Drowning Detection System

Submitted in partial fulfillment of the
requirements of the degree of
Bachelor of Computer Engineering

by

Shardul Chavan
Sanket Dhake
Shubham Jadhav

Under Guidance of

Prof. Johnson Mathew

University of Mumbai Department of Computer Engineering Datta Meghe College of Engineering
Plot #98, Sector 3, Airoli,
Navi Mumbai, Maharashtra 400708

Academic Year 2021-22

Datta Meghe College of Engineering
Airoli, Navi Mumbai

CERTIFICATE

This is to certify that the project entitled "**Drowning Detection System**" is bonafide work of "**Shardul Chavan, Sanket Dhake, Shubham Jadhav**" submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of "undergraduate" in "computer engineering".

Prof. Johnson Mathew	**Dr. A. P. Pande**	**Dr. S. D. Sawarkar**
Project Guide	**Head of the Department**	**Principal**

Datta Meghe College of Engineering
Airoli, Navi Mumbai

Project Approval

This project report entitled **"Drowning Detection System"** of the students **"Shardul Chavan, Sanket Dhake, Shubham Jadhav"** approved for the degree of computer engineering.

Internal Examiner **External Examiner**
Date : **Date :**

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included; we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of the Students	**Signature**
Shardul Chavan	
Sanket Dhake	
Shubham Jadhav	

Acknowledgment

Motivation and guidance are the keys to success. I would like to extend my thanks to all the sources of motivation.
We would like to grab this opportunity to thank **Dr. S. D. Sawarkar,** principal for the encouragement and support he has given for our project.

We express our deep gratitude to **Dr. A. P. Pande**, Head of the department who has been the constant driving force behind the completion of this project.

We wish to express our heartfelt appreciation and a deep sense of gratitude to my project guide **Prof. Johnson Mathew** for his encouragement, invaluable support, timely help, lucid suggestions, and excellent guidance which helped us to understand and achieve the project goal. His concrete directions and critical views have greatly helped us in the successful completion of this work.
We extend our sincere appreciation to all professors for their valuable inside and tips during the designing of the project. Their contributions have been valuable in so many ways that we find it difficult to acknowledge them individually.

We are also thankful to all those who helped us directly or indirectly in the completion of this work.

Place: Airoli
Date: 11/10/21

Name of Students
Sanket Dhake
Shardul Chavan
Shubham Jadhav

Abstract

This project provides the insights of a real-time video surveillance system capable of automatically detecting drowning incidents in a swimming pool. Drowning is the 3rd reason for the highest unintentional deaths, and that’s why it is necessary to create trustable security mechanisms.

Currently, most of the swimming pool's security mechanisms include CCTV surveillance and lifeguards to help in drowning situations. But this method is not enough for huge swimming pools like in amusement parks. Nowadays, Some of security systems are using AI for drowning detection using cameras situated underwater at a fixed location and also by using floating boards having a camera mounted on the bottom side so that underwater view can be captured. But the main problems in these systems arise when the pool is crowded and vision of cameras is blocked by people.

In this project, rather than using underwater cameras, we are using cameras situated on top of the swimming pool to get an upper view of the swimming pool so that entire swimming pool will be under surveillance all time.

INDEX

List of Figures

Chapter 1

Introduction

Drowning is the 3rd reason for the highest unintentional deaths, and that’s why it is necessary to create trustable security mechanisms. This project aims to create a system that will be able to automatically detect drowning incidents in the swimming pool using human action detection. The drowning detection model will be used to process and classify video that will be given to the system which will be recorded using live surveillance cameras. The system will break this video in image frames and apply model over it and if the early actions of drowning like hand waving, water splashing or diving is detected then the system will set the alarm so that the life guards can initiate their rescue operations.

Chapter 2

Literature Review

This section aims to identify and discuss the lacunae and similarities respectively, in some of the previous works related to drowning detection systems.

- Lei Fei, Wang Xueli, Chen Dongsheng, proposed a background subtraction method for drowning detection and swimmer identification using visual surveillance in their research paper. This method fails to reflect real background accurately thus restricting model accurate shape detection of moving objects. It also fails to reflect sudden background changes.

- Ajil Roy, Dr. K. Srinivasan, proposed drowning detection using RFID-based swimming goggles, however, this model also fails to overcome the limitation of accuracy since the water sensor is not placed very close to the mouth and nose. But this model successfully overcomes limitations of video surveillance-based drowning detection systems like the need for high power computing devices.

- Chi Zhang, Xiaoguang Li, Fei Lei, proposed “A Novel Camera-Based Drowning Detection Algorithm” using input video sequence obtained from underwater cameras. In this case, to detect drowning swimmers an implementable real-time detection system with high accuracy will be needed.

Chapter 3

Requirement Analysis

3.1 Hardware Requirements: (At least)

- Processor – Intel Core i3
- RAM – 4GB
- 50 GB SSD

3.2 Software Requirements:

- Python 3.6+ (required)
- Jupyter notebook 4.0+
- Latest version of required packages

3.3 Functional requirement

- Program should be capable of analyzing video provided by user.
- User friendly working of the program.
- Early detection of drowning people in swimming pools.
- Raise an alarm on the system when drowning is detected so that lifeguards can initiate their rescue operations.
- Minimum false alarm and maximum accuracy of detection.

3.4 Operational Requirements

- The database should be capable of storing video files.
- The system should be able to process each frame of video in short time for early detection

3.5 Technical requirements

- Good quality surveillance cameras should be used.
- Cameras should be set at an angle from which maximum area can be covered with any obstruction.
- An alarm should be raised to alert the lifeguard if drowning is detected.

Chapter 4

Project Design

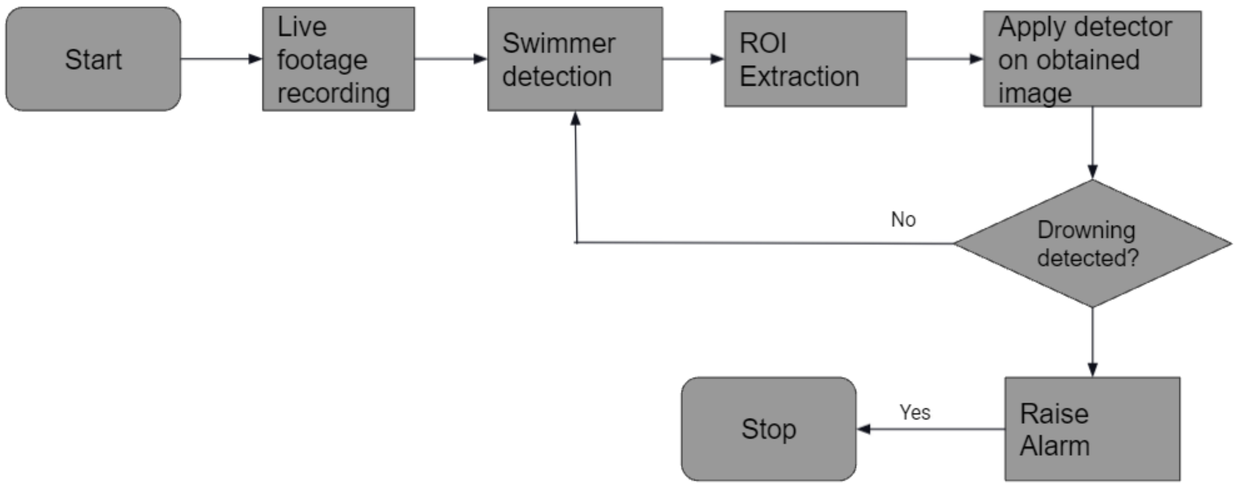


Fig. 4.1 Project Design

4.1. Identify Question or Problem:

The aim of this project is to create a model that can identify whether a person in the swimming pool is drowning or not. In the current world, the solution to this problem is already in implementation which involves monitoring people through underwater cameras situated on sidewalls of the pool and placing surfboards in the pool with having an underwater camera at bottom of the board, but in this method, it becomes difficult to get the entire view of swimming pool because of vision blocked by people. In this project, rather than using underwater cameras, we are using cameras situated on top of the swimming pool to get an upper view of the swimming pool for drowning detection.

The model will be given a set of videos wherein each video, a person will be performing an action. The label of a video will be the action that is being performed in that particular video. The model will have to learn this relationship, and then it should be able to predict the label of an input (video) that it has never seen. Technically, the model would have to learn to differentiate between various swimming actions` and drowning.

Chapter 5

Design and Implementation

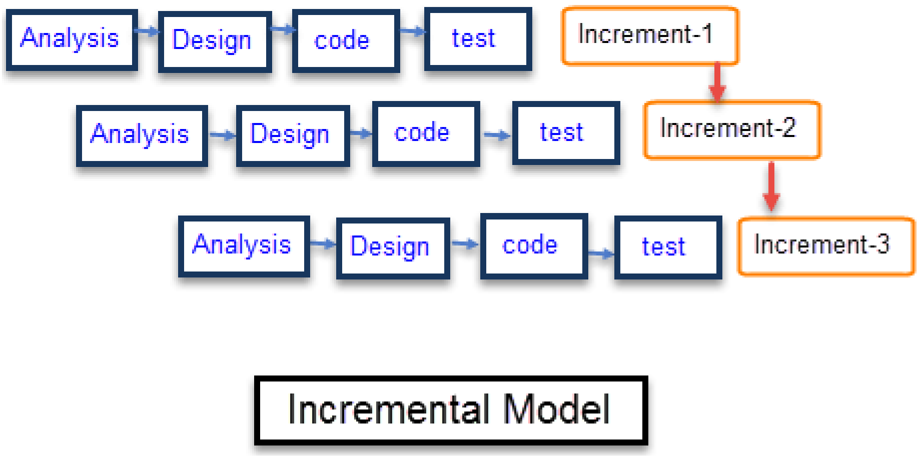


Fig.5.1 Software Design

5.1. For Software Design: Incremental Model

Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of the software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.

Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.

The system is put into production when the first increment is delivered. The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments. Once the core product is analyzed by the client, there is a plan development for the next increment.

5.2. Analyzing & training the model:

In our project, we need a model which can predict whether the person detected in the input video is swimming or drowning. For that we have trained a LRCN model and using multiple videos of swimming and drowning category as a training set. Within the swimming category we have provided videos related to multiple swimming types like backstroke, freestyle swimming, butterfly, breaststroke, etc. To get better accuracy videos used for training are cropped such that only required activities are visible in video.

In this project we have used Long-term Recurrent Convolutional Network (LRCN) model. Each frame of video will be passed sequentially from this models for drowning detection.

5.3. Testing and Deployment:

When the videos are pre-processed the method named as train_test_split() is used to divide data in training and testing sets. The testing set is not used while training so that effectiveness of the model can be tested on entirely new videos. In this test process the model will be applied on videos in the test set and the prediction of the model is verified in the form of accuracy metrics which consists of precision, recall & f1-score. This accuracy also depends on data generated by model training process in form of plot diagram which consists of curves for training loss, training accuracy, value loss, value accuracy with respect to loss/accuracy on y-axis and epochs on x-axis, in ideal model, accuracy should be near to 1 and loss should be near to 0.

Once these test results satisfy the expectations of the user then this software will be deployed and installed in the system to which the swimming pool surveillance camera will be connected. For working of the project python versions higher than 3.6 are required with all the updated packages. The system should have high computational power. User will have to move the project file and saved model in a single folder and run the file to start the surveillance process of the project. The project will keep running until the user presses the escape key. There is no time limitation to run the project. As long as the system and camera is on the project will keep running.

5.4. Software Implementation:

The tasks involved are the following:

- Downloading, extracting and pre-processing a video dataset

  The video dataset used for training the LRCN model is a blend of UCF50 dataset & videos that are downloaded from Youtube. The video dataset for swimming and drowning are not available on any websites so we had to download related videos for the same one by one from youtube.

  But these videos had extra activities like in swimming videos there was a person approaching the swimming competition , taking their stance and handshakes after the race. In drowning related videos there was rescue operation as well in video. So we cropped such extra activities and kept only the required part of the video which were actual swimming and drowning actions.

  Finally the videos of diving, breaststroke swimming from UCF50 video dataset where combined with the data that was created by us for further training of the models.

  For training the model image frames were extracted from each video and used for model training, and same for testing.

- Dividing the dataset into training and testing data

  The videos which we used for creating the model were splitted in training and testing sets using train_test_split() methods.The testing set is not used while training so that effectiveness of the model can be tested on entirely new videos.

- Create a neural network and train it on the training data

  In this project we have used 2 models which are ConvLstm2D model and LRCN model. In this phase we used training set videos to train the model by passing them through multiple layers of ConvLstm2D network and then implemented LRCN network to compare the better results provide by the respective models. The architecture of both models are as follows are as follows:

ConvLstm2D model:

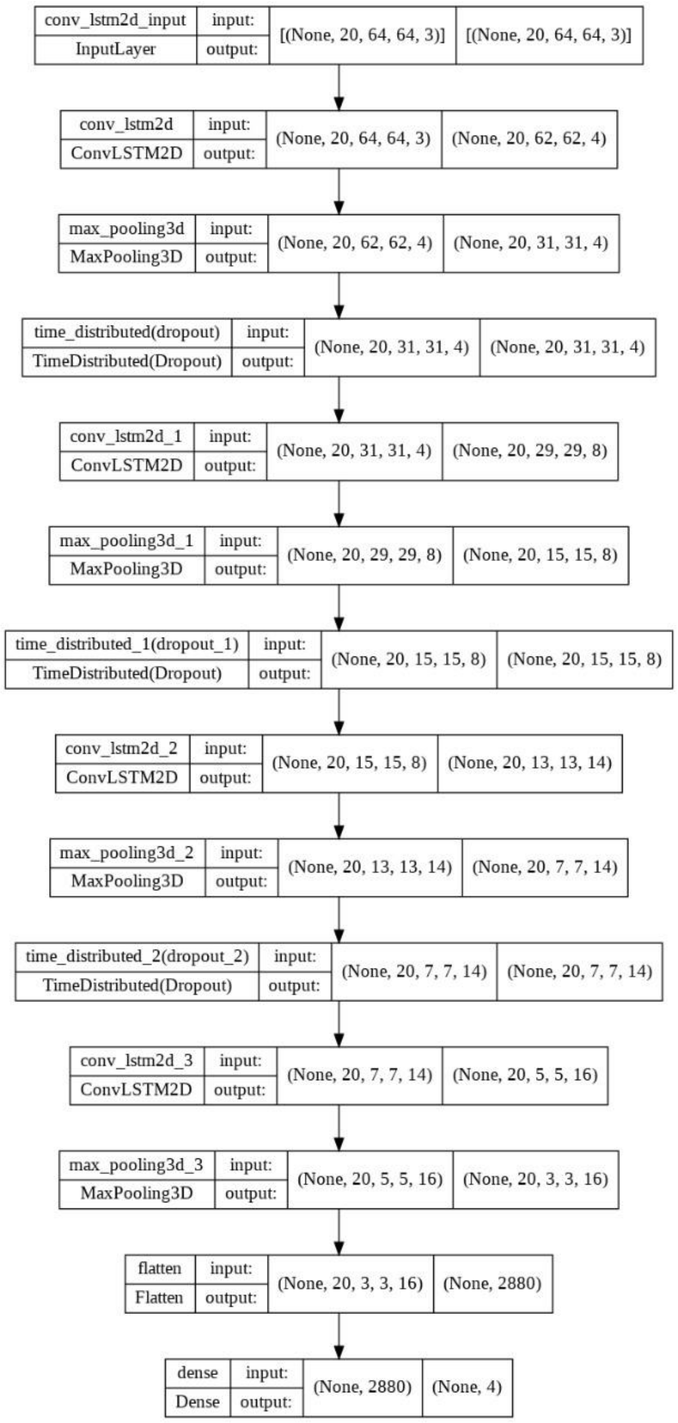
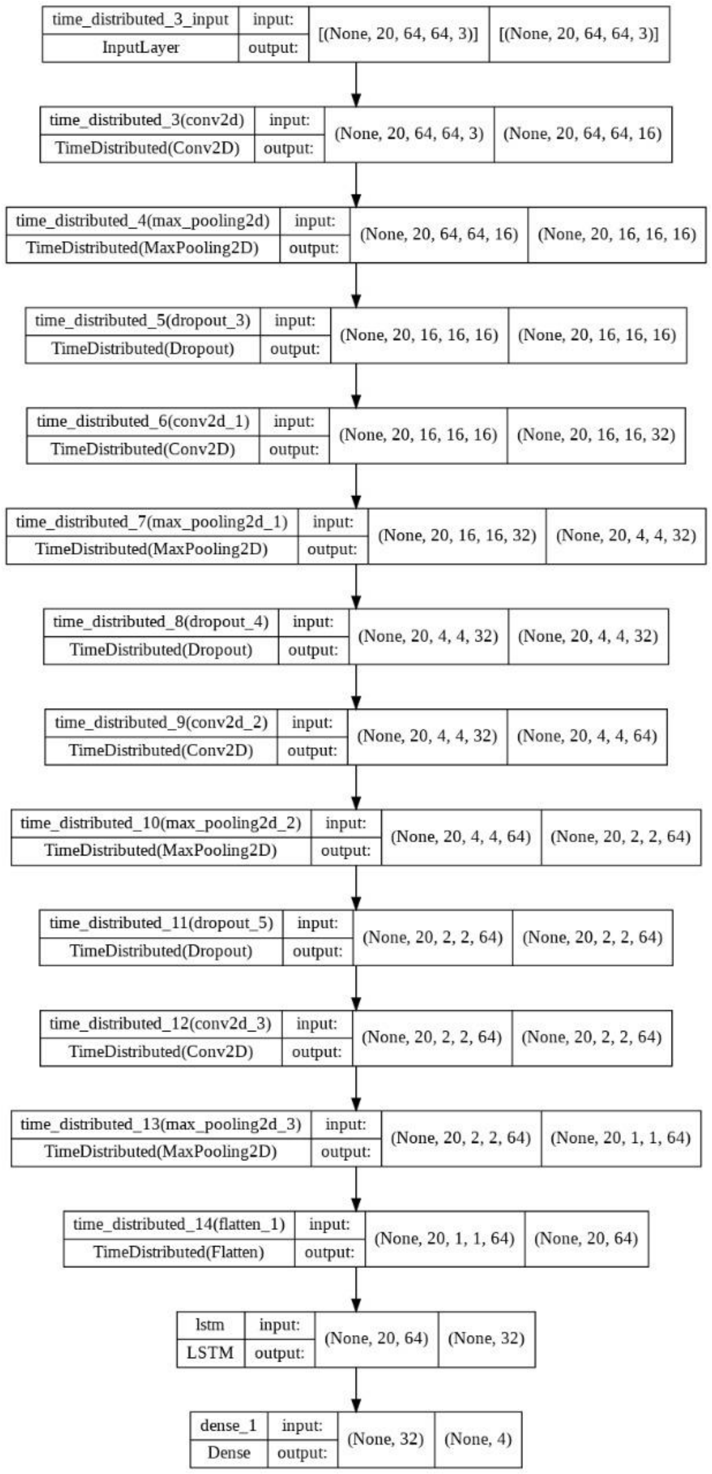

