



## Crime scene prediction and analysing its accuracy with frames using deep neural network

Devishree D. S.

[devishreeshivsaroj@gmail.com](mailto:devishreeshivsaroj@gmail.com)

S. J. C. Institute of Technology,  
Chikkaballapura, Karnataka

Divakar K. M.

[dnshgowda@gmail.com](mailto:dnshgowda@gmail.com)

S. J. C. Institute of Technology,  
Chikkaballapura, Karnataka

Ashini K. A.

[ashini.ka23@gmail.com](mailto:ashini.ka23@gmail.com)

S. J. C. Institute of Technology,  
Chikkaballapura, Karnataka

Arnav Singh Bhardwaj

[arnav94300@gmail.com](mailto:arnav94300@gmail.com)

S. J. C. Institute of Technology,  
Chikkaballapura, Karnataka

Sheikh Mohammad Younis

[shaykhyounis@gmail.com](mailto:shaykhyounis@gmail.com)

S. J. C. Institute of Technology,  
Chikkaballapura, Karnataka

### ABSTRACT

*Crime scene prediction without human intervention can have an outstanding impact on computer vision. In this paper, we present DNN in the use of a detect knife, blood and gun in order to reach a prediction whether a crime has occurred in a particular image. We emphasized on the accuracy of detection so that it hardly gives us the wrong alert to ensure efficient use of the system. This paper use Non-linearity ReLu, Convolutional Neural Layer, Fully connected layer and dropout function of DNN to reach a result for the detection. We use Tensorflow open source platform to implement NN to achieve our expected output. This system can achieve the test accuracy of 90.2 % for the datasets we have that is very much competitive with other systems for this particular task.*

**Keywords**— Crime scene, Neural Layer, Convolutional, Tensorflow

### 1. INTRODUCTION

Crime scene prediction from a camera is very important while working on a field of computer vision. In the modern era of science and technology, people setup surveillance cameras in different areas to get rid of crime. Still, it cannot help people as quick as people want to respond. Usually, after occurring a crime scene, law enforcement agencies come to the area and take the footage from the video that was recorded at the time of the crime scene. Then, law enforcement agencies analyze the video and take necessary evidence of a crime scene. We believe this is a very slow process to act on a crime scene. For this reason, we wanted to make a system that can quickly act on a crime scene. Moreover, there are a lot of cameras being installed in different areas by law enforcement agencies or by any organization. They have to monitor all the cameras at a time with a human being. If a computer system can detect the threatening objects and give alert to the authority just after detection, the proper authority can quickly take action to stop the potential criminal before he

commits any crime. For example, 1st July of 2016, an incident occurred in Dhaka in a restaurant.

Terrorist went to the restaurant with guns, hand grenades and a knife etc. But initially, law enforcement agencies did not understand how much dangerous the terrorists are. If the camera installed over there can give information to law enforcement agencies by any media (IP camera or control from police station etc.) just after exposing the weapons, law enforcement agencies can respond to the scene very quickly and may save important lives. This incident helped us to think more deeply to make a system that can be learned to detect threatening objects. In our paper, we worked on detecting revolver, machine gun, shot gun, blood and knife using a convolutional neural network.

### 2. PROPOSED CONVOLUTIONAL NEURAL NETWORK MODEL

#### 2.1 Introduction

As firearms, knife and blood detection from an image are our objectives, we propose a simple block diagram using CNN which focused on ensuring accuracy rate as high as possible. In Tensor Flow platform, this model can identify almost all the input data with high accuracy. A block diagram the model is given in Fig1.

#### 2.2 Load dataset as classes

We want to detect knife, gun (short gun, revolver and machine gun) and blood. We make dataset which has 5 classes as input images. Every detection object has to have a single class of image in the dataset. We make the datasets using the image of 150x100 pixels (dimension). We call multiple images as a batch. In this paper, batches of images are taken to train the system and shows output for a particular input. Our system not only detects the objects it also shows the graph. Starting an interactive session can help to structure our system. It also helps to call computational graph to check how the system is working. Then

the images are sent to the next step for the feature extraction. In figure 2, the sample dataset is given according to classes.

