Stampede Detection

Using Multi-Column Convolutional Neural Networks

Hritvik Patel
PES1201700125
Computer Science and Engineering
PES University
Bangalore, India
hritvik.patel4@gmail.com

Shreyas BS
PES1201700956
Computer Science and Engineering
PES University
Bangalore, India
bsshreyas99@gmail.com

Archana P
PES1201701543
Computer Science and Engineering
PES University
Bangalore, India
arch.2421@gmail.com

***Abstract*—With this project we aim to build a deep network that would be able to predict the occurrence of a stampede. On compiling crowd images obtained from different sources and labelling them, we created our dataset. The grayscale resized images were fed into a configuration of an auto-encoder and a multi-column CNN. The auto encoder was used to compress images without much information loss. The output of the auto-encoder was fed into the CNN which used different sized filters to detect features and thus classify the images.**

Keywords—MCNN, Auto-encoder, Stampede, Dataset, Filters, Scale Invariance

I. INTRODUCTION

In recent years, frequent crowd stampede accidents have posed great threats to public security. Based on the traditional video surveillance technology, making full use of computer vision, image processing, and other related technology, applications like detecting targets and recognising abnormal human behaviour, and risk assessment of the state of crowds in public places for timely warning have great potential solutions and to prevent disastrous situations like stampedes and their consequences.

In this paper, we propose a solution to detect a stampede using a combination of an Auto-encoder and a multi-column Convolutional Neural Network (MCNN).

II. DATASET

The dataset was compiled from multiple sources listed in the references, since it was not readily available. This dataset contains images of stampede and non-stampede situations. It consists of 1199 images of which 595 images are non-stampede and the remaining are stampede.

Our images were labelled using a csv file. The label of the image was either a 1 or a 0. If an image was labelled 1, it was a stampede image, else non-stampede. Since all our images were coloured, we had to convert it to grayscale using the OpenCV python library. This had to be done as having 3 channels in the input image, i.e, R,G,B, would prove to be heavy for the network in terms of computation. The images in our dataset were of various resolutions and had to be resized into a standard dimension. We chose the dimensions of the resized image to be 100 x 100.

III. ARCHITECTURE

Auto-encoder

The grayscale input images of size 100 x 100 were fed into the auto-encoder. These input images were also provided as the target for the auto-encoder, as it had to learn how to reconstruct the image after its compression. The architecture was as follows:

The input layer of the auto-encoder consisted of 10000 neurons, corresponding to the flattened input array for an image.

The first hidden layer had 5000 neurons, followed by another hidden layer of 2500 neurons. As depicted by Fig.1, the following layers each consisted of 2500,5000 and 10000 neurons respectively. The activation unit used for each neuron was ReLU as the data consisted of images for which the pixel values would not be negative. As shown in Fig.1, the Encoder consisted of the input layer along with the hidden-layer1, hidden-layer2 and hidden-layer3, while the Decoder consisted of hidden-layer4 and the output layer.

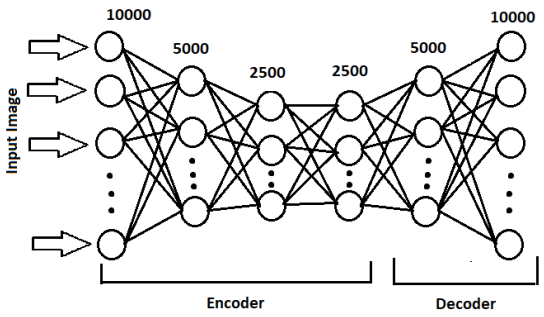

Fig.1

The output of the encoder (hidden-layer3) were compressed images of size (50 x 50). These were fed into the Convolutional Neural Network as input. The loss used for the auto-encoder was MSLE(Mean Squared Logarithmic Error). As the range of the pixel values in an image(target values) was high, in order to avoid penalising large errors more than small ones, we chose MSLE as our loss function.

Convolutional Neural Network

The input layer corresponding to the output from the trained encoder consisted of 2500 neurons. Our MCNN was inspired by the architecture used in [1].
Each input image was reshaped into dimensions of (50 x 50 x 1). A multi-column CNN (MCNN), as depicted in Fig.2, was employed to recognise features in the input image and classify the image as stampede or non-stampede. As our dataset consisted of images which were clicked from different distances, they had human faces of different sizes. Hence we used three columns, each with a different filter size. This made our model scale invariant and helped it learn better contrary to the vanilla CNN.

The architecture of the first column is as described:
As a part of the first convolutional layer, we applied 16 filters each of size 9 x 9 on the input image, which gave us 16 feature maps of the same size as the input image. This was ensured by padding the input image.
These feature maps were passed through a max pool layer which employed a 2 x 2 kernel. The dimensions of the feature maps were reduced by half.
The next three convolutional layers each applied filters of size 7 x 7 on the output of the previous layers and produced 32, 16 and 8 feature maps respectively.
Column 2 had an architecture similar to column 1, with the filter sizes as 7 x 7, 5 x 5, 5 x 5, 5 x 5 and the number of activation maps produced were 20, 40, 20, 10 respectively for the convolutional layers.
Column 3 also had an architecture similar to column 1, with the filter sizes as 5 x 5, 2 x 2, 2 x 2, 3 x 3 and the number of activation maps produced were 24, 48, 24, 12 respectively for the convolutional layers.
Thus the final convolutional layer of the three columns produced three sets of activation maps, described as follows:

1. Column 1:
   Number of feature maps: 8
   Size of feature map: 25 x 25
2. Column 2:
   Number of feature maps: 10
   Size of feature map: 25 x 25
3. Column 3:
   Number of feature maps:12
   Size of feature map: 25 x 25

These sets of feature maps were concatenated to give us a final set of 30 feature maps of size 25 x 25.

