# Crime Scene Prediction by Detecting Threatening Objects Using Convolutional Neural Network

Mohammad Nakib
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
nakibinbd@gmail.com

Md. Sakibul Hasan
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
sakibulhasanjitu@gmail.com

Rozin Tanvir Khan
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
rozin2345@gmail.com

Jia Uddin
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
engrjiauddin@gmail.com

***Abstract*—Crime scene prediction without human intervention can have outstanding impact on computer vision. In this paper, we present CNN (Convolutional Neural Network) in the use of detect knife, blood and gun from an image. Detecting these threatening objects from image can give us a prediction whether a crime occurred or not and from where the image is taken. We emphasized on the accuracy of detection so that it hardly gives us wrong alert to ensure efficient use of the system. This model use Rectified Linear Unit (ReLU), Convolutional Layer, Fully connected layer and dropout function of CNN to reach a result for the detection. We use Tensorflow, a open source platform to implement CNN to achieve our expected output. The proposed model achieves 90.2% accuracy for the tested dataset.**

***Keywords—Convolutional Neural Network, Crime Scene, TensorFlow, Object detection***

## I. Introduction

Nowadays, predicting a crime scene can ease the job of law enforcement agencies. As we can see there are a lot of CCTV cameras being installed to monitor a certain area. But, the increasing number of CCTV cameras is problematic too, as we need to monitor all the cameras manually. Life would be more easy if computers can predict crime scene by processing CCTV camera's video and the increasing demand of artificial intelligence (AI) in the security sector is a must. Previously a number of feature extraction methods are used in crime scene detection such as, HOG [8], SIFT [7], etc. In [4], a Multi scale convolutional network was used for classification. As Deep Neural Network (DNN) can give outstanding performance on image classification, in [6], the authors focused on the problems of object localization in the image. In [10], the authors use neural network for weapon detection. Automatic feature representation gives better performance than manual feature and unsupervised approach can give very close results to supervised approach [5]. In [11,14], the proposed algorithm can detect firearm and knife from image, where they use OpenCV.

In real life situation, even a single time is missed in detecting firearm or knife, it is possible that we may fail to save many important lives. Therefore, in this paper, we proposed an efficient method for detecting threatening objects using Convolutional Neural Network (CNN).

The rest of the paper is organized as follows: chapter II presents proposed model, Experimental results and analysis is presented in chapter III, and finally conclude in chapter IV.

## II. Proposed Model

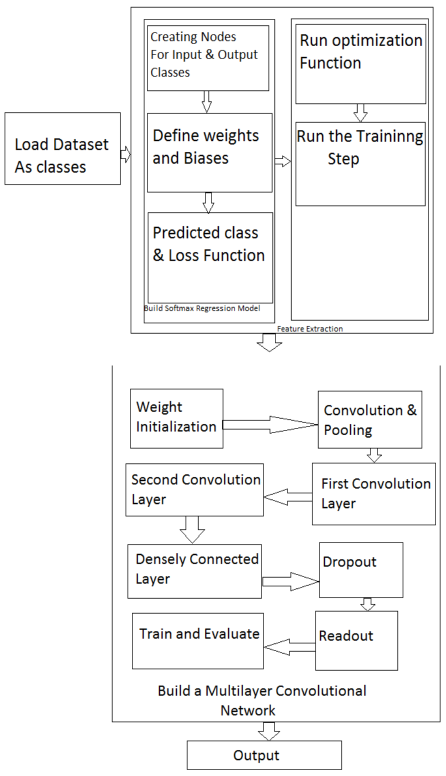

Fig. 1. Block Diagram of Model

A block diagram the model is given on Fig. 1, where feature extraction is a vital issue. The following sub-sections describe the different block diagram of the proposed model.

## *A. Load Dataset as Classes*

The dataset that we use for validating our model consists of knife, gun (short gun, revolver and machine gun) and blood. In Fig. 2, the sample dataset is given according to classes.