### 2.3 Feature extraction

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

**2.3.1 Build a Softmax Regression Model:** Initially, we make a softmax regression model by making a single linear layer. But we change this single linear layer to multilayer convolutional layers. To extract the feature from input images, we need to take some variables for computations.

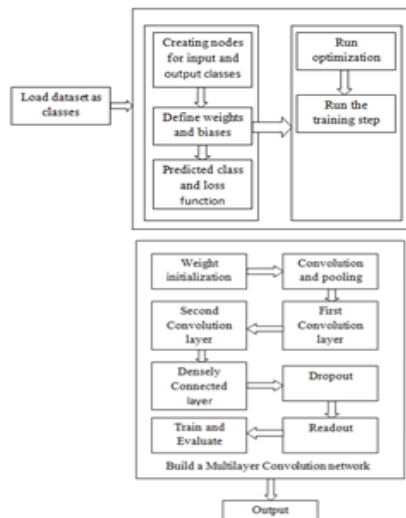


Fig. 1: Block diagram of proposed model

As mentioned above that our images which has  $150 \times 100$  pixels (dimension). At first, we turn the whole image into a single linear array and our first batch of images contains 50 images. Therefore, we make a placeholder, X which has size 1500 columns (as  $150 \times 100 = 15000$ ) and 50 rows (as 50 images for the first batch). Every row contains the values of a single image.



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

We also make a placeholder which contains information about classes. The size of this matrix is  $50 \times 6$ . As we have 5 classes

for 50 images first. 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 the weights matrix is  $1500 \times 6$ . As the dimension of our images in datasets is  $150 \times 100$  pixel (dimension) and the class number is 6. Moreover, the size of bias matrix b is  $1 \times 6$  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 adds the bias b with the multiplication matrix.

$$Y = X \cdot W + b$$

Here, 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 through 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 was used for prediction for a batch of images. It finds the probability error within the classes we make. It shows some problems for our detection system as the system can classify between a revolver, machine gun and shot gun. It arises problems too. There are some features matched with other classes. While calculating loss function, our system shows a high rate of loss function rate and sometimes totally misleading information while classifying gun types. But, in this paper, it is not a big issue as our purpose is only to detect a gun. No matter what kind of gun an image contains. This occurs only while in bad quality images of the gun but it can predict gun type perfectly for a batch of good quality images where cross-entropy is lower.

**2.3.2 Training model:** We defined our loss function so far. In the part of the model, we describe the steps of 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 a 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 the previous batch. We followed the order of batches of images as 50, 100, 150, 200, 250, 300, 350, 400, and 450. With every batch of training, it replaces the placeholders we created at software regression model. After this, we evaluated the model and found the inefficiency in our model.

### 2.4 Build Multilayer Convolutional Model

To build a multilayer convolutional network from the model we have created so far, we need to follow some steps like 1. Weight Initialization 2. Convolution & Pooling 3. First Convolutional Layer 4. Second Convolutional Layer 5. Densely Connected Layer 6. Dropout 7. Readout 8. 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 a small amount of noise we

use is 0.1. Moreover, we use ReLu in our model. Therefore, to avoid “dead neurons”, we use a slightly positive initial bias. We initialize the bias with slightly positive values of 0.1. These initialization processes are 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  $2 \times 2$  blocks in our input images. Stride: number of pixels to slide our filter matrix over the input matrix Zero padding: padding the input matrix with zeroes. In the datasets, we have images of  $150 \times 100$ . We divide every image with  $2 \times 2$  blocks with a stride of 1. As we use max pooling we take the maximum value of the block and keep in the matrix. It helps to take the most weighted value so that we can take the best value to predict the shape more efficiently. First convolutional layer computes 600 features for each  $5 \times 5$  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  $150 \times 100$  and last parameter to reshape needs number of channels of color and in our case, it is 1 as we are working on the grayscale image. Then we use ReLu for denoising.

