**Haar cascade**

Haar cascade is a machine learning based object detection algorithm. Initially it uses many positive and negative images and then uses them to train feature functions. The trained model is then ready to detect and localize objects in new unseen images. The number of features used for finding a complex object (a human, for example) is very large. In order to save computational time and resources, the features are grouped in various stages (cascaded) and are applied on images one after another. The most distinguishing features are checked in the beginning stages and the window goes to next stages only when a possibility of finding an object is found. Otherwise the window is discarded and the next window is checked. Its only when a window satisfies all the stages of the cascade, the object is said to be localized and is marked with a bounding box.

In this project a Haar Cascade model was trained to localize the position of human in the video frame. The frame is then sent to a Convolutional Neural Network (CNN), both of which have been trained using a custom image dataset. This method is divided into various steps which include:

1. Creating the dataset
2. Training the Haar Cascade model
3. Constructing and training the CNN model
4. Testing the performance of the model on images and video clips of falling

## 3.1 Collecting the image dataset for training Haar Cascade and CNN models

The images for the dataset were extracted from the frames of video clips of falling and non-falling instances. A Mi A3 smartphone’s wide angle camera was used to capture the videos. The camera has a resolution of 1920×1080 pixels and the video was taken at a frame rate of 30 frames per second. The frames were extracted from these videos using the OpenCV library functions.

During the extraction of these frames, the images are cropped to remove the portions of the frame which are not needed to detect a fall. For example, a person lying on a bed is assumed to be sleeping and hence needs not be considered by the fall detection algorithm. Also, the cropped frame has a dimension of 960×650 pixels. This is resized by a factor of 0.2 to convert it into 192×130 pixels size. This makes the processing of these images faster and easier for the algorithms.

A total of 389 images were extracted from the video of duration 10 minutes and 49 seconds. Out of these 389 images, 270 images were selected for the dataset which showed distinct poses of human (fallen or not fallen) while the rest were discarded because they did not represent any relevant features.

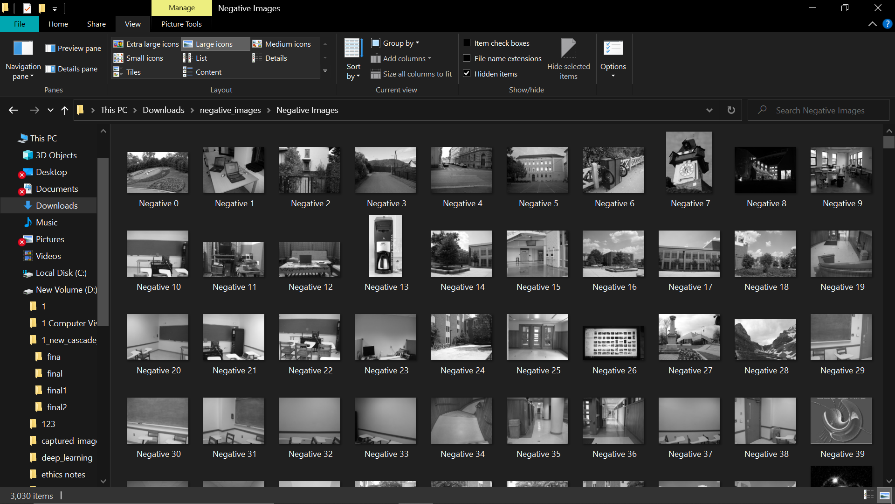
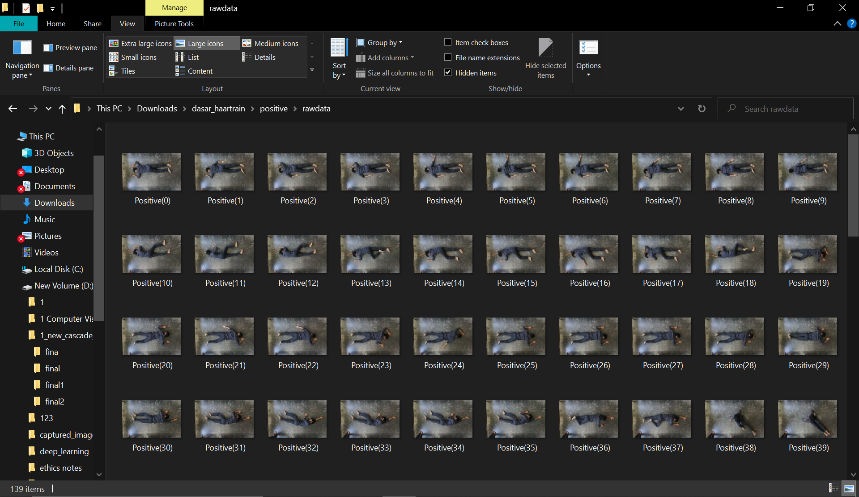
## 3.2 Training the Haar Cascade