conv_lstm2d_input | InputLayer | input: [(None, 20, 64, 64, 3)] | output: [(None, 20, 64, 64, 3)]
conv_lstm2d | ConvLSTM2D | input: (None, 20, 64, 64, 3) | output: (None, 20, 62, 62, 4)
max_pooling3d | MaxPooling3D | input: (None, 20, 62, 62, 4) | output: (None, 20, 31, 31, 4)
time_distributed(dropout) | TimeDistributed(Dropout) | input: (None, 20, 31, 31, 4) | output: (None, 20, 31, 31, 4)
conv_lstm2d_1 | ConvLSTM2D | input: (None, 20, 31, 31, 4) | output: (None, 20, 29, 29, 8)
max_pooling3d_1 | MaxPooling3D | input: (None, 20, 29, 29, 8) | output: (None, 20, 15, 15, 8)
time_distributed_1(dropout_1) | TimeDistributed(Dropout) | input: (None, 20, 15, 15, 8) | output: (None, 20, 15, 15, 8)
conv_lstm2d_2 | ConvLSTM2D | input: (None, 20, 15, 15, 8) | output: (None, 20, 13, 13, 14)
max_pooling3d_2 | MaxPooling3D | input: (None, 20, 13, 13, 14) | output: (None, 20, 7, 7, 14)
time_distributed_2(dropout_2) | TimeDistributed(Dropout) | input: (None, 20, 7, 7, 14) | output: (None, 20, 7, 7, 14)
conv_lstm2d_3 | ConvLSTM2D | input: (None, 20, 7, 7, 14) | output: (None, 20, 5, 5, 16)
max_pooling3d_3 | MaxPooling3D | input: (None, 20, 5, 5, 16) | output: (None, 20, 3, 3, 16)
flatten | Flatten | input: (None, 20, 3, 3, 16) | output: (None, 2880)
dense | Dense | input: (None, 2880) | output: (None, 4)

Fig 5.2 ConvLstm2D Model Architecture

LRCN model:



time_distributed_3_input | input: | [(None, 20, 64, 64, 3)]
InputLayer | output: | [(None, 20, 64, 64, 3)]
time_distributed_3(conv2d) | input: | (None, 20, 64, 64, 3)
TimeDistributed(Conv2D) | output: | (None, 20, 64, 64, 16)
time_distributed_4(max_pooling2d) | input: | (None, 20, 64, 64, 16)
TimeDistributed(MaxPooling2D) | output: | (None, 20, 16, 16, 16)
time_distributed_5(dropout_3) | input: | (None, 20, 16, 16, 16)
TimeDistributed(Dropout) | output: | (None, 20, 16, 16, 16)
time_distributed_6(conv2d_1) | input: | (None, 20, 16, 16, 16)
TimeDistributed(Conv2D) | output: | (None, 20, 16, 16, 32)
time_distributed_7(max_pooling2d_1) | input: | (None, 20, 16, 16, 32)
TimeDistributed(MaxPooling2D) | output: | (None, 20, 4, 4, 32)
time_distributed_8(dropout_4) | input: | (None, 20, 4, 4, 32)
TimeDistributed(Dropout) | output: | (None, 20, 4, 4, 32)
time_distributed_9(conv2d_2) | input: | (None, 20, 4, 4, 32)
TimeDistributed(Conv2D) | output: | (None, 20, 4, 4, 64)
time_distributed_10(max_pooling2d_2) | input: | (None, 20, 4, 4, 64)
TimeDistributed(MaxPooling2D) | output: | (None, 20, 2, 2, 64)
time_distributed_11(dropout_5) | input: | (None, 20, 2, 2, 64)
TimeDistributed(Dropout) | output: | (None, 20, 2, 2, 64)
time_distributed_12(conv2d_3) | input: | (None, 20, 2, 2, 64)
TimeDistributed(Conv2D) | output: | (None, 20, 2, 2, 64)
time_distributed_13(max_pooling2d_3) | input: | (None, 20, 2, 2, 64)
TimeDistributed(MaxPooling2D) | output: | (None, 20, 1, 1, 64)
time_distributed_14(flatten_1) | input: | (None, 20, 1, 1, 64)
TimeDistributed(Flatten) | output: | (None, 20, 64)
lstm | input: | (None, 20, 64)
LSTM | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 4)

Fig 5.3 LRCN Model Architecture

- Test the model on the test data

  In this test phase we applied the model on videos in the test set and the prediction of the model is verified in the form of accuracy matrix which consists of precision, recall, f1-score, support as rows and micro avg, macro avg and weighted avg as column. The values within the matrix are between 0 and 1 which shows the performance of the trained model. This accuracy also depends on data generated by model training process in form of plot diagram which consists of curves for training loss, training accuracy, value loss , value accuracy with respect to loxx/accuracy on y-axis and epochs on x-axis, In ideal model, accuracy should be near to 1 and loss should be near to 0.

- Compare the performance of the model with some pre-existing models

  The performance of the model can be compared using prediction matrix and accuracy/loss plots of both models. Higher the number close to 1 better the performance.

Chapter 6

Technologies Used

We already know that neural networks perform very well for image recognition. In particular, a specific type of neural networks called Convolutional Neural Networks (CNNs) is best suited for the task of image recognition. So for our project we are going to implement Long Term Recurrent Convolution Network (LRCN) approach.

6.1 LRCN Approach

The Long Term Recurrent Convolution Network is proposed by Jeff Donahue in 2016. The proposed Long-term Recurrent Convolutional Network approach is a combination of Convolutional Neural Network (CNN) & Recurrent Neural Network (RNN). It is end-to-end trainable and suitable for large-scale visual understanding tasks such as video description, activity recognition and image captioning. It can be trained or learn temporal dynamics and convolutional perceptual representations as it is directly connected to convolutional network. The main idea is to use a combination of CNNs to learn visual features from video frames and LSTMs to transform a sequence of image embeddings into a class label, sentence, probabilities, or whatever you need. Thus, raw visual input is processed with a CNN, whose outputs are fed into a stack of recurrent sequence models.

The proposed approach consists of CNN for feature extraction from the input provided to the model and then followed by LSTM layers that predict the action of the human whether one is drowning, swimming or diving.