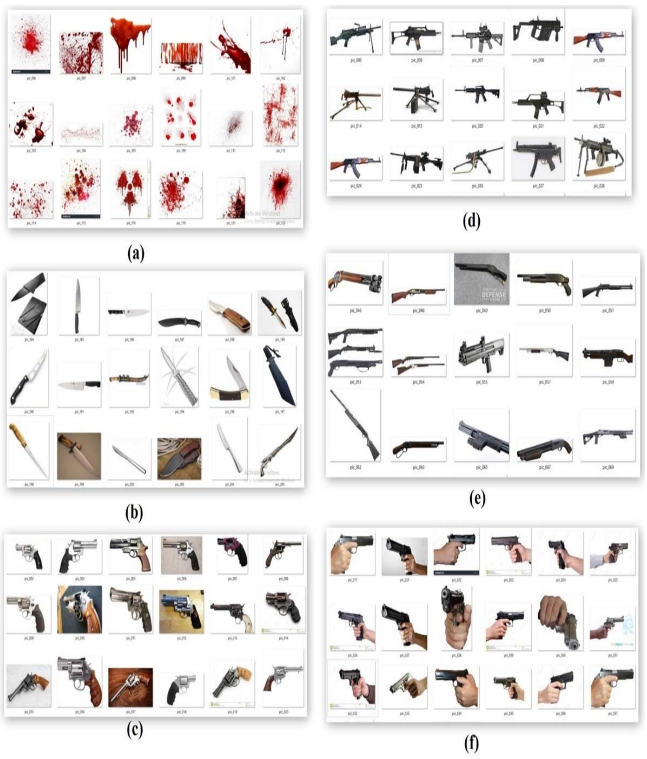

Fig. 2. Sample dataset of different classes (a) Blood (b) Knife (c) Revolver (d) Machine Gun (e) Shot Gun (f) Gun in hand

## *B. Feature Extraction*

To extract features from the images, we use some steps to generalize the images and turn the images into certain pattern. The main two steps of the feature extraction are "build softmax regression model" and "Training the model". These two steps include some sub steps.

*1) Build a Softmax Regression Model:* Initially we make a softmax regression model by making a single linear layer. The number of images in a batch changes in training step and it increases by 50 images. We have to define our weights W and biases b. For our input, the size of weights matrix is 1500×6, where image size is 150×100 pixel (dimension) and total number of class is 6. In addition, the size of bias matrix b is 1×6 because it only contains the biases of 6 classes only for our input. After creating placeholders and variables we now can execute the regression model.

It is the multiplications of the vectorized input images $X$ with weights $W$ and add the bias $b$ with the multiplication matrix (shown in eqn. 1).

$$Y = X.W + b \qquad (1)$$

In Equation 1, $Y$ indicates the prediction of images that is given as input as batches. We can specify the loss function which indicates how good the model's prediction is for the batch. Our target is to keep loss function as low as possible in our system to maintain the efficiency of detection.

In this paper, the loss function is the cross-entropy between the target and softmax activation function which is used for prediction for a batch of images. It finds the probability error within the classes that we make. It shows some problems for our detection system while classifying between revolver, machine gun and shot gun. There are some features matched easily with other classes while classifying between revolver, machine gun and shot gun. For example, every type of guns has similar features like trigger, barrel, etc

*2) Training Model:* In this part of the model, we describe the steps how the system is trained using the tools we made. We optimize our error function using gradient optimizer and then run the training steps.

Gradient descent optimizer is a useful function that reduces error function using a learning rate. In this model, gradient descent optimizer uses gradient descent algorithm. At the time of feeding data to the models it is always considered to lower the learning rate to slow the process for gaining better accuracy. We set the learning rate to 0.003 in Gradient Descent Optimizer. This function takes the error function and computes the partial derivative of the error function relative all the weights and biases. We repeat this for upcoming batches of training images and reduce the cross-entropy.

Returned operation from gradient descent optimizer function updates the parameter of training. Training has to be run repeatedly with the increment of 50 images in each batch than previous batch. We followed the order of batches of images as 50, 100, 150, 200, 250, 300, 350, 400, and 450. During the training it replaces the placeholders at software regression model. After this we evaluated the model and found the inefficiency in our model.

## *C. Build Multilayer Convolutional Model*

To build multilayer convolutional network from the model we have created so far, we need to follow some steps like weight initialization, convolution and pooling, first convolutional layer, second convolutional layer, densely connected layer, dropout, readout, train and evaluate.

Weight initialization is important as we work with a lot of weights and biases in the models. We initialize the weights with a small amount of noise for symmetry breaking and to avoid 0 gradients. The value of weights with small amount of noise we use is 0.1. Moreover, we use ReLu in our model. Therefore, to avoid "dead neurons", we use slightly positive initial bias. We initialize the bias with slightly positive values of 0.1. These initialization processes is done within a function.

Convolution and pooling operation executes the task of handling boundaries. We keep convolutions of the stride of 1 and are zero padded to maintain the output as like input. We use pooling over a 2×2 blocks in our input images. In the dataset, we have images of 150×100 pixels. We divide the every image with 2×2 block with stride of 1. As we use max pooling, we take maximum value of the block and keep it in the matrix. It helps to take the most weighted value so that we can take best value for predicting the shape more efficiently.