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 is also various nonlinear functions like sigmoid or tanh instead of ReLU [2], but ReLu [2] has outdone all the other methods in terms of denoising. For an input  $150 \times 100$  pixel ReLU Scan through the full pixels and denoises to the edges. At the end of this layer, we use max pooling ( $2 \times 2$  blocks) over the images makes the size of the original image to  $75 \times 50$ .

Second Convolutional layer does the same thing as the first convolutional layer. Still, it has a little bit of in terms of taking parameters to construct weight. Weight takes 5, 5, 600, 1200 as a shape in the parameters of making variable. As mentioned above first two elements of parameters are patch number and here, input channels = 600, output channels = 1200. Then, again we run ReLu and pooling like first convolutional layer. Image size becomes half than before after running max pooling again ( $37 \times 25$ ).

Fully connected layer mainly gives an output of probabilities of the classes we have in our dataset. After pooling operation, this layer is executed with the image size of  $37 \times 25$ . We add a fully connected layer with 2048 neurons to process on the whole image. The weight of the 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 a batch of vectors and multiply by weight matrix and add a bias. We apply this whole calculation as a parameter to ReLu [2]. Before getting the output as probability we apply dropout [18] 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 [18] 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 [18], we create a placeholder for probability on which the neuron's output is kept. We use the returned operation of ReLu from the second fully connected layer and placeholder as parameters of dropout. The last layer of this model is called the 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 biasing variable of taking shape 6 as a parameter because of a number of class we have in the dataset. Then the returned value of dropout function is multiplied with the weight variable created in the readout layer and sums up with the 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. This process is as identical as we did in our model before to train simple software regression model. But there are few changes here than before:

- We use ADAM optimizer [19] in this part which is more sophisticated than Gradient descent optimizer.
- We keep a variable in parameters training to control dropout.
- We keep logging of every 50th repeated task of training to keep information of result.

### 3. EXPERIMENTAL SETUP AND RESULT ANALYSIS

A new method based on deep learning techniques for Crime detection in video surveillance cameras. The proposed method has been evaluated in the UCSD dataset and showed an increase in the accuracy of Crime Detection.

**Table 1: 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

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 the algorithm is done, here we have used TensorFlow, which is an interface for training machine learning algorithms, and also executing such algorithms. It is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. We have used Datasets of guns, knives, machineguns, revolvers and shotguns to detect whether there is a Threatening scene is going to happen or not.

The steps we followed are:

**Step 1:** Setting up Tensorflow.

**Step 2:** Creating a set of training images.

**Step 3:** Run the virtual machine.

**Step 4:** Link the images virtually.

**Step 5:** Update the code.

**Step 6:** Compile the code.

**Step 7:** Calculate train accuracy.



**Step 8:** Calculate validation accuracy.

**Step 9:** Calculate cross entropy.

**Step 10:** use a test sample to see the result.

### 3.1 Setting up tensorflow

TensorFlow is an interface for training machine-learning algorithms, and performing such algorithms. It is an open source software library for numerical computation using data flow graphs. A computation done using TensorFlow can be performed on a wide variety of systems, ranging from mobile devices up to large-scale distributed systems. The system can be used to express a wide variety of algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems. Tensorflow can be set up by two processes, for GPU support or CPU support. TensorFlow programs typically run significantly faster on a GPU than on a CPU. Unfortunately, as we don't have a high-level GPU, we run sensor flow on a CPU based system.

### 3.2 Creating a set of the training dataset

The dataset used is obtained from the Image net dataset. The dataset has been generated by supplying pictures from search engines like Google, Bing, etc. were noise-free pictures are given out. Normal data could have been used but those would have had a high level of noise margins which we have avoided for our training purposes in our convolutional models. For training to work well, we need to gather at least a hundred photos of each kind of object we want to recognize. In the whole of the dataset, 394 types of a blood sample, 302 guns in hand sample, 396 knife samples, 426 machinegun samples, 482 revolver samples, 406 shotgun samples.