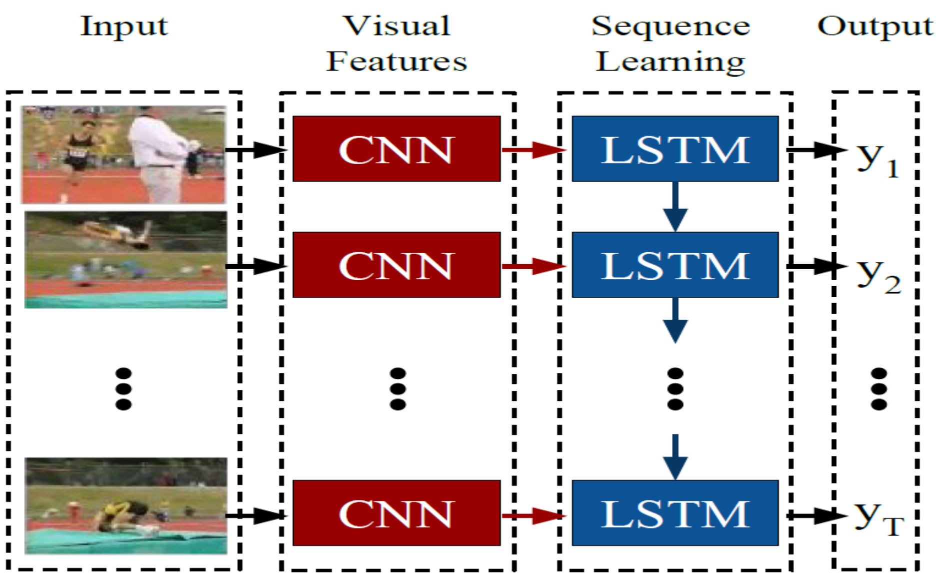


Fig.6.1 LRCN Architecture

6.2. Convolutional Neural Networks

Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

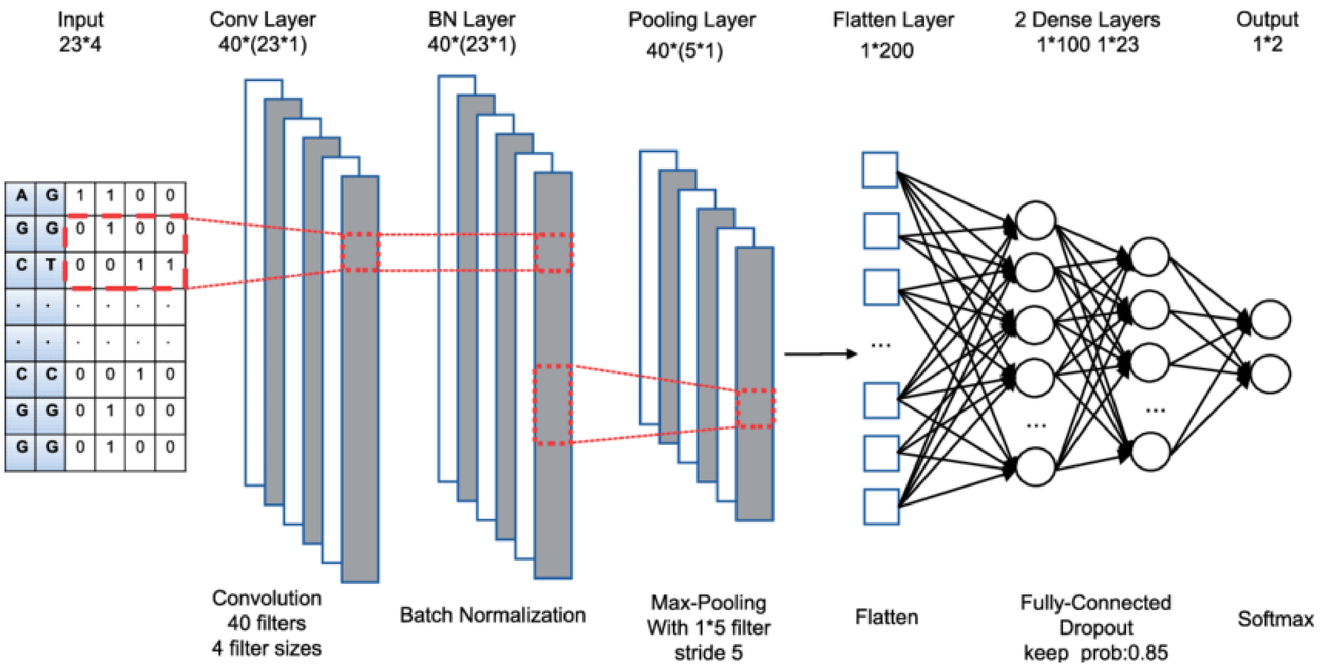


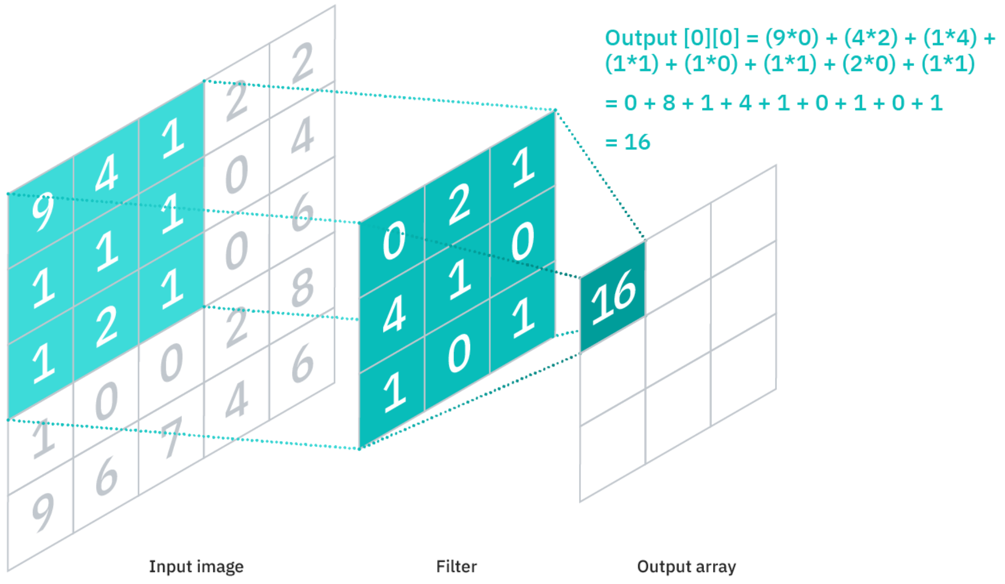
Fig.6.2 CNN Architecture

The network starts with the input, which is the image. It is passed sequentially into Convolutional and Pooling layers. Then all Fully connected layers end with the output layer. There are three basic layers, of which the architecture of the convolutional neural network is built. Such a formation allows the network to give an image at the input, and at the output obtain the probability of a class occurrence in the image.

6.3. Convolutional Layer

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.



Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

Fig.6.3 Convolutional Layer

As you can see in the image above, each output value in the feature map does not have to connect to each pixel value in the input image. It only needs to connect to the receptive field, where the filter is being applied. Since the output array does not need to map directly to each input value, convolutional (and pooling) layers are commonly referred to as “partially connected” layers. However, this characteristic can also be described as local connectivity.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of back propagation and gradient descent. However, there are three hyper parameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

1. The **number of filters** affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.
2. **Stride** is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.
3. **Zero-padding** is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding:
    a. **Valid padding:** This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
    b. **Same padding:** This padding ensures that the output layer has the same size as the input layer
    c. **Full padding:** This type of padding increases the size of the output by adding zeros to the border of the input.

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.

Convolutional neural networks use multiple filters to find image features that will allow for object categorization. In the CNN scheme there are many kernels responsible for extracting these features. The edge kernel is used to highlight large differences in pixel values. If the pixel next to it has a similar intensity, it will be black after the transformation. A big difference between them will make it white.

6.4. Pooling layer

Pooling layers, also known as down sampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

Max pooling: As filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling

Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of over fitting.

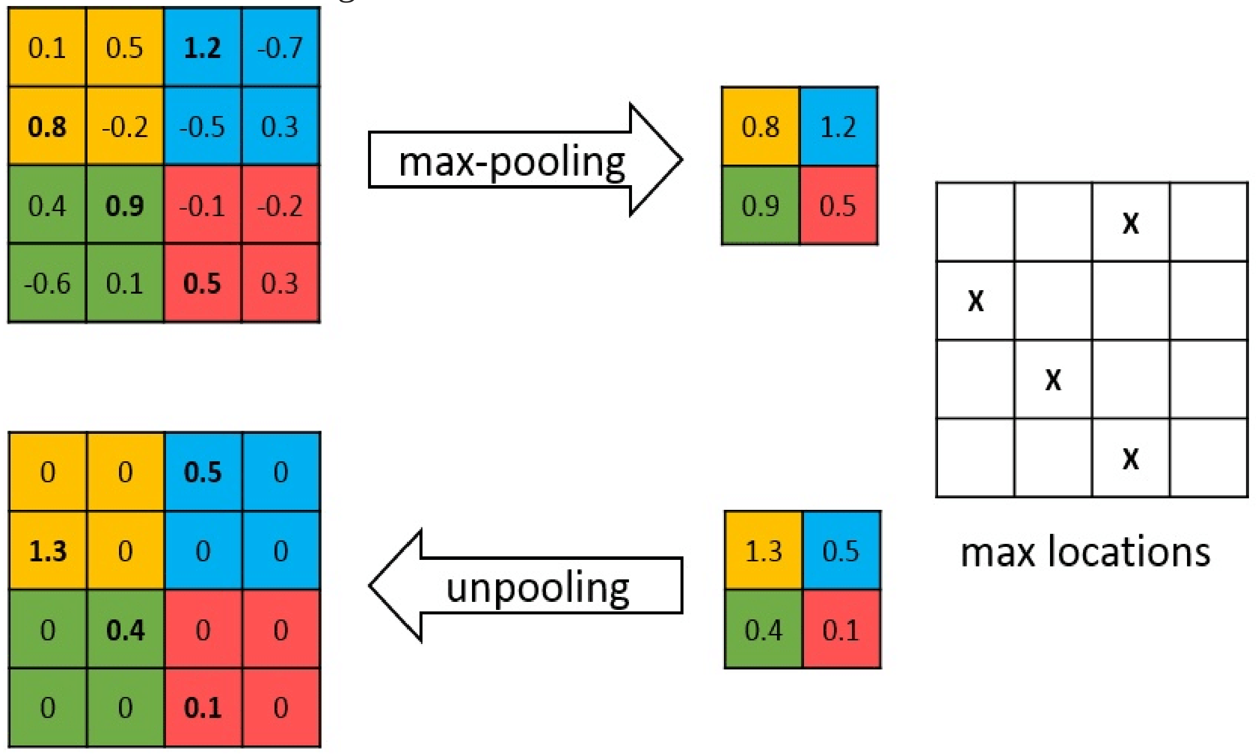


Fig.6.5 Max Pooling

6.5. Fully Connected layer

Input data to a fully connected layer is the output data from the last connecting layer. The task of the FC layer is to flatten the value in the vector. This vector is combined with other fully connected layers, creating on top of the Convolutional Neural Network simple neural network. The last layer is responsible for determining the probability of belonging to a particular class.

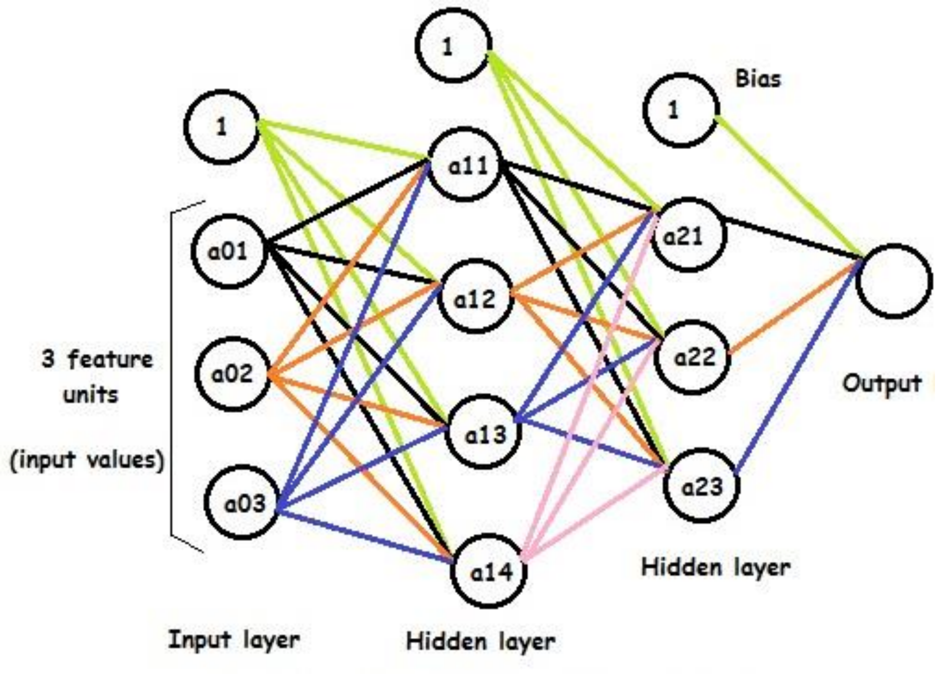


Fig.6.6 CNN with Fully Connected Layer

In the above model:

- The first/input layer has 3 feature units and there are 4 activation units in the next hidden layer.
- The 1's in each layer are bias units.
- a01, a02 and a03 are input values to the neural network.They are basically features of the training example.
- The 4 activation units of first hidden layer is connected to all 3 activation units of second hidden layer the weights/parameters connect the two layers.