A Haar Cascade model was trained using a tool that was taken from a YouTube channel-“Dasaradh Makes”. The ‘dasar\_haartrain.rar’ file was downloaded and its contents were extracted. It consists of various executable files to select the Region of Interest (ROI), to create samples, to train the haar cascade and other necessary functions. The steps used in training the cascade using this tool are illustrated below:

Step 1:

Training the haar cascade requires two categories of images- ‘Positive’ and ‘Negative’. The positive images are those ones in which there is a human in the frame. 139 such images are taken from our image dataset. The positive images must be in .BMP format. The negative images comprises of anything arbitrary which does not have a human. We took 1200 grayscale images from Kaggle Datasets negative images.

Figure [6] (a) Negative images, Kaggle Dataset (b) Positive images



(a)

(b)

Step 2:

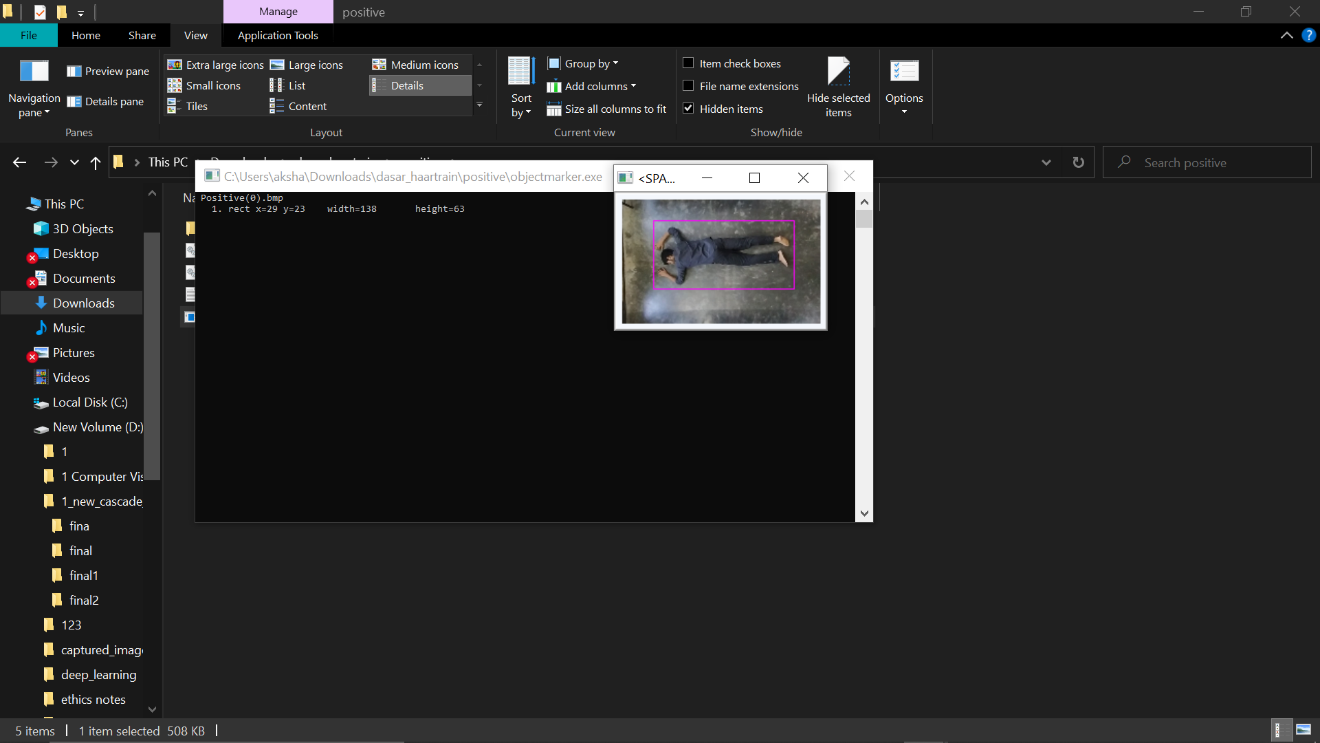
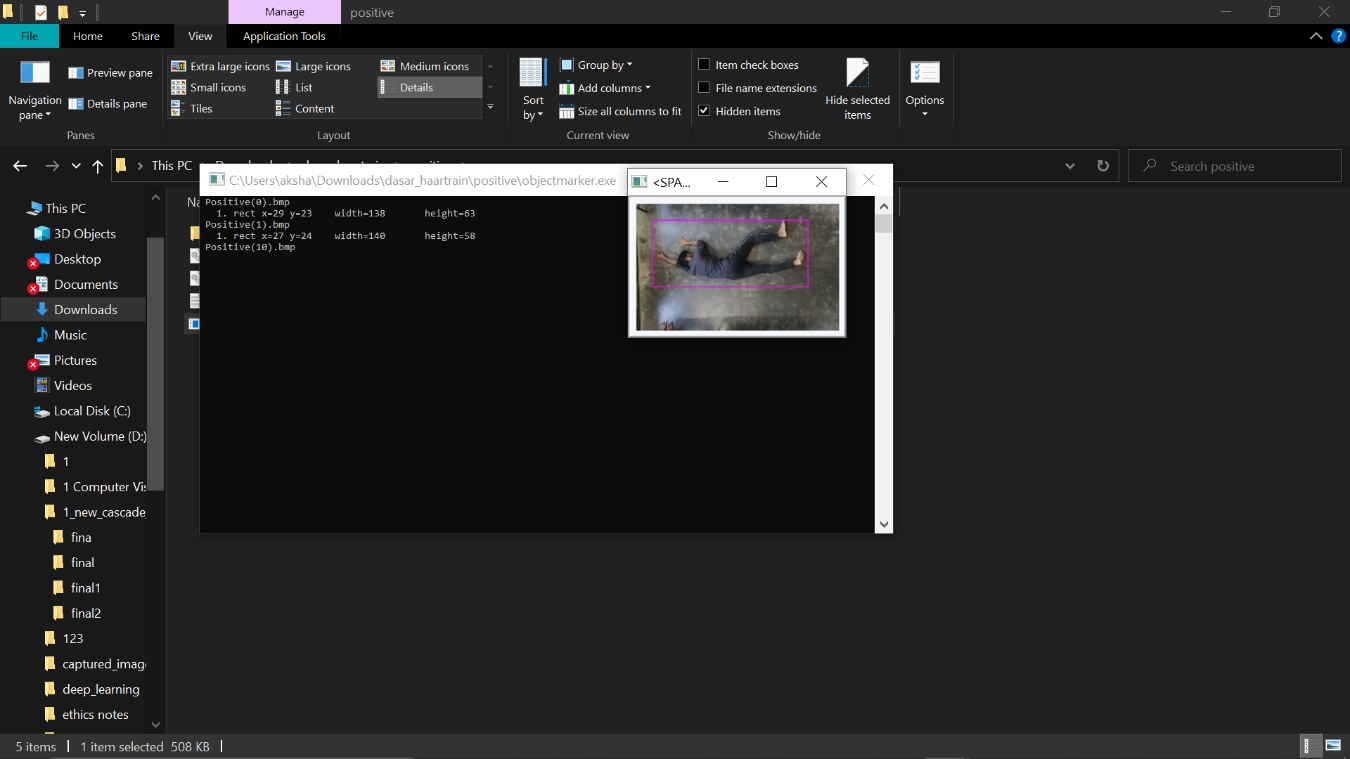
We need to select the region of interest (ROI) for each of the positive images using the application ‘objectmarker’. We have trained the cascade to detect full human body. For this, we need to enclose the human body in each image by dragging the cursor over the region. This gives the coordinates of the bounding box representing human in each image.