### 3.3 Training





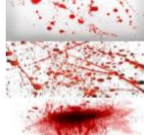



Once the bottlenecks are complete, the actual training of the top layer of the network begins. In that time a series of step outputs, each one showing training accuracy, validation accuracy, and the cross entropy. The training accuracy shows what per cent of the images used in the current training batch was labelled with the correct class. The validation accuracy is the precision of a randomly-selected group of images from a different set. Cross-entropy is a loss function which gives a glimpse into how well the learning process is progressing. The key difference is that the training accuracy is based on images that the network has been able to learn from so the network can over fit to the noise in the training data. A true measure of the performance of the network is to measure its performance on a data set not contained in the training data -- this is measured by the validation accuracy. If the training accuracy is high but the validation accuracy remains low that means the network is over fitting and memorizing particular features in the training images that aren't helpful more generally. The training's objective is to make the loss as small as possible. There will be an accuracy value of between 90% and 95%, though the exact value will vary from run to run since there's randomness in the training process.

### 3.4 Distortions

A common way of improving the results of image training is by deforming, cropping, or brightening the training inputs in random ways. We enable these distortions by passing random crop, --random scale and random brightness to the script. These are all percentage values that control how much of each of the distortions is applied to each image. It's reasonable to start with values of 5 or 10 for each of them and then experiment to see which of them help. flip\_left\_right will randomly mirror half of the images horizontally, which makes sense as long as those inversions are likely to happen in the application.

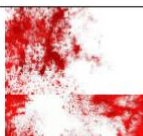



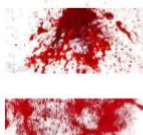



### 3.5 Accuracy measurement

From Fig 4, we can see that in most cases it successfully detects blood. In some cases, it gives fault. In a real-life scenario, the color, depth, anything can matter in detecting blood. So some mistakes will occur, as it is not fully prepared for any scenario.

Training Set	Testing set	Result	Accuracy
		Knife: 50% Blood: 29% Shotgun: 17% Machinegun: 0.3% Gun in hand : 1%	<b>Detected with Flaws</b>
		Knife: 0.1% Blood: 97% Gun in hand: 0.1% Machinegun: 0.8% Revolver: 0.7%	<b>Perfectly Detected</b>
		Knife: 3.3% Blood: 94% Shotgun: 0.5% Machinegun: 0.3% Gun in hand : 1.2%	<b>Perfectly Detected</b>
		Knife: 3.7% Blood: 31% Revolver: 10% Machinegun: 8.1% Gun in hand : 45%	<b>Detected with Flaws</b>

**Fig. 5: Blood sample testing 1**

From figure 5, we can see that in most cases it successfully detects blood. In some cases, it gives fault. In a real-life scenario, the color, depth, anything can matter in detecting blood. So some mistakes will occur, as it is not fully prepared for any scenario.

Training Set	Testing set	Result	Accuracy
		Knife: 0.1% Blood: 92% Revolver: 0.1% Machinegun: 0.07% Gun in hand : 6.7%	<b>Perfectly Detected</b>
		Knife: 41% Blood: 55% Shotgun: 0.9% Machinegun: 0.6% Gun in hand : 1.3%	<b>Detected With flaws</b>
		Knife: 10% Blood: 79% Revolver: 2.8% Machinegun: 4.4% Gun in hand : 1.7%	<b>Perfectly Detected</b>
		Knife: 0.2% Blood: 99% Revolver: 0.1% Machinegun: 0.06% Gun in hand : 0.07%	<b>Perfectly Detected</b>

**Fig. 6: Blood sample testing 2**

From figure 7, we can see that in most cases it successfully detects Revolver. In some cases, it gives fault. In real life scenario, machinegun, shotgun and revolver have many similarities in many cases according to models and size, so it is pretty much easy for the classifier to confuse among them, so some error occurs.