6.6. Flattening layer

The last stage of a convolutional neural network (CNN) is a classifier. It is called a dense layer, which is just an artificial neural network (ANN) classifier and an ANN classifier needs individual features, just like any other classifier. This means it needs a feature vector.

Therefore, we need to convert the output of the convolutional part of the CNN into a 1D feature vector, to be used by the ANN part of it. This operation is called flattening. It gets the output of the convolutional layers, flattens all its structure to create a single long feature vector to be used by the dense layer for the final classification.

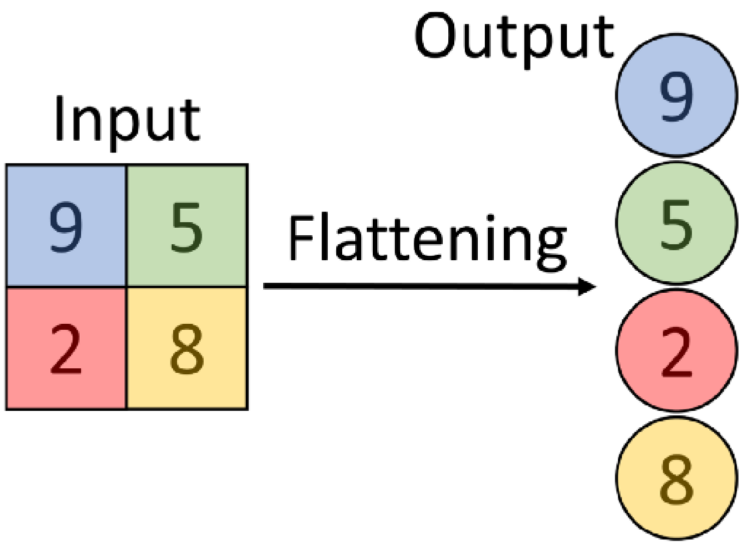


Fig.6.7 Flattening layer of Neural Network

6.7 Long Short-Term Memory Networks

LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. Talking about RNN, it is a network that works on the present input by taking into consideration the previous output (feedback) and storing in its memory for a short period of time (short-term memory). Out of its various applications, the most popular ones are in the fields of speech processing, non-Markovian control, and music composition. Nevertheless, there are drawbacks to RNNs. First, it fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such "long-term dependencies". Second, there is no finer control over which part of the context needs to be carried forward and how much of the past needs to be 'forgotten'. Other issues with RNNs are exploding and vanishing gradients (explained later) which occur during the training process of a network through backtracking. Thus, Long Short-Term Memory (LSTM) was brought into the picture. It has been so designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered. Long time lags in certain problems are bridged using LSTMs where they also handle noise, distributed representations, and continuous values. With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden Markov model (HMM). LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for fine adjustments. The complexity to update each weight is reduced to O(1) with LSTMs, similar to that of Back Propagation Through Time (BPTT), which is an advantage.

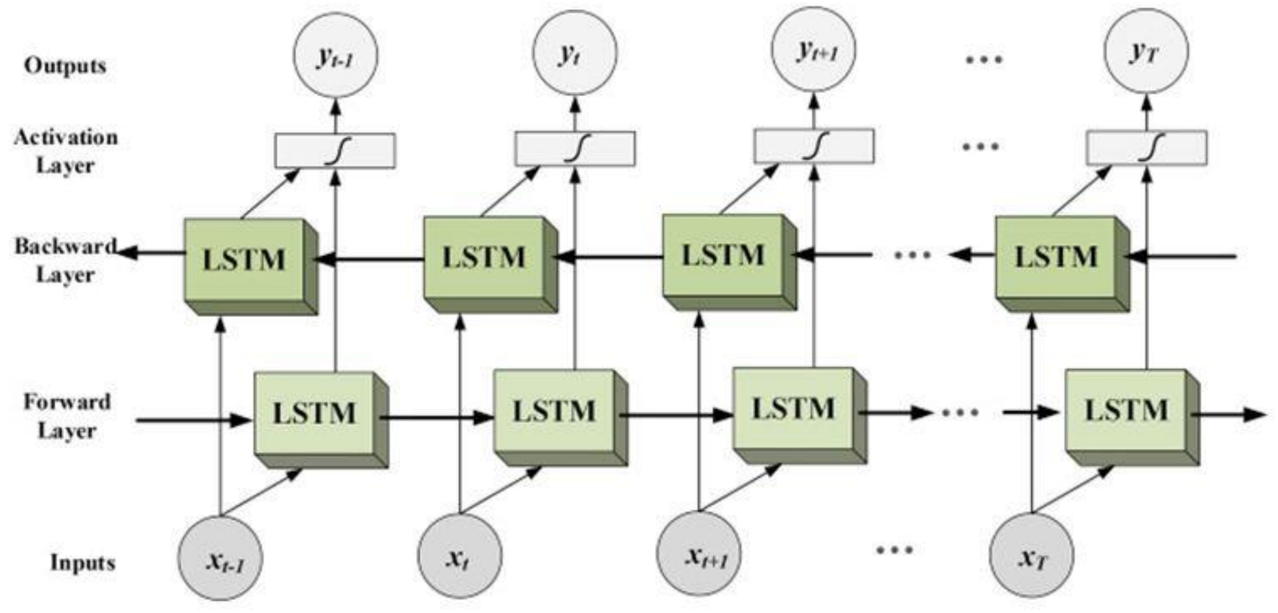


Fig.6.8 LSTM Network

Chapter 7

Testing

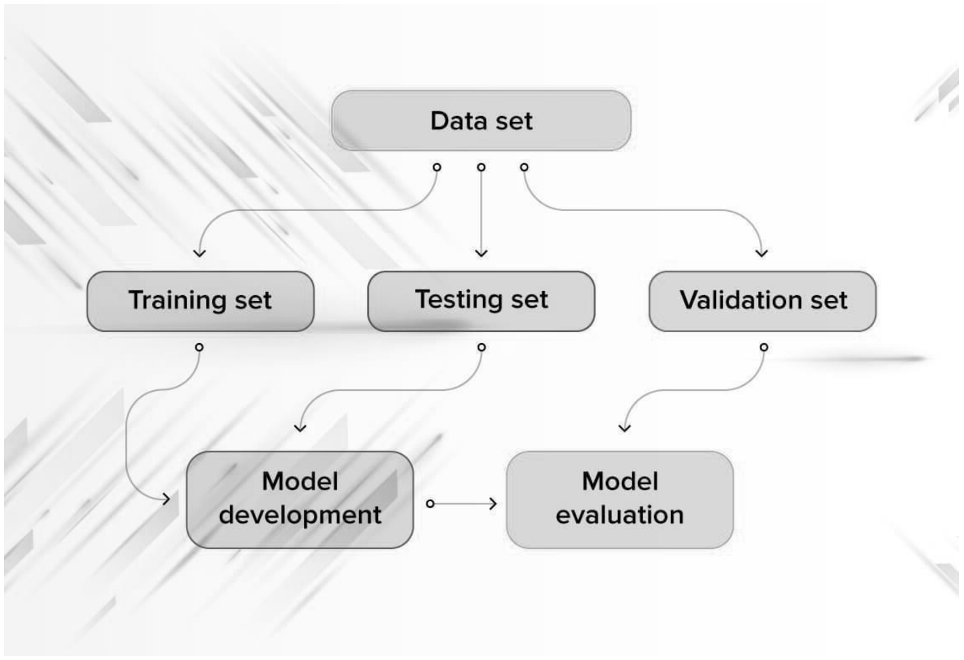


Fig.7.1 Software Testing Architecture

- **A brief procedure:**
    - The entire dataset is divided into 3 parts – training data, validation data and test data.
    - The model is trained on the training data repeatedly for a number of iterations. These iterations are known as epochs. After each epoch, the model is tested using the validation data.
    - Finally, the model that performed the best on the validation data is loaded.
    - The performance of this model is then evaluated using the test data.

- **Training set**

  A training set is a portion of a data set used to fit (train) a model for prediction or classification of values that are known in the training set, but unknown in other (future) data. The training set is used in conjunction with validation and/or test sets that are used to evaluate different models.

- **Validation set.**

  Having only a training set and a testing set is not enough if we do many rounds of hyper parameter-tuning (which is always). And that can result in overfitting. To avoid that, we can select a small validation data set to evaluate a model. Only after we get maximum accuracy on the validation set, we make the testing set come into the game.

- **Test set (or holdout set).**

  Our model might fit the training dataset perfectly well. But where are the guarantees that it will do equally well in real life? In order to assure that, we select samples for the testing set from our training set, for example, the samples that the machine hasn’t seen before. It is important to remain unbiased during selection and draw samples at random. Also, we should not use the same set many times to avoid training on our test data. Our test set should be large enough to provide statistically meaningful results and be representative of the data set as a whole.