Fig 7: training haar cascade model

(b)

(a)

Step 3:

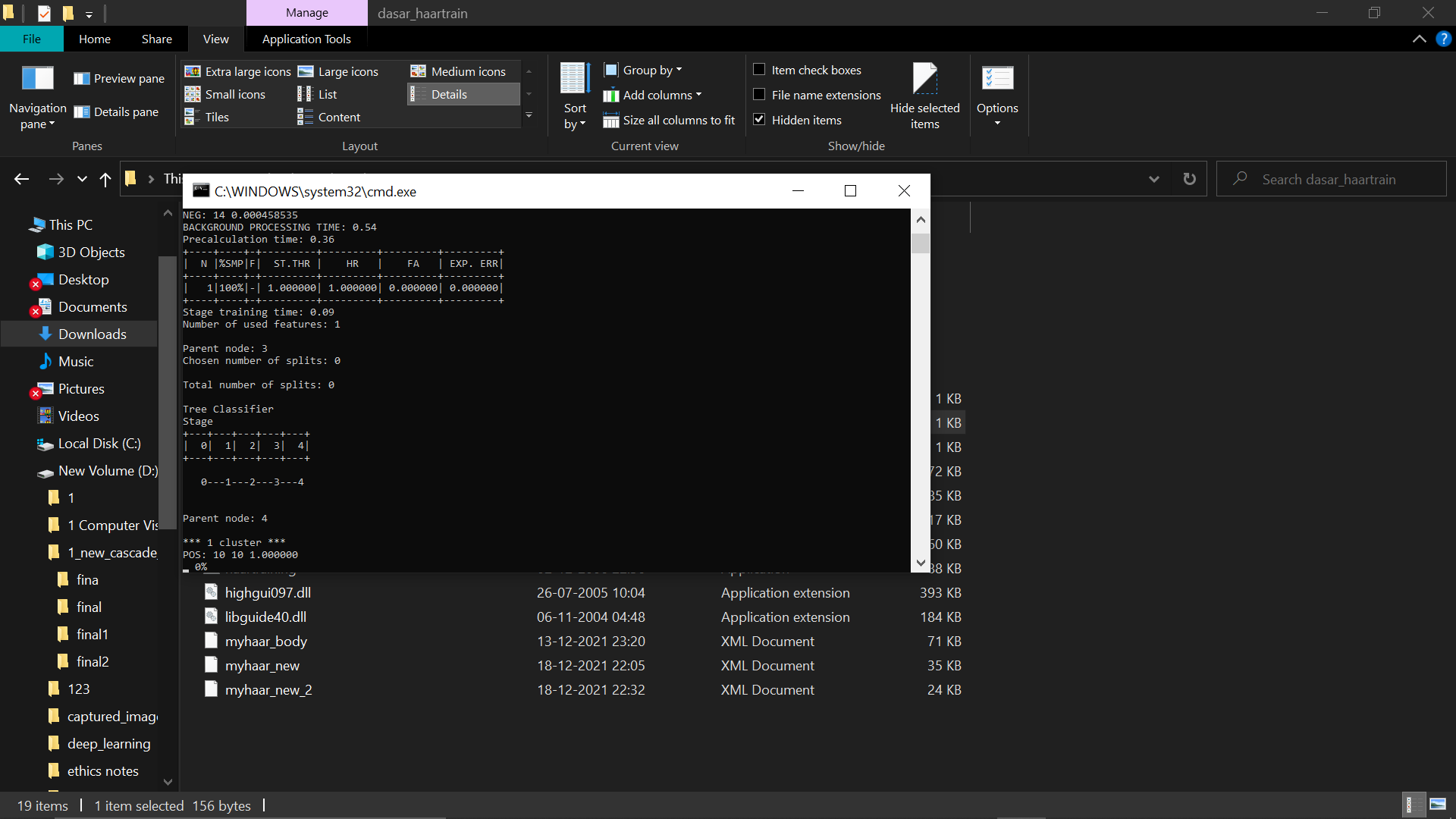
‘01 samples\_creation’ application is run which prepares the images to be used for training. Then the cascade is trained using the ‘02 haarTraining’ application. This application searches the images for all types of features and a set of features which best represents human body are selected. Multiple stages of haar cascade are trained depending on the quantity and complexity of the training images.