Training Set	Testing set	Result	Accuracy
		Knife: 0.9% Revolver: 5% Shotgun: 13% Machinegun: 47% Gun in hand : 32%	<b>Detected</b>
		Knife: 1.2% Revolver: 6.5% Shotgun: 38% Machinegun: 19% Gun in hand : 33%	<b>Detected</b>
		Knife: 0.2% Revolver: 11% Shotgun: 18% Machinegun: 5% Gun in hand : 64%	<b>Detected</b>
		Knife: 0.5% Revolver: 5.5% Shotgun: 50% Machinegun: 21% Gun in hand : 20%	<b>Detected</b>

Fig. 7: Revolver sample testing 1

From figure 8, we can see that in most cases it successfully detects Revolver. In some cases, it gives fault. In real life scenario, machinegun, shotgun and revolver have many similarities in many cases according to models and size, so it is pretty much easy for the classifier to confuse among them, so some error occurs.

Training Set	Testing set	Result	Accuracy
		Knife: 0.05% Revolver: 98% Shotgun: 0.78% Machinegun: 0.22% Gun in hand : 0.6%	<b>Perfectly Detected</b>
		Knife: .006% Revolver: 18% Shotgun: 19% Machinegun: 1.4% Gun in hand : 59%	<b>Detected</b>
		Knife: 8.8% Blood: 0.6% Shotgun: 9.8% Revolver: 54% Gun in hand : 25%	<b>Detected</b>
		Knife: 0.1% Revolver: 2.7% Shotgun: 85% Machinegun: 0.2% Gun in hand : 11%	<b>Not Detected</b>

Fig. 8: Revolver sample testing 2

From figure 9, we can see that in most cases it successfully detects Knife. In some cases, it gives fault. In real life scenario, the color, size, length and width anything can matter in detecting knife. So some mistakes will occur, as it is not fully prepared for any scenario.

From figure 10, we can see that in most cases it successfully detects Knife. In some cases, it gives fault. In real life scenario, the color, size, length and width anything can matter in detecting knife. So some mistakes will occur, as it is not fully prepared for any scenario.

Training Set	Testing set	Result	Accuracy
		Knife: 98% Blood: 0.6% Shotgun: 0.2% Machinegun: 0.1% Revolver: 0.002%	<b>Perfectly Detected</b>
		Knife: 90% Blood: 5.2% Shotgun: 1.5% Machinegun: 1.8% Revolve: 0.3%	<b>Perfectly Detected</b>
		Knife: 78% Blood: 8.2% Shotgun: 4.7% Machinegun: 6.1% Revolver: 2.1%	<b>Perfectly Detected</b>
		Knife: 25% Blood: 50% Shotgun: 2.8% Revolver: 0.78% Gun in hand : 19%	<b>Perfectly Detected</b>

Fig. 9: Knife sample testing 1

Training Set	Testing set	Result	Accuracy
		Knife: 61% Blood: 31% Shotgun: 1.4% Machinegun: 5.5% Revolver: 0.38%	<b>Perfectly detected</b>
		Knife: 98% Blood: 0.4% Shotgun: 0.1% Machinegun: 0.1% Revolver: 0.6%	<b>Perfectly detected</b>
		Knife: 99% Blood: 0.04% Shotgun: 0.005% Machinegun: 0.005% Revolver: 0.005%	<b>Perfectly detected</b>
		Knife: 51% Blood: 25% Shotgun: 5.2% Machinegun: 0.2% Gun in hand : 17%	<b>Perfectly detected</b>

Fig. 10: Knife Sample Testing 1

From figure 11, we can see that in most cases it successfully detects machinegun. In some cases, it gives fault. In real life scenario, machinegun, shotgun and revolver have many similarities in many cases according to models and size, so it is pretty much easy for the classifier to confuse among them, so some error occurs.