In this test phase we applied the model on videos in the test set and the prediction of the model is verified in the form of accuracy matrix which consists of precision, recall, f1-score, support as rows and micro average, macro average and weighted average as column. The values within the matrix are between 0 and 1 which shows the performance of the trained model. This accuracy also depends on data generated by model training process in form of plot diagram which consists of curves for training loss, training accuracy, value loss, value accuracy with respect to loss/accuracy on y-axis and epochs on x-axis, In ideal model, accuracy should be near to 1 and loss should be near to 0.

Accuracy graph after implementing ConvLstm2D model:

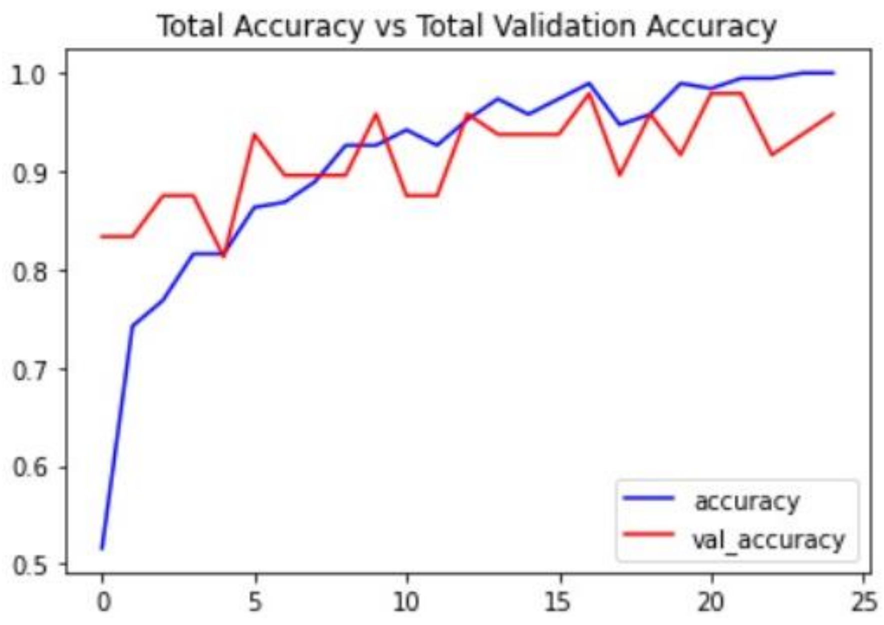


Fig 7.2 Accuracy Graph for ConvLstm2D

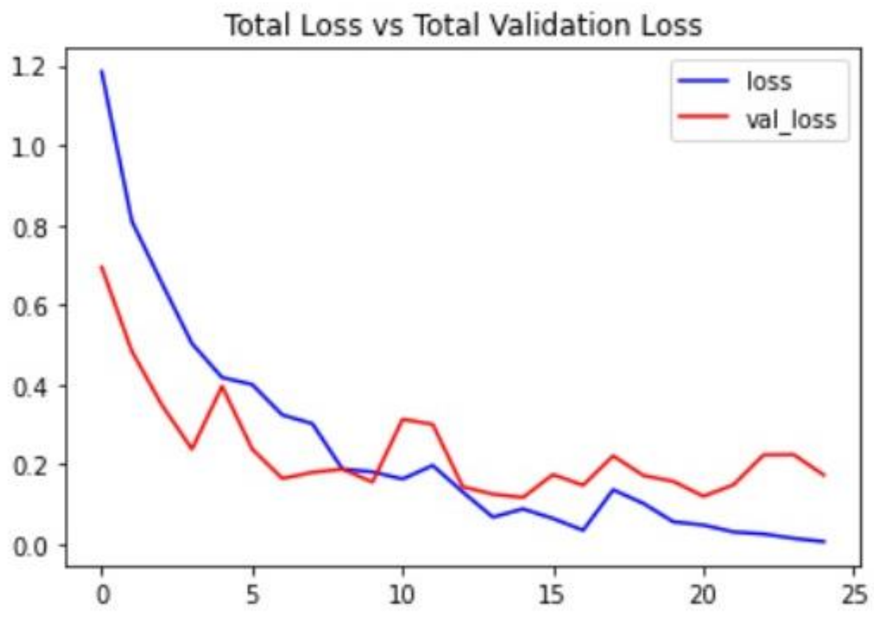


Fig 7.3 Loss Graph for ConvLstm2D

Accuracy graph after implementing LRCN model:

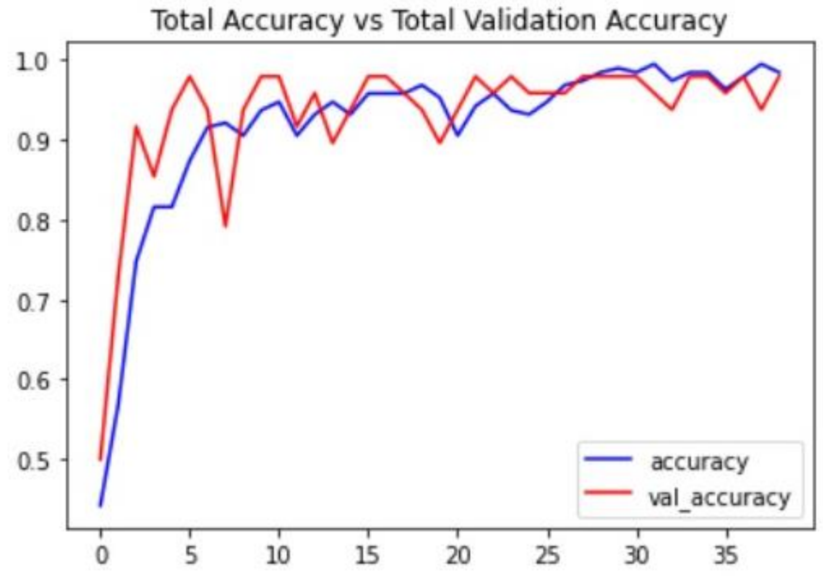


Fig 7.4 Accuracy Graph for LRCN

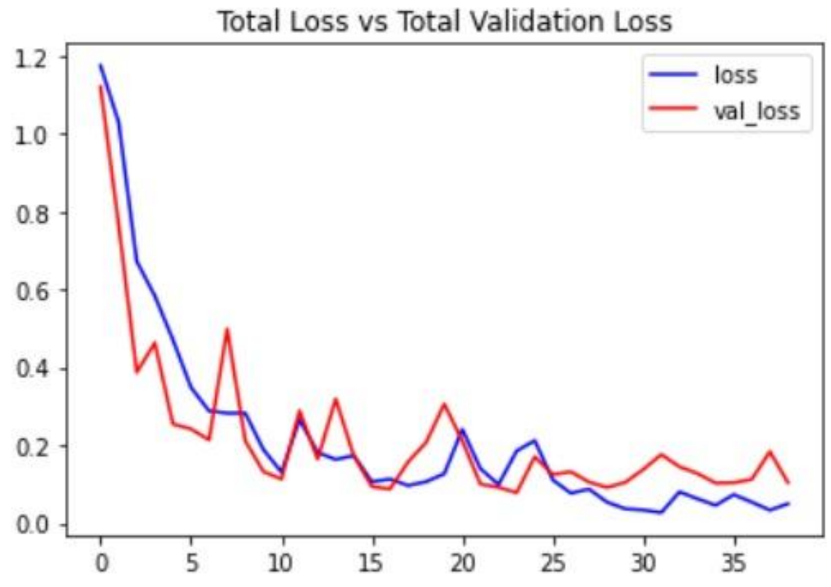


Fig 7.5 Loss Graph for LRCN

Features to be tested:

This is a listing of what is to be tested from the USERS viewpoint of what the system does. This is not a technical description of the software, but a USERS view of the functions. It differs significantly in comparison Identifying Test Items. Low-level non-technical descriptions Level of risks identified.

Software's to be tested include:

- Windows 10 operating system
- Python version
- Packages version
- Camera's Software

Hardware's to be tested include:

•Computer system
•Camera

Project Test Cases:

Test case id	Test Case objective	Steps	I/P Data	Expected Results	Actual Results	Status
TC_1	User can use live camera input	1.Connect the camera 2.run the program	Live video	Program should accept live camera video	Program Accepted live camera video	pass
TC_2	User can use saved video file as input	1.Move input video file to programs folder 2.run the program	Saved Video	Program should accept provided video	Program accepted provided video	pass
TC_3	Allows user to stop program execution	1.press esc key	Esc key	Program should stop execution	Program stopped execution	pass
TC_4	Alarms will be sounded once drowning detected	1.Run program on provided video 2.If drowning is detected alarm will sound	Live video	Program should sound alarm when drowning is detected	Program sounded an alarm when drowning was detected.	pass