First convolutional layer computes 600 features for each 5×5 patch. For this layer weight variable will have a shape of 5, 5, 1, 600 in the parameter of making weight variable. In the shape the first two is patch size, then 1 is input channels and 600 is output channels. We kept limitation of 600 features as computation is always costly in terms of hardware, complexity and time. To execute the layer we need to reshape X with 150×100 and last parameter to reshape needs number of channels of color and in our case it is 1 as we are working on grayscale image. Then we use ReLu [2] for de-noising. An additional term, ReLu [2] is used in neural network operation.

It is an element wise operation and it is used pixel per pixel. ReLu [2] replaces all the negative pixel values by zeros so in that way it would become non-linear. There are also various nonlinear functions like sigmoid, tanh, etc. [2], but ReLu has outdone all the other methods in terms of de-noising. For an input 150×100 pixel, ReLu scan through the full pixels and denoises to the edges.

At the end of this layer, we use max pooling (2×2 block) over the images makes the size of original image to 75×50. Secondly, convolutional layer does the same thing as first convolutional layer. Still it has little bit of in terms of taking parameters to construct weight. Weight takes 5, 5, 600, 1200 as shape in the parameters of making variable. As mentioned above fist two element of parameters are patch number and here, input channels = 600, output channels =1200. Then, again we run ReLu and pooling like what we did in first convolutional layer. Image size becomes half after running max pooling again (37×25).

Fully connected layer mainly gives output of probabilities of the classes we have in our dataset. After pooling operation, this layer is executed with the image size of 37×25. We add fully connected layer with 2048 neurons to process on the whole image. Weight of fully connected layer takes shape as 37, 25, 1200, 2048 in the parameter of making variable and bias variable takes shape as 2048 in the parameter of making variable. Then we reshape images from the pooling layer to batch of vectors and multiply by weight matrix and add a bias. We apply this whole calculation as parameter to ReLu. Before getting the output as probability we apply dropout [12] to minimize overfitting.

When the data from the fully connected layer is fetched, and showed in the accuracy graph there is always some exponential difference when output data is driven away from the given input test and there creates a large gap called overfitting in the accuracy graph of the data. To reduce the difference, a function dropout is used. It flattens the images of the output value along size with input one resulting in much cleaner and accurate output. To implement dropout, we create a placeholder for probability on which the neuron's output is kept. We use the returned operation of ReLu from second fully connected layer and placeholder as parameters of dropout.

The last layer of this model is called readout layer. We make another variable of weight in this layer with the shape as 2048, 6 in the parameter. As mentioned above, we use 2048 to scan through the entire image and 6 because we have 6 classes. Again, a bias variable of taking shape 6 as parameter because of number of class we have in dataset. Then the retuned value of dropout function is multiplied with the weight variable created in readout layer and sums up with bias matrix and gives the prediction of multiple convolutional networks.

The last process we execute in our model is training and evaluating the model according to multiple convolutional networks. In this work, the output shows the percentage of every class matches with the trained dataset of our system. In Fig. 3, the result of analyzing a picture according to different class is given in percentage.

## III. Experimental Result and Analysis

The dataset information that we built for the system is given in Table I. We gathered this dataset for our system's purpose where we made necessary changes such as cropping the raw image using the bounding box parameter that was given, then we divided each firearm and their respective models into different folders (the labels for both training and testing images were given). After that the implementation of algorithm is done, here we have used TensorFlow [3], which is an interface for training machine learning algorithms, and also executing such algorithms. We have used dataset of guns, knives, machine guns, revolvers and shotguns to detect whether there is a Threatening scene is going to happen or not.

Table I. Dataset Information

| Testing Data | Training Data | |
|---|---|---|
| 400 | 394 | Blood |
| 400 | 396 | Knife |
| 400 | 426 | Machinegun |
| 400 | 482 | Revolver |
| 400 | 406 | Shotgun |
| 400 | 302 | Gun in hand |

Using our dataset for training the network for blood samples, we detected blood from various real life crime scenes. The problem here is, sometimes if it gets any color that is very similar color to blood, the surveillance camera may misinterpret it as blood, so there are some areas we can develop for more accuracy. Also, in most cases, it perfectly detects blood, with additional objects. In some cases, where there is blood with knife, it can detect both. As we have trained it for mass amount and at the same time limited amount of blood in a scenario, it has gained an accuracy of almost 90 to 97 percent (shown in Fig. 4), which can also change because of real life scenario.