Training Set	Testing set	Result	Accuracy
		Knife: 0.4% Revolver: 0.4% Shotgun: 41% Machinegun: 56% Gun in hand : 0.8%	<b>Perfectly detected</b>
		Blood: 0.5% Revolver: 0.7% Shotgun: 10% Machinegun: 84% Gun in hand : 3.5%	<b>Perfectly detected</b>
		Knife: 1.5% Revolver: 2.7% Shotgun: 18% Machinegun: 28% Gun in hand : 47%	<b>Detected with flaws</b>

Fig. 11: Machinegun sample testing





Training Set	Testing set	Result	Accuracy
		Knife: 0.1% Revolver: 0.4% Shotgun: 10% Machinegun: 81% Gun in hand: 6.8%	Perfectly detected
		Knife: 0.3% Revolver: 0.2% Shotgun: 12% Machinegun: 86% Gun in hand: 0.3%	
		Knife: 0.2% Revolver: 0.08% Shotgun: 1.8% Machinegun: 97% Gun in hand: 0.03%	
		Knife: 2.1% Revolver: 1.3% Shotgun: 26% Machinegun: 27%	

Fig. 12: Machinegun sample testing 2

From this figure 12, we can see that in most cases it successfully detects machinegun. In some cases, it gives fault. In real life scenario, machinegun, shotgun and revolver have many similarities in many cases according to models and size, so it is pretty much easy for the classifier to confuse among them, so some error occurs.



Training Set	Testing set	Result	Accuracy
		Knife: 0.4% Revolver: 4.6% Shotgun: 31% Machinegun: 12% Gun in hand: 50%	Detected
		Knife: 0.6% Revolver: 0.03% Shotgun: 91% Machinegun: 1.9% Gun in hand: 5.6%	
		Knife: 17% Revolver: 12% Blood: 16% Machinegun: 22% Gun in hand: 20%	
		Knife: 5.8% Blood: 2.0% Shotgun: 51% Machinegun: 31%	

Fig. 13: Shotgun sample testing 1

From figure 13, we can see that in most cases it successfully detects shotgun. In some cases, it gives fault. In real life scenario, machinegun, shotgun and revolver have many similarities in many cases according to models and size, so it's pretty much easy for the classifier to confuse among them, so some error occurs.



Training Set	Testing set	Result	Accuracy
		Knife: 1.5% Revolver: 3.0% Shotgun: 6.7% Machinegun: 23% Gun in hand: 64%	Detected with flaws
		Blood: 3.3% Revolver: 3.7% Shotgun: 2.2% Machinegun: 81% Gun in hand: 7.0%	
		Blood: 3.3% Revolver: 10% Shotgun: 3.5% Machinegun: 72% Gun in hand: 6.1%	
		Knife: 0.19% Revolver: 3.1% Shotgun: 4.1% Machinegun: 90% Gun in hand: 4.7%	

Fig. 14: Shotgun sample testing 2

From this figure 14, we can see that in most cases it can't successfully detect shotgun. In most cases, it gives fault. In real life scenario, machinegun, shotgun and revolver have many similarities in many cases according to models and size, so it's pretty much easy for the classifier to confuse among them, so some error occurs.

#### 4. RESULT ANALYSIS

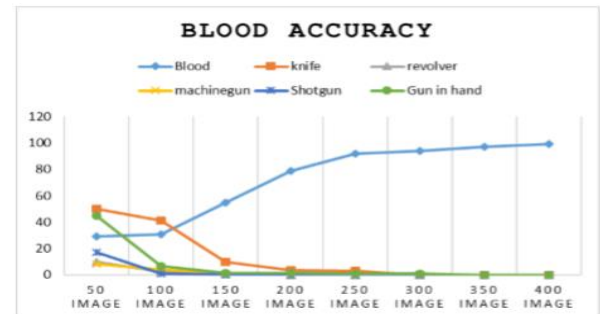


Fig. 15: Blood accuracy graph

From figure 15 we can see that, as we have trained it for an increasing number of blood samples, the accuracy percentage also increased from 29% to approximately 97% to 99%.