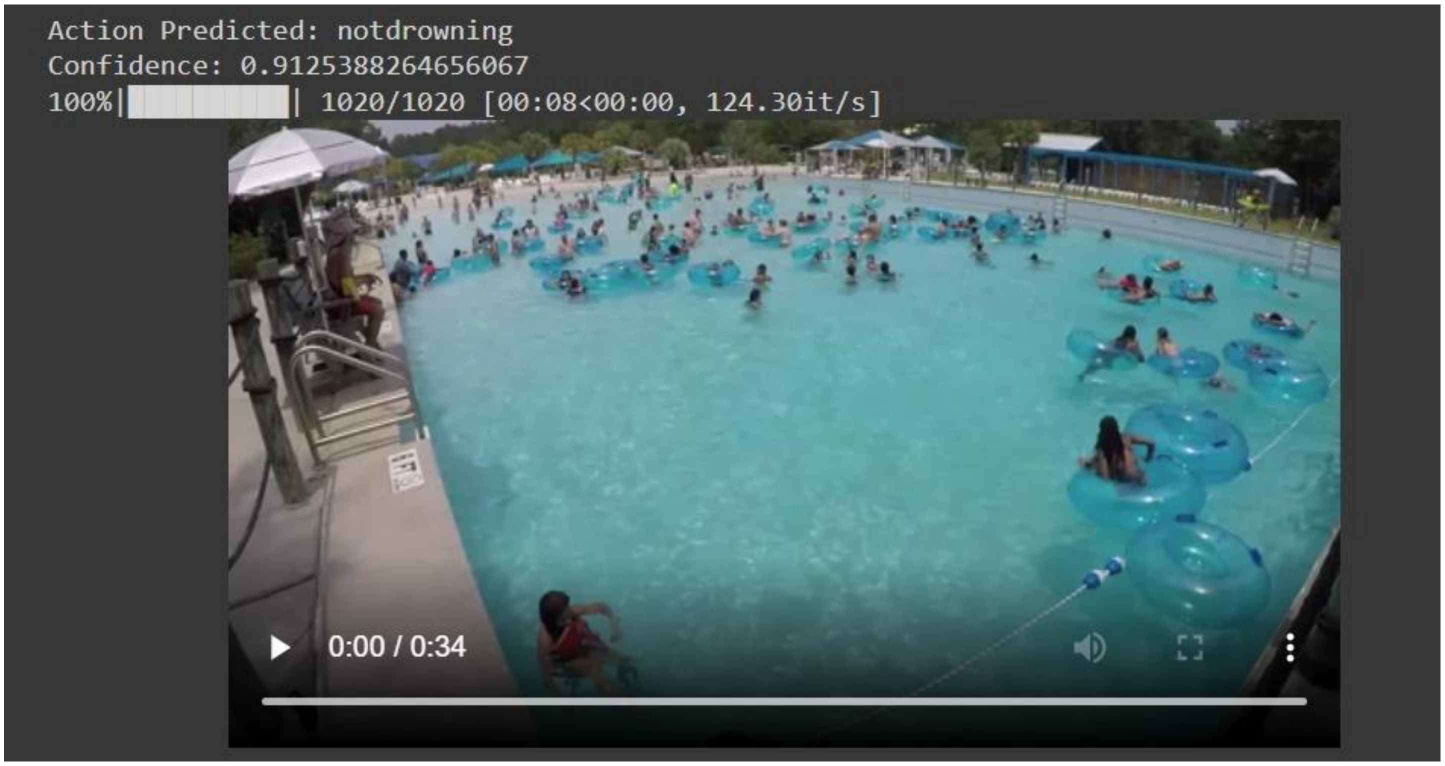
Chapter 8

Results and Discussions

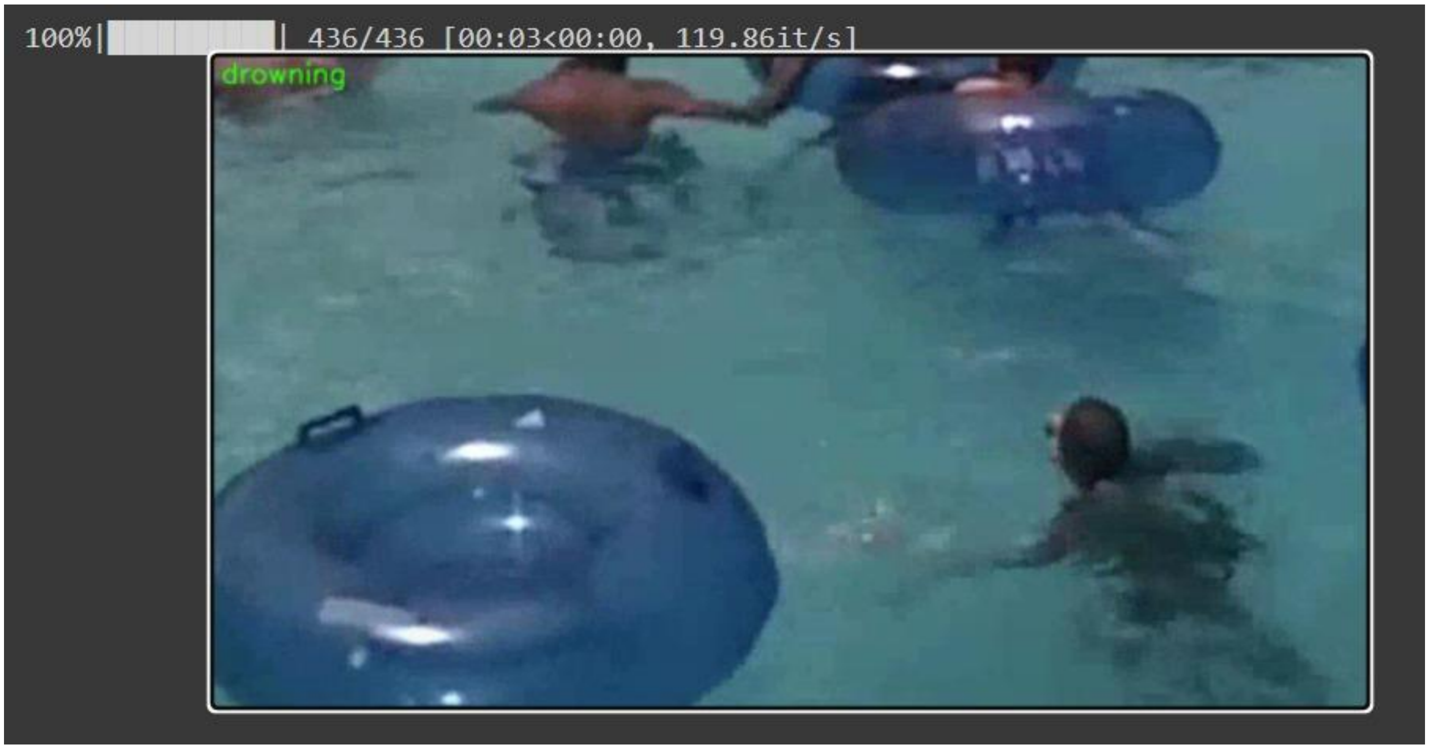
After the successful completion of the project, one can:

1. Observe the video surveillance and rely on the drowning detection system.
2. An alarm will be raised if someone's detected drowning.
3. Drowning preventive measures can be performed due to early alerts raised by the system.
4. The project could examine the actions performed by swimmers to detect drowning more accurately.

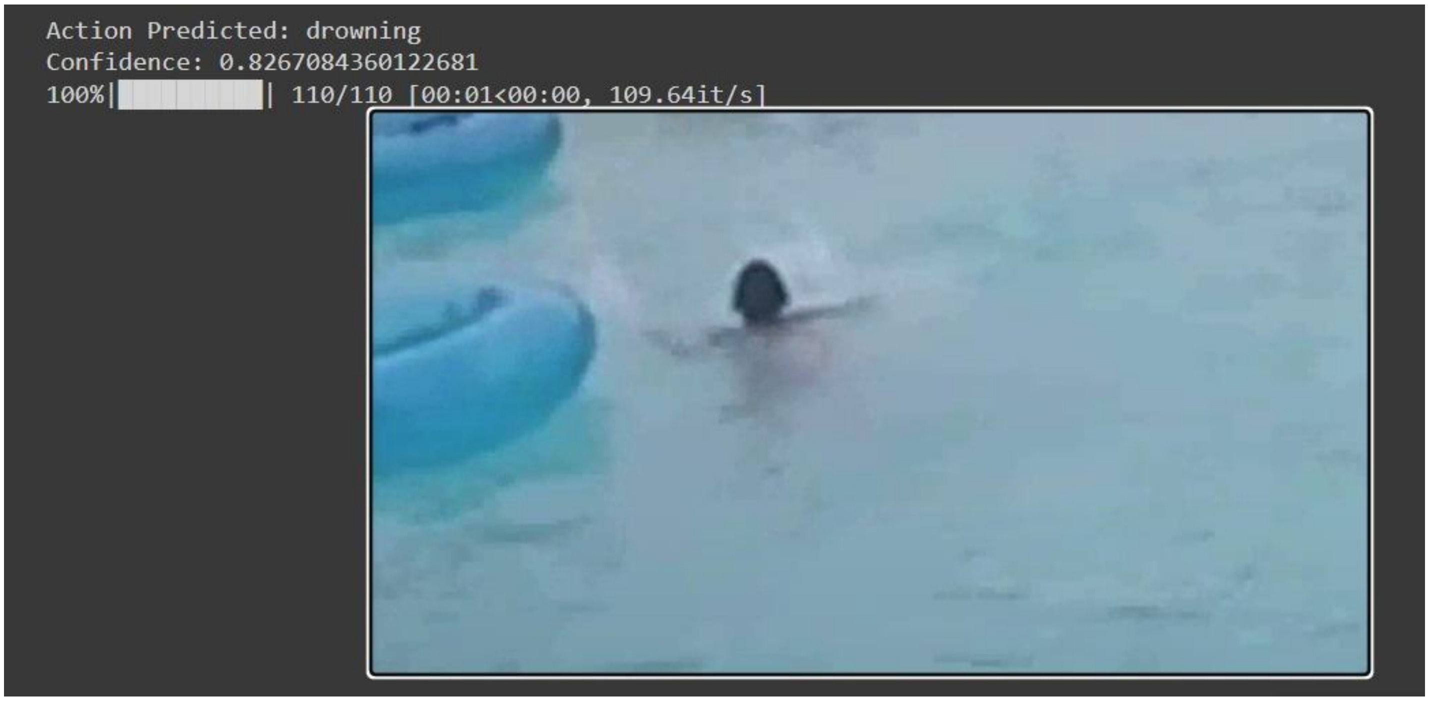
Output:



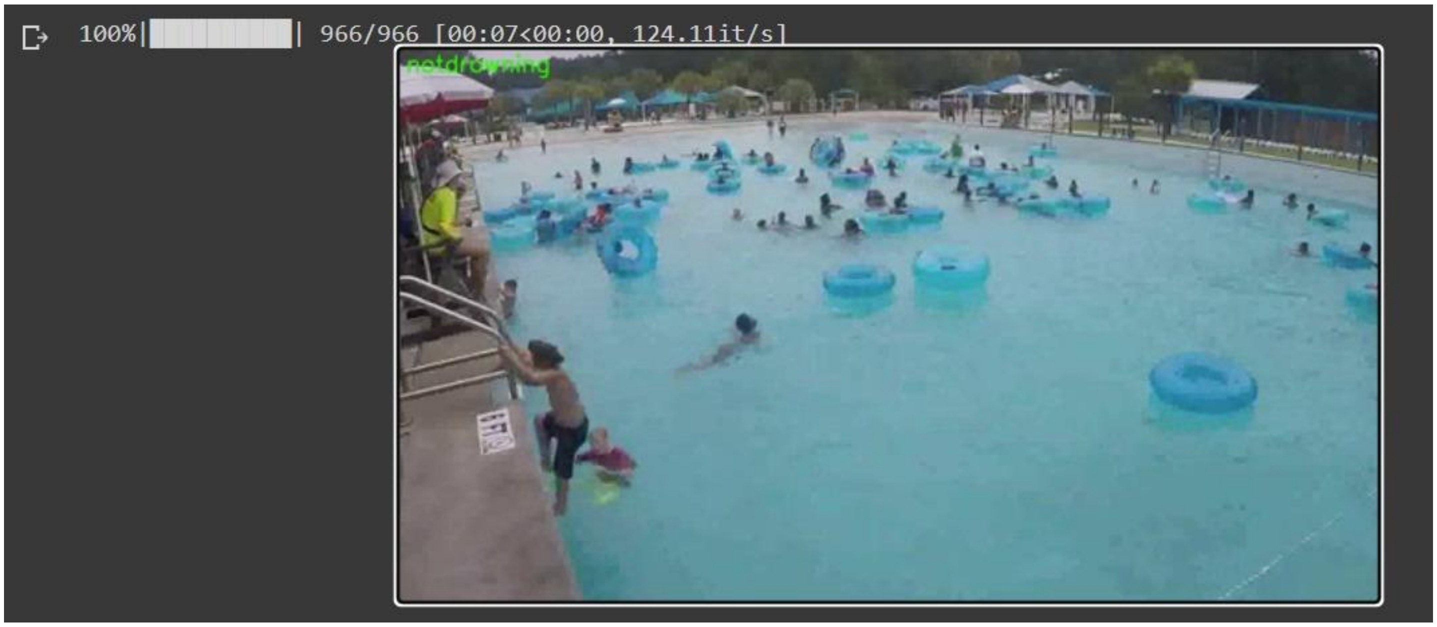
```
Action Predicted: notdrowning
Confidence: 0.9125388264656067
100%|██████████| 1020/1020 [00:08<00:00, 124.30it/s]
```



```
100%|██████████| 436/436 [00:03<00:00, 119.86it/s]
```



```
Action Predicted: drowning
Confidence: 0.8267084360122681
100%|██████████| 110/110 [00:01<00:00, 109.64it/s]
```



```
100%|██████████| 966/966 [00:07<00:00, 124.11it/s]
```

Chapter 9

Conclusions and Future Scope

Conclusion:

Once we have the working drowning detection model we can feed live video footage of the swimming pool to it so that it can keep detecting continuously for any drawing activities. If drowning is detected it will be highlighted on the system screen as well as alarms will be raised to alert security guards so that they can initiate rescue operations.

Future Scope:

To get better image processing and drowning detection accuracy, modern methodologies can be used to neglect the blurring of images due to the reflection of light by water. After its implementation, this system also can be used on sea beach for drowning detection.

Chapter 10

References

- https://link.springer.com/content/pdf/10.1007/3-540-47979-1_20.pdf

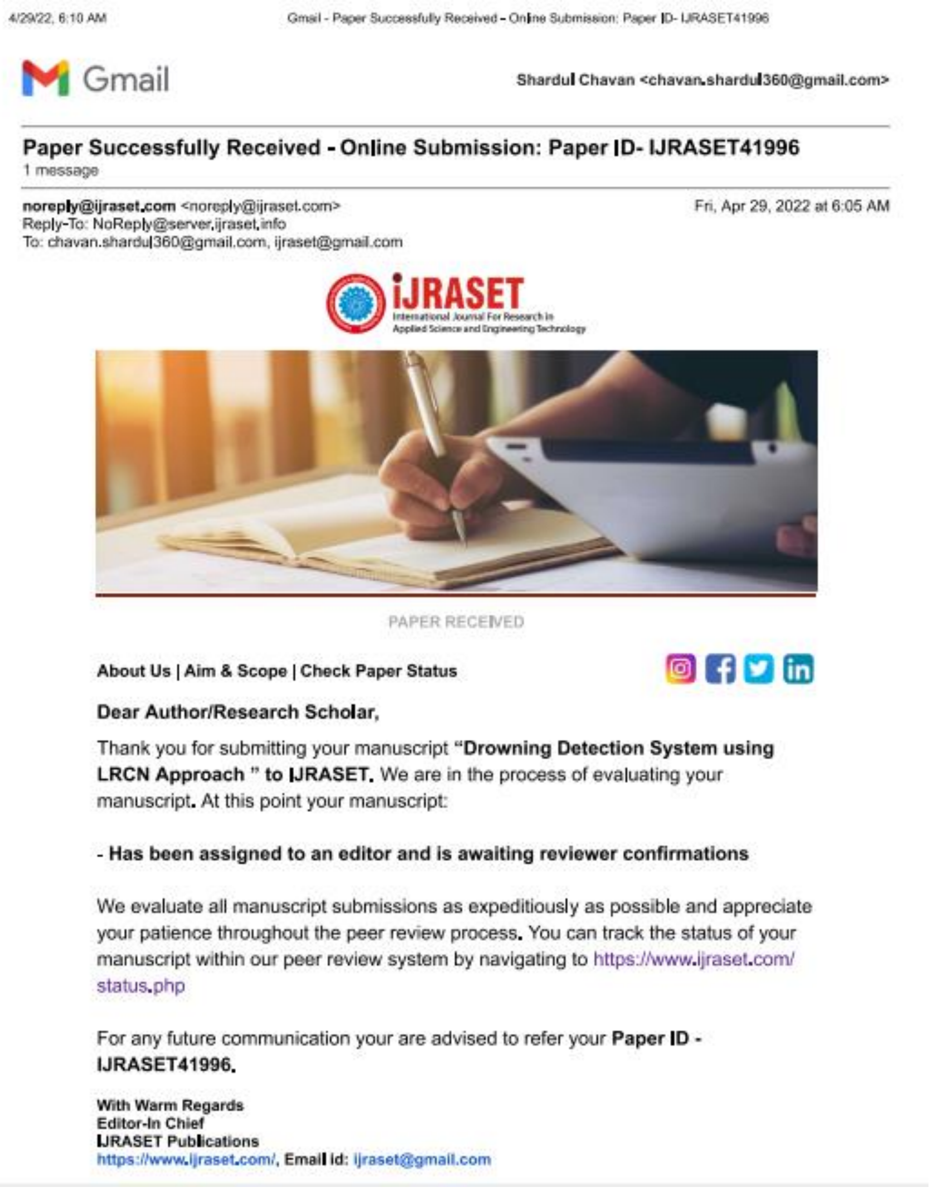
- https://scholar.archive.org/work/k6feepj5z5dfxmwar32flg2jqq/access/wayback/http://www.es.sdu.edu.cn/project/doc/papers/icess09-papers/15-1.pdf

- https://www.angeleye.tech/en/

- https://kobiso.github.io/research/research-lrcn/

<u>Chapter 11</u>
Publications
Drowning Detection System

Shardul Chavan, Sanket Dhake, Shubham Jadhav
Prof. Johnson Mathew
Department of Computer Engineering
Datta Meghe College of Engineering, Navi Mumbai



4/29/22, 6:10 AM Gmail - Paper Successfully Received - Online Submission: Paper ID- IJRASET41996

Gmail Shardul Chavan <chavan.shardul360@gmail.com>

Paper Successfully Received - Online Submission: Paper ID- IJRASET41996
1 message

noreply@ijraset.com <noreply@ijraset.com> Fri, Apr 29, 2022 at 6:05 AM
Reply-To: NoReply@server.ijraset.info
To: chavan.shardul360@gmail.com, ijraset@gmail.com

IJRASET
International Journal For Research in Applied Science and Engineering Technology

PAPER RECEIVED

About Us | Aim & Scope | Check Paper Status

Dear Author/Research Scholar,

Thank you for submitting your manuscript **"Drowning Detection System using LRCN Approach " to IJRASET.** We are in the process of evaluating your manuscript. At this point your manuscript:

- Has been assigned to an editor and is awaiting reviewer confirmations

We evaluate all manuscript submissions as expeditiously as possible and appreciate your patience throughout the peer review process. You can track the status of your manuscript within our peer review system by navigating to https://www.ijraset.com/status.php

For any future communication your are advised to refer your **Paper ID - IJRASET41996.**

With Warm Regards
Editor-In Chief
IJRASET Publications
https://www.ijraset.com/, **Email id:** ijraset@gmail.com