These activation maps were each then fed into a network of dense layers:

First dense layer: 18750 neurons
Second dense layer: 1875 neurons
Third dense layer: 187 neurons

The outputs of the third dense layer were fed into a final output neuron with sigmoid activation, which was used to classify the images based on a threshold of 0.5. The output values above 0.5 were classified as 1(stampede) and those below 0.5 were classified as 0 (non-stampede). The network was trained with sigmoid cross entropy as the loss function.

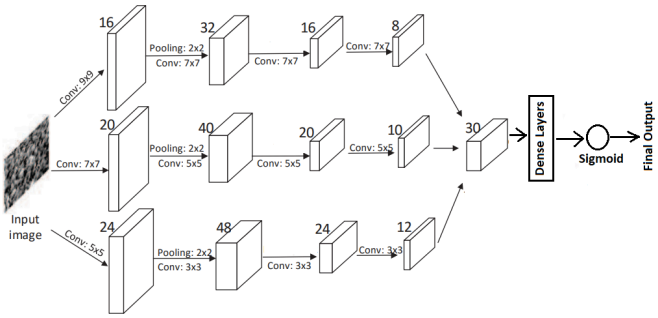

Fig.2

IV. RESULTS

The MCNN and the auto-encoder performed considerably well on the dataset. The dataset was shuffled 5 times, and the average loss of the auto-encoder was 0.47 and the average accuracy of the MCNN was 89.24%. The model's high accuracy can be attributed to its scale invariant property which resulted from the use of filters of varying sizes.
The results obtained have been visualised in Fig.3 and Fig.4.

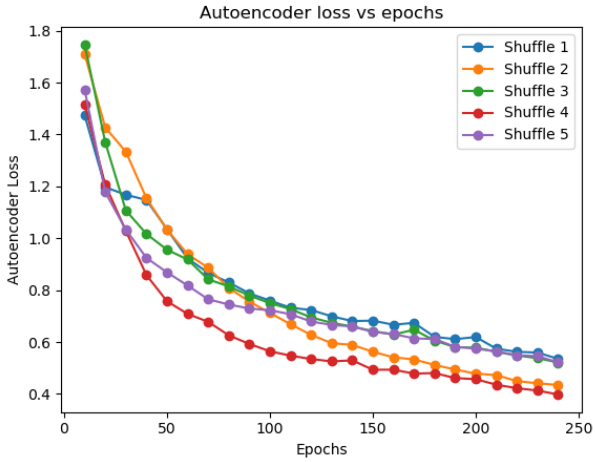

Fig.3

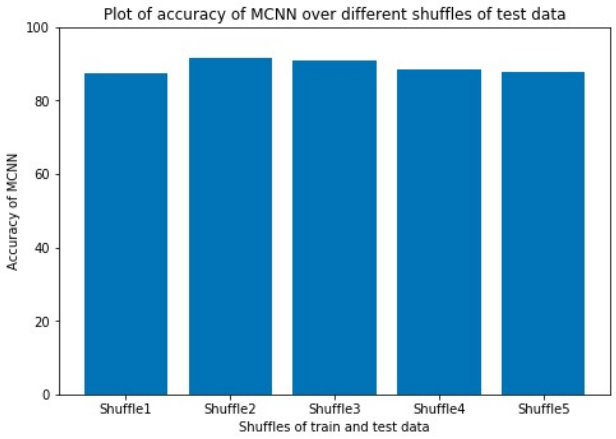

Fig.4

CONCLUSION

Thus, with the combination of an auto-encoder and a multi column CNN, we were able to able to build a scale invariant model that predicts the occurrence of a stampede. On its integration with appropriate hardware, this would make a complete system that can be used to prevent the loss of lives.

ACKNOWLEDGEMENT

We would like to extend our gratitude to our course professors Prof. Srikanth H.R for his constant guidance and support and Prof. Srinivas K.S for giving us this opportunity to explore the domain of deep learning with this project.

REFERENCES

1. https://zpascal.net/cvpr2016/Zhang_Single-Image_Crowd_Counting_CVPR_2016_paper.pdf
2. https://www.kaggle.com/danaelisanicolas/high-density-crowd-counting/data#
3. https://bgfons.com/uploads/crowd/?C=D;O=A
4. https://towardsdatascience.com/guide-to-coding-a-custom-convolutional-neural-network-in-tensorflow-bec694e36ad3
5. https://towardsdatascience.com/deep-autoencoders-using-tensorflow-c68f075fd1a3