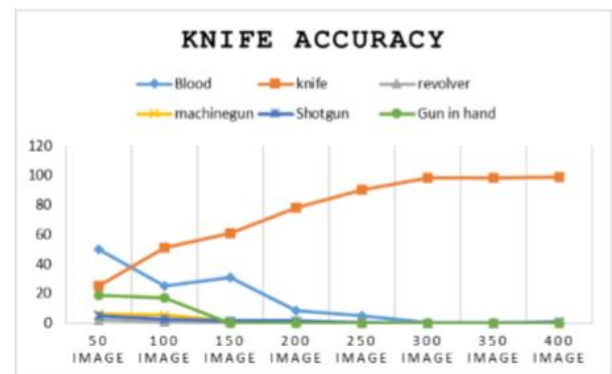


Fig. 16: Knife accuracy graph

From figure 16 we can see that, as we have trained it for an increasing number of knife samples, the accuracy percentage also increased from 20% to approximately 98-99%. But at the same time in a real life scenario, the percentage can fluctuate.

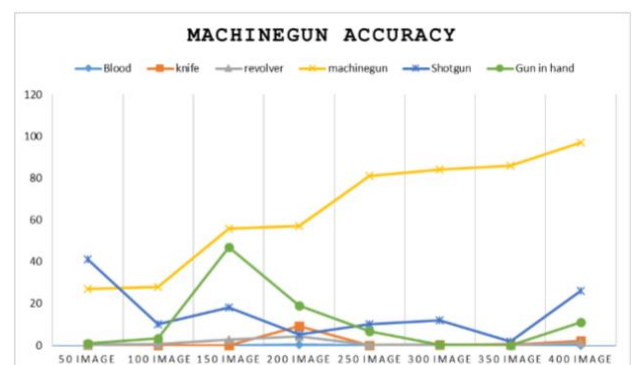


Fig. 17: Machinegun accuracy graph

From figure 17 we can see that, as we have trained it for an increasing number of shotgun samples, the accuracy percentage also increased from 25% to 85%. But at the same time in a real life scenario, the percentage can fluctuate. From figure 3.18 we can see that, as we have trained it for an increasing number of revolver samples, the accuracy percentage also increased from 2.7% to 76%. But at the same time in a real-life scenario, the percentage can fluctuate.

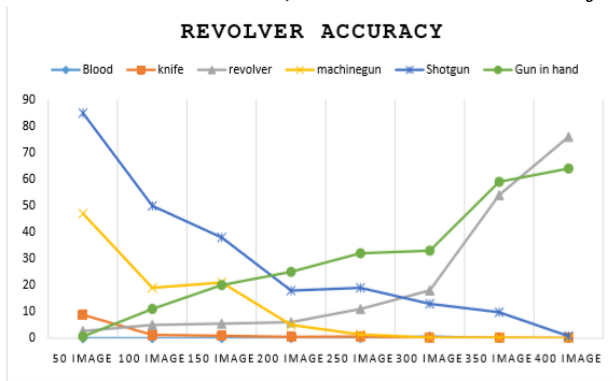


Fig. 18: Revolver accuracy graph

From figure 19, we summarized our final accuracy. Using our dataset for training the network for blood samples, we detected blood from various real-life crime scenario. The problem here is, sometimes if anything occurs which's color is close 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 a knife, it can detect both. As we have trained it for the mass amount and at the same time a limited amount of blood in a scenario, it has gained an accuracy of almost 90 to 97 per cent, 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. But as we increased training data samples, after 200 images and 4000 iterations, the accuracy was between 90 to 99%.

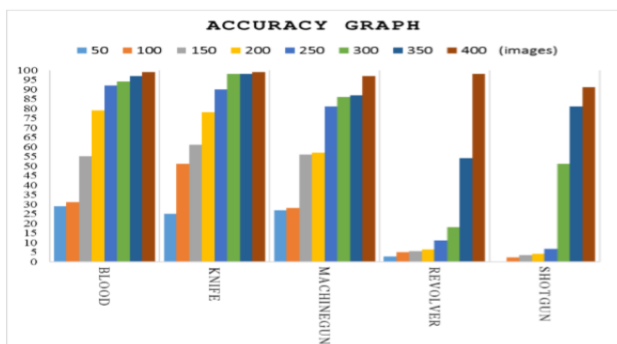


Fig. 19: Final Accuracy graph

In case of training the network for knife samples, the graph shows good accuracy in detecting knife. The DNN successfully detects knife, because knives shape is different from machinegun, 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.