At first, when we used 50-100 images as training sample, the accuracy lasted between 25 to 30 percent (shown in Fig. 4). But as we increased training data samples, after 200 images and 4000 iterations, the accuracy was between 90 to 99% (shown in Fig. 4).

| Training Set | Testing set | Result | Accuracy |
|---|---|---|---|
| | | Knife: 0.1%<br>Blood: 92%<br>Revolver: 0.1%<br>Machinegun: 0.07%<br>Gun in hand : 6.7% | **Perfectly Detected** |
| | | Knife: 0.05%<br>Revolver: 98%<br>Shotgun: 0.78%<br>Machinegun: 0.22%<br>Gun in hand : 0.6% | **Perfectly Detected** |
| | | Knife: 98%<br>Blood: 0.6%<br>Shotgun: 0.2%<br>Machinegun: 0.1%<br>Revolver: 0.002% | **Perfectly Detected** |
| | | Knife: 1.5%<br>Revolver: 2.7%<br>Shotgun: 18%<br>Machinegun: 28%<br>Gun in hand: 47% | **Detected with flaws** |

Fig. 3. Output (results shown in percentage)

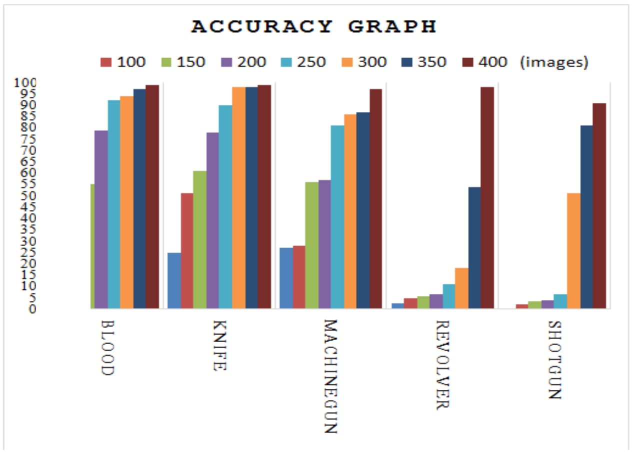

Fig. 4. Accuracy Graph

In case of training the network for knife samples, the graph in Fig. 4 shows good accuracy in detecting knife. The CNN successfully detects knife, because knifes shape is different from machine gun, revolver and shotguns. As we have trained it for different size, shape and angles in a scenario, it has gained an accuracy of almost 90 to 97 percent, which can also change because of real life scenario.

The main problem occurs in training revolver, machine gun and shotgun. When we first trained the CNN on machine gun, it was able to detect machine gun perfectly, because at that time we have not trained the CNN on revolver and shotguns, but as soon as we trained the CNN model on both revolvers and shotguns, it becomes difficult for the CNN to differentiate. So the graph in Fig. 4 shows first 50 to 200 sample images, as there are more than 100 models of different revolvers, shotguns and machine guns, it is not easy for the Model to correctly detect every class of firearms, as some models are pretty similar.

## IV. Conclusions

This paper presented a new approach to detect the blood, knife and gun, which help us to predict the crime scene occurred or not. The wrong alert is reduced that makes us our model very efficient for this task. TensorFlow is the best platform we found for this field. The proposed model using CNN demonstrates around 90.2% accuracy for the tested dataset.

## References

[1] Abadi, Martín, "TensorFlow: A system for large-scale machine learning," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, Georgia, USA. 2016.

[2] "An Intuitive Explanation Of Convolutional Neural Networks," The data science blog, 2017. [online]. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets

[3] "An open-source software library for Machine Intelligence," Web. 25 Dec. 2016. [online]. https://www.tensorflow.org/

[4] P. Sermanet and Y. Lecun, "Traffic sign recognition with multi-scale Convolutional Networks," International Joint Conference on Neural Networks, 2011.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[6] Erhan, Dumitru, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov, "Scalable Object Detection Using Deep Neural Networks," 2014 IEEE Conference on Computer Vision and Pattern Recognition (2014).

[7] D. Lowe, "Object recognition from local scale-invariant features," In ICCV, 1999.

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," In CVPR, 2005.

[9] Ali, Khawlah Hussein, and Tianjiang Wang, "Learning Features for Action Recognition and Identity with Deep Belief Networks," 2014 International Conference on Audio, Language and Image Processing (2014).

[10] O'reilly, Dean, Nicholas Bowring, and Stuart Harmer, "Signal Processing Techniques for Concealed Weapon Detection by Use of Neural Networks," 2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel (2012).

[11] Grega, Michael, Andrzej Matiolanski, Piotr Guzik, and Mikolaj Leszczuk, "Automated Detection of Firearms and Knives in a CCTV Image," Sensors vol.16, issue 1, 2016.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," J. Mach. Learn. Res. Vol. 15, issue 1 pp. 1929-1958, January 2014.

[13] Kingma, Diederik P., and Jimmy Ba., "Adam: A Method For Stochastic Optimization," 3rd International Conference for Learning Representations, San Diego, 2015 v1 (2014).

[14] S. Brahmbhatt, "Introduction to Computer Vision and OpenCV," Practical OpenCV, pp. 3–5, 2013.