Fig 8: Cascade trainer application

After training is complete, a file called ‘myhaar.xml’ is created which contains the detailed parameters of all the stages of the Haar Cascade. This file will be required later on in the python program to detect and locate the human in a frame. Hence, the training of Haar Cascade is complete.

**Results:**

Although Haar cascades are very simple to train and give fast detections, they have a major drawback of being less accurate especially for complex objects like human bodies. It means that Haar cascades lead to miss object of interest as well as detect undesired objects. It can be seen from the results which were obtained with a cascade classifier trained to detect full human bodies.

Not only they are missing the full bodies (by detecting only heads) but are also detect leg as full body which is completely incorrect. Due to these reasons, haar cascade classifier was found to be unreliable and hence, unsuitable for this project.

## 

## 3.3 Convolutional Neural Network (CNN)

The CNN model consists of a number of layers including input layer, convolutional layers, batch normalization layers, maxpooling layers, fully-connected layers, and output layers. We tested different CNN modules including VGG-19, Xception, AlexNet. We also designed some CNN modules ourselves by adding various layers from the Tensorflow library. We got the best prediction accuracy using the Xception CNN module. Xception uses depthwise seperable convolutional layers in addition to the regular layers used in other CNN models. So, we have used the Xception module for our Fall Detection System. The module can be divided into three parts- entry flow, middle flow and exit flow.

This convolutional base is followed by a flattening layer and some fully-connected layers. This gives the complete CNN model. The flattening layer converts the input image with dimensions (200,200,3) into a single long 1-dimensional feature vector that can be given to the fully-connected layers. The fully connected layers include two dense layers with 200 and 100 neurons, respectively, followed by a single neuron in the output layer for binary classification.

The image dataset for training the CNN model consists of total 270 images. Out of these, 135 represent falls and the other 135 represent no-falls. The dataset is divided into sub-categories for the training process:

Fig 10: Dataset split into train, validation and test

These images are given to Tensorflow’s Image data generator module. This module outputs the given set of images with a large number of variations like rescaling, rotating, height or width shifting, shearing, flipping and zooming. This makes the model more accurate and robust and also tends to prevent overfitting. The images are normalized by dividing all intensity values by 255. Train\_generator, Validation\_generator and Test\_generator create such sets of images for train, validation and test purposes, respectively. Class\_mode is taken as ‘binary’ because our model does binary classification i.e. fallen and not-fallen.

