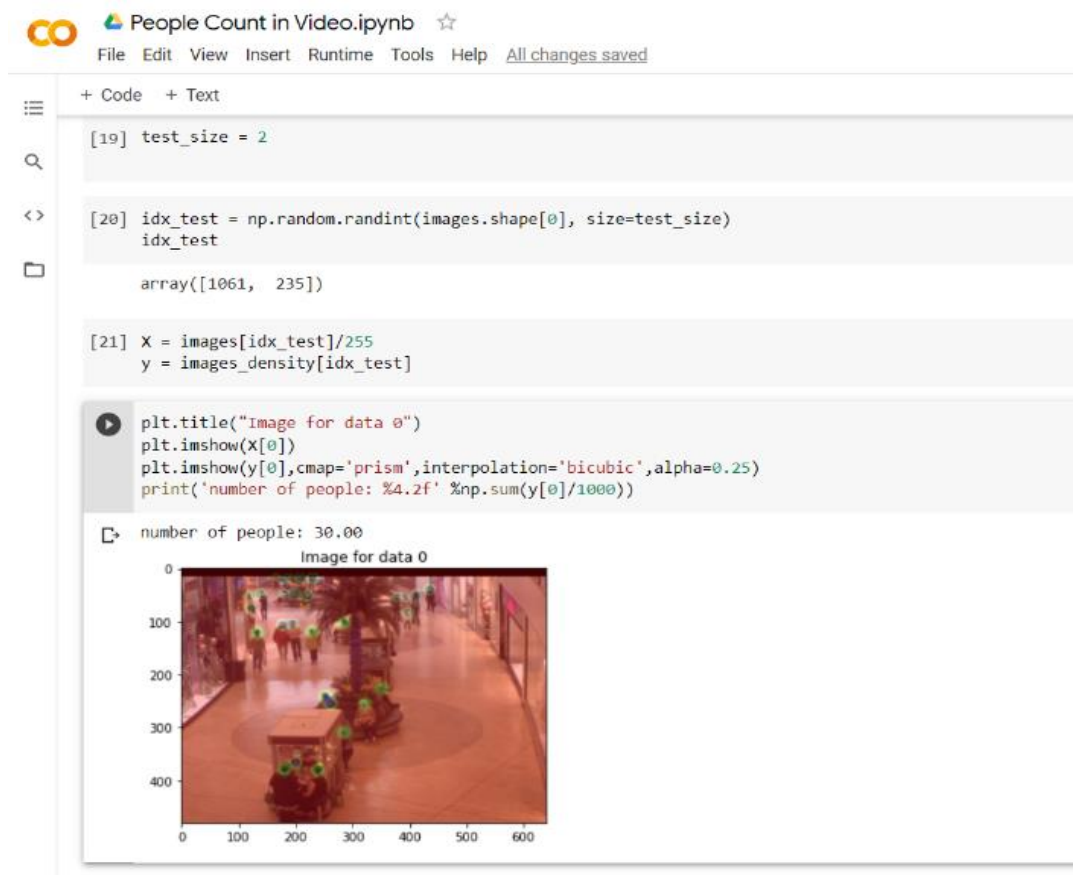


Fig 4.2.2.1: Testing and Plotting people

4.2.3 Accuracy of the model



Fig 4.2.3: Accuracy

4.2.4 Output Screenshots

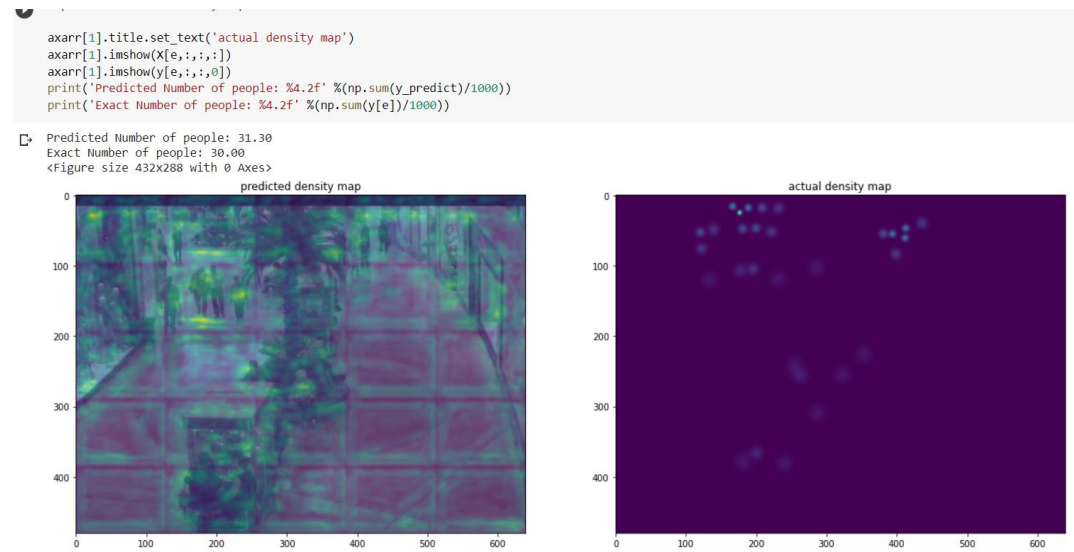


Fig 4.2.4: Output

4.2.5 Manually Testing for different images

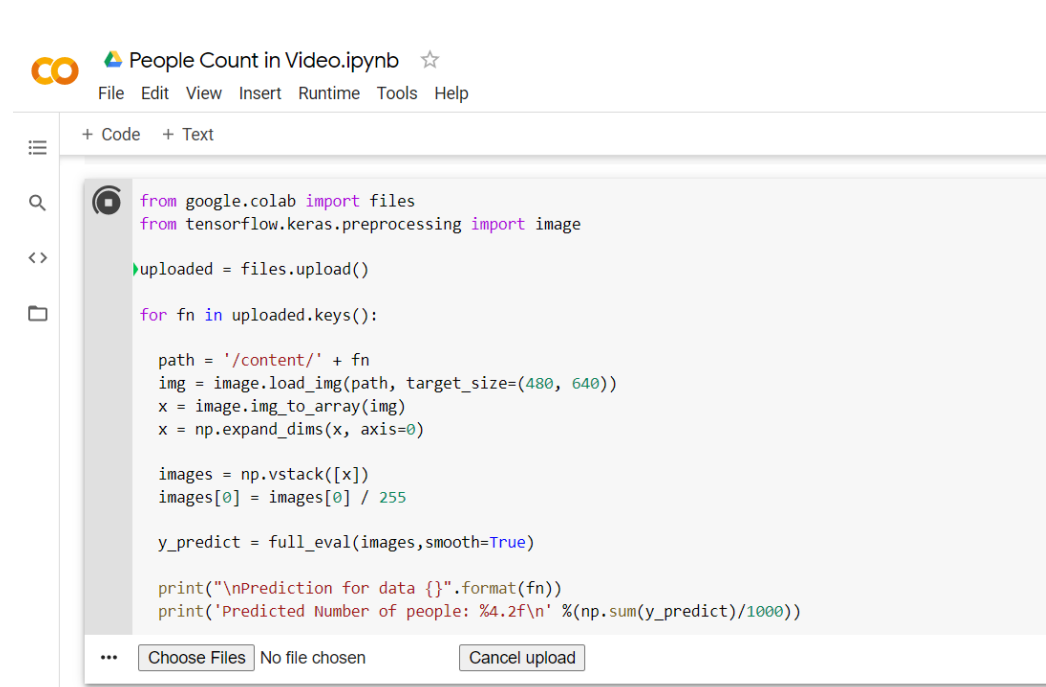


Fig 4.2.5: Manually choosing image from device

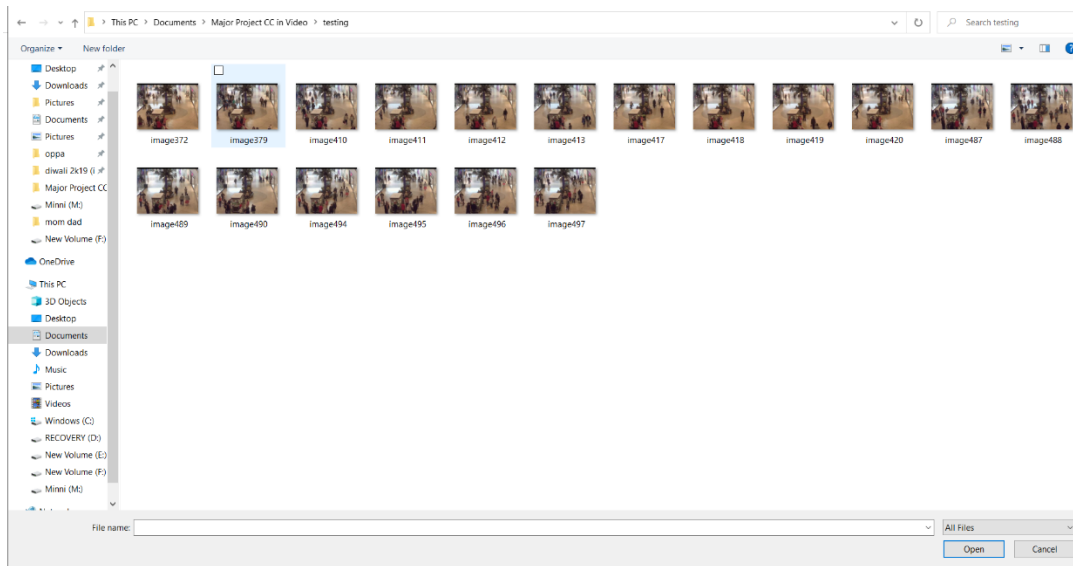


Fig 4.2.5.1: Selecting any image for prediction



Fig 4.2.5.2: Output for the selected image

5. CONCLUSION AND FUTURE SCOPE

In this project, a model is built for counting numbers of people from a crowd. The numbers of people in a crowd is counted for several reason either for safety or security purposes. Here significant algorithms for feature extraction, detection of people and tracking people are used. For feature extraction VGG16 algorithm is used, and the Structural Similarity Index(SSIM) is used to measure the similarity between two images and model can finally count the number of people present. Transposed Convolutional layer is used for the up sampling method rather than the conventional up sampling method. With this our procedure is completed with higher level of accuracy in the detection of numbers of people. This model also gives information about people location and density through the density map.

The future enhancement of this application is

- To increase the accuracy.
- To detect the people directly from on going CCTV footage, thus we can predict the total number of people in the mall as dividing video into frames may have same number of people in 5 – 10 frames if the person is at same place for so long.
- To Improve the model and predict the people count in night time video or images.

6. REFERENCES

- [1] Kowcika A, “People Count from the Crowd in Surveillance Videos”, International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017).
- [2] LokeshBoominathan, Srinivas S Skruthiventi, "A Deep Convolution Network for Dense Crowd Counting", Cornell University Library, 2016.
- [3] CCTV footage from shopping mall from shutterstock.com.
- [4] keithweaver/split-video-by-frame in python from github.
- [5] VGG16 – Convolutional Network for Classification and Detection.
- [6] Documentation on SSIM: Structural Similarity Index.