Our main purpose was to detect every different class like knives, blood, revolvers, shotguns and machineguns. Our model was able to perfectly detect blood and knives, but in case of firearms, it is not yet up to the mark. But the main purpose of guessing as for if a crime has happened or not can be guessed by detecting any type of firearms, whether it is a revolver, machinegun or shotgun. Also where there are multiple objects, like blood on the knife, machineguns alongside a revolver – whatever the scenario is, it was able to detect multiple class from the same scenario. So we can say that the proposed model is accurate in most cases, with some exceptions which can be fixed by doing further research.

## 5. CONCLUSION

We can successfully detect blood, knife and gun on which we can predict the crime scene occurred or not. The wrong alert is

reduced that makes us our model very efficient for this task compared to other models. To implement a convolutional neural network, TensorFlow is the best platform we have found so far for this field and implanted in it. In this paper, we have 6 classes and for each of the class, our system can detect the threatening objects and give a result from an image. The results are given in percentage for each of the objects we want to detect. Predicting crime scene by detecting threatening objects can have far reaching impact on computer vision field. For our datasets, the test accuracy is 90.2 % that is very competitive with the systems we have seen so far.

## 6. REFERENCES

- [1] Abadi, Martín, et al. "TensorFlow: A system for large-scale machine learning." Proceedings of the 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. Web. 10 Feb. 2017. Available: <https://ujjwalkarn.me/2016/08/11/intuitive-explanationconvnets>
- [3] "An open-source software library for Machine Intelligence". Web. 25 Dec. 2016. Available: <https://www.tensorflow.org/>
- [4] Dinesh Kumar Saini, Dikshika Ahir and Amit Ganatra, "Techniques and Challenges in Building Intelligent Systems: Anomaly Detection in Camera Surveillance", Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems, Springer International Publishing Switzerland, 2016
- [5] B Ravi Kiran, Dilip Mathew Thomas, Ranjith Parakkal, "An overview of deep learning based methods for unsupervised and semisupervised anomaly detection in videos", MDPI Journal of Imaging, arXiv:1801.03149v1, 2018
- [6] Ryota Hinami, Tao Mei, and Shin'ichi Satoh, "Joint Detection and Recounting of Abnormal Events by Learning Deep Generic Knowledge", arXiv:1709.09121v1, 2017
- [7] "Learn TensorFlow and Deep Learning, without a PhD | Google Cloud Big Data and Machine Learning Blog | Google Cloud Platform." Google Cloud Platform. Web. 10 Mar. 2017. Available: <https://cloud.google.com/blog/big-data/2017/01/learn-tensorflow-and-deep-learning-without-a-phd>
- [8] P. Sermanet and Y. Lecun, "Traffic sign recognition with multi-scale Convolutional Networks," The 2011 International Joint Conference on Neural Networks, 2011.
- [9] 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), 2016K. 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.
- [10] Y. Xu, T. Mo, Q. Feng, P. Zhong, M. Lai, E. I. Chang, et al. Deep learning of feature representation with multiple instance learning for medical image analysis. In ICASSP, 2014.
- [11] 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).
- [12] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. Cornell University Library.
- [13] D. Lowe. "Object recognition from local scale-invariant features". In ICCV, 1999.
- [14] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In CVPR, 2005. [15] 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).
- [15] 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).
- [16] Grega, Michael, Andrzej Mاتیolanski, Piotr Guzik, and Mikolaj Leszczuk. "Automated Detection of Firearms and Knives in a CCTV Image." *Sensors* 16.1 (2016): 47
- [17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (January 2014), 1929-1958.
- [18] Kingma, Diederik P., and Jimmy Ba. "Adam: A Method For Stochastic Optimization" 3rd International Conference for Learning Representations, San Diego, 2015 v1 (2014).
- [19] M. Ribeiro, A.E.L., and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos", *Pattern Recognition Letters*, ELSEVIER, 2017.