The Xception CNN module is imported from tensorflow.keras.applications with the parameters from ImageNet. The input shape is set as (200,200,3) which is same as that of our input images.

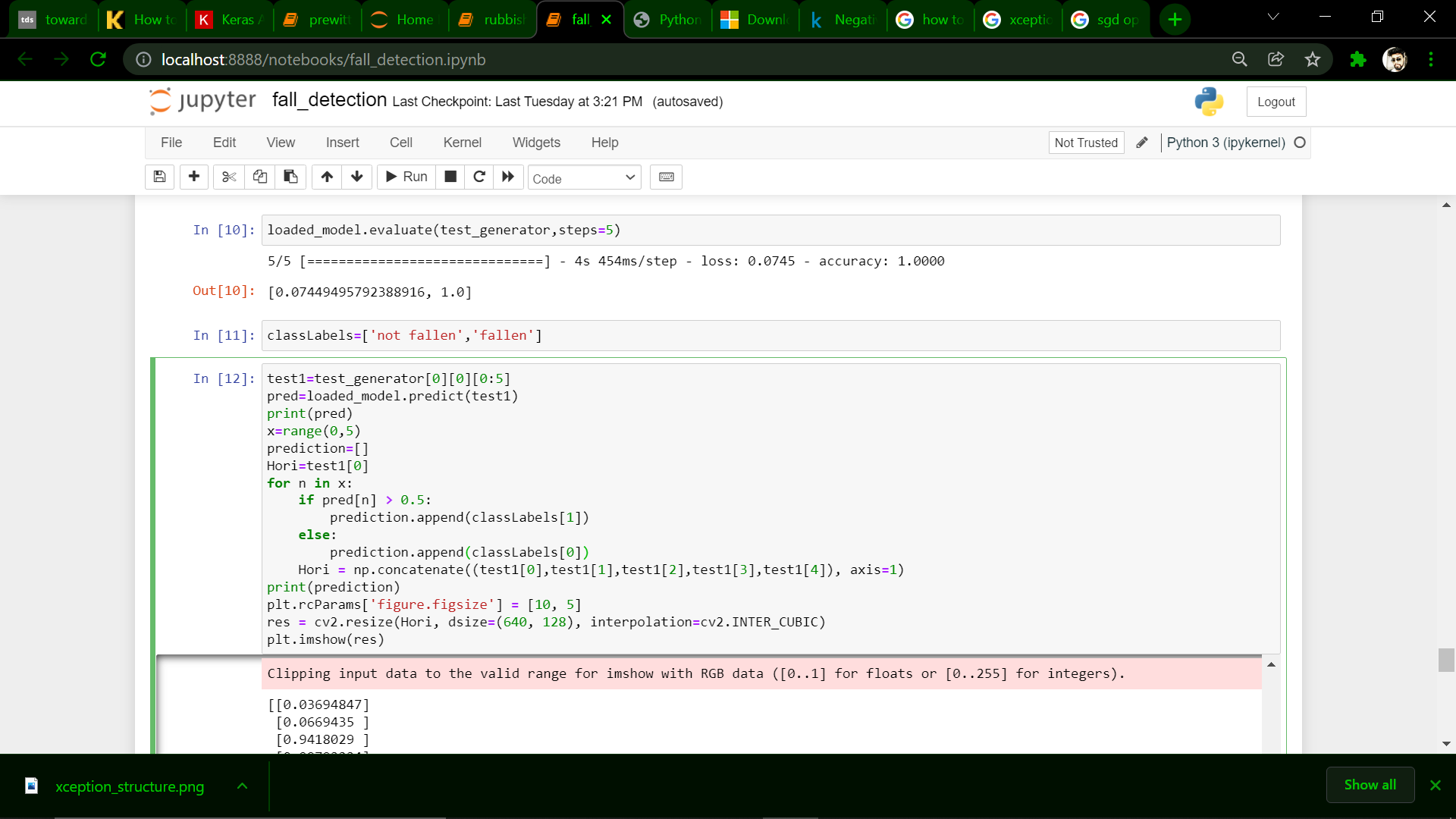
The layers of CNN model are added one by one. First of all, the convolutional base (conv\_base) is added. conv\_base is followed by a flattening layer the flattening layer is followed to two fully connected layers. The first and second hidden layers have 200 and 100 neurons, respectively. Rectified Linear Unit (ReLU) is used taken as the activation for both the layers. The output layer consists of a single neuron with sigmoid activation because it gives binary output.

Next, the model is compiled. The loss type is given as ‘binary\_crossentropy’ and Stochastic Gradient Descent (SGD) optimizer is used.

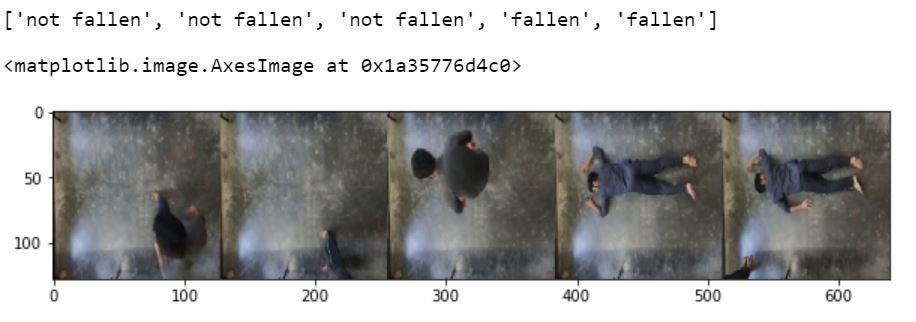
Finally, the model is training is done for 5 epochs which means that all the model training is done for 5 cycles. Also, each epoch is carried out in 19 steps because the batch size is 5 (95/5=19). Validation steps= 20/5=4. As it can be seen from the screenshot given below, the accuracy goes on increasing while the loss goes on decreasing with each epoch.

# **4. Results and Observation**

## 4.1 Accuracy

The accuracy of the model is determined using the test images which are unseen for the model. The accuracy on the test data was found to be almost 1 (100%) and loss was found to be 0.07449.

## 4.2 Testing

Then we used the model to predict the output for some of the test images. The CNN model gives a fraction between 0 and 1 as the output. The values close to 0 correspond to ‘not fallen’ and values close to 1 represent ‘fallen’. A threshold of 0.5 is taken to make the decision. The result of the prediction i.e. fallen or not fallen is given corresponding to each of the test case.

Next, we tested the predictor on a video clip which has instances of fall as well as of not fall. The result of prediction is written on the video frame itself. Here are examples of the predictions:

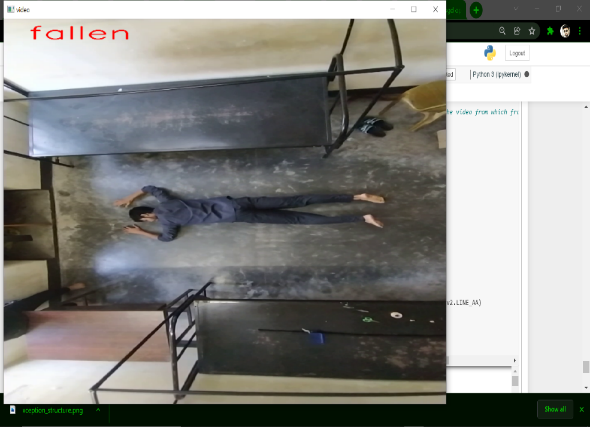


Fig 11: Results of the model for the test video

The classification of fallen and not fallen humans was done very accurately by the CNN model trained. However, it throws not light on the features of human body like orientation, angle, posture, aspect ratio, etc. To study the human body features which distinguish various postures including standing, sitting and falling, various techniques were used.