Automatic Traffic Sign Localization and Classification from Dash-cam Images

*Shafkat Rahman Farabi, †Md. Zahidul Islam, ‡MD. Fakhruddin Gazzali,
§SM Hasibul Haque Himel, ¶Md.Rukonnuzzaman and ‖Raiyan Ahmed
Department of Computer Science and Engineering, Islamic University of Technology, Dhaka, Bangladesh
*shafkatrahman@iut-dhaka.edu,†zahidulislam@iut-dhaka.edu ‡fakhruddingazzali@iut-dhaka.edu
§himel3.1416@gmail.com, ¶rukonfahim@gmail.com, ‖raiyanahmed@iut-dhaka.edu
**160041002,†160041010,‡160041014,§160041018, ¶160041016,‖160041037*

***Abstract*—We propose and implement a system that can detect and classify traffic signs from images taken by dashboard camera of automobiles. Out system takes a deep learning based approach. We design a novel convolutional neural network architecture that can discriminate between 43 classes. We train and test our model on GTSRB dataset.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Our project primarily aims at building a system that can solve the problem of detecting and classifying traffic signs accurately from images. This falls within the domain of computer vision. We applied deep learning to accomplish this challenging task.

Artificial intelligence and machine learning is ubiquitous in this modern day and age. Application of state of the art machine learning techniques can be found all around us. Some promising approaches of machine learning can be computer vision, natural Language processing, machine translations, self driving cars etc. In case of computer vision tasks, machine learning has been applied very successfully for performing tasks such as image classification, object recognition, image segmentation and many more. With the recent advent of deep learning, these fields have seen the development of novel state of the art methods on the popular benchmarks.

The problem we are trying to address falls within the domain of computer vision and the sub-domain of image recognition/classification. When driving, there are usually a lot of traffic signs and/or instructions on the road meant for the driver. Often following these instructions under different lighting conditions while driving can turn out to be quite challenging. Cars are becoming increasingly more automated with the advent of technology. Yet,to our knowledge, no AI assisted computer vision systems are used to help the operator of automobiles to safely operate their vehicles. Our proposed system will help create a more safe condition for driving cars by analyzing images captured by dashboard mounted cameras

of automobiles and automatically recognizing, classifying and annotating traffic signs. Self driving cars need such system to develop a clear understanding of the roads around them. Normal cars can also benefit from systems like this to alert the driver whenever necessary.

Fig. 1. Traffic sign localization and recognition. Red rectangles shows the localization of traffic sign. The recognized class is shown in yellow text.

To accomplish the task of detecting and localizing traffic signs in a image, we used haar cascade classifier [1] which yields reasonable results with fast detection time.

After detecting possible rectangles that might contain traffic sign, we input the cropped and processed image in a deep convolutional neural network [2] that classifies the traffic sign in one of 42 different classes. We trained and tested the deep neural network end to end on publicly available *German Traffic Sign Recognition Benchmark* dataset.

out contributions are to design a novel deep learning architecture and pairing this with a object localization system and developing a pipeline that can localize and classify images with a high accuracy.

II. Background Study

This project tries to apply deep learning for solving a computer vision task. Deep learning is the application of neural network that is deep in the context of number of layers. Because this is a computer vision task, we opted to utilize a convolutional neural network for this specific task. Out implemented system can be divided in 2 parts. The first portion is responsible for localizing traffic sign from a image of arbitrary size (preferably one taken using dashboard camera of a vehicle. Second portion is the deep neural network responsible for classification. Detailed discussions of both follow.

A. Traffic sign detection and localization

To find out the regions containing traffic signs we used a well known machine learning technique called *Haar Cascade Classifier* [1]. It is an effective object detection technique that gave us reasonable results in cropping out the areas of the image that might contain a traffic sign in it. After that, we perform classification on that cropped image to get which traffic sign is it. Haar feature based cascade classifier were initially proposed by Paul Viola and Michael Jones in their work titled, "*Rapid Object Detection using a Boosted Cascade of Simple Features(2001)* [3]" (add reference here). In this method, we have to provide a lot of positive and negative sample. The positive samples contain images of traffic signs. The negative samples contain random object that are not traffic signs. The algorithm extracts features from the positive and negative images which look like convolutional kernels.

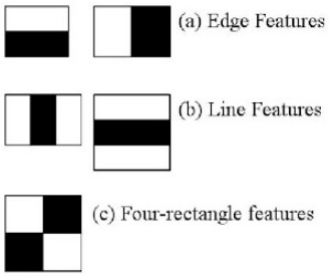

Fig. 2. Haar Kernels

Feature that we calculate in this method are actually a single value for each of the kernel that we get by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle. That will be a lot of features as the kernels traverse through the whole image. But, the algorithm uses many clever techniques to make this algorithm very fast. It uses *Adaboost* [4] (add reference for adaboost paper) algorithm to focus more on the wrongly classified samples. The classifier is actually a collection of many weak classifier which together gives a strong performance. Rather that using a lot of features together, it groups them into separate stages applied one by one. If a window fails the first stage we discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a detected region. That's why it is called *Cascade of Classifiers*. This algorithm is also referred to as *Viola-Jones* [3] object detection algorithm.

After finding the regions that may contain the traffic signs using *Viola-Jones* object detection algorithm, we crop out the traffic signs using boundary boxes and pass it to the neural network for classification.

B. Traffic sign classification

This portion takes as input a cropped image containing a traffic sign and classifies it as one out of 43 different classes. For developing this portion, we used popular and publicly available dataset *German Traffic Sign Recognition Benchmark*.

Fig. 3. Some samples from the GTSRB dataset

The *German Traffic Sign Recognition Benchmark* or *GTSRB* dataset is a multi-category classification competition held at IJCNN 2011. The dataset is composed of 50,000 images in total and 43 classes. The images are all cropped traffic signs. The dataset mimics real life situation in terms of lighting and image quality. After preliminary assessment, we found certain issues with the dataset:

- The images have low resolution.
- The images have low contrast.
- The dataset has a class skew. Some classes have as many as 2000 examples, while some have as little as 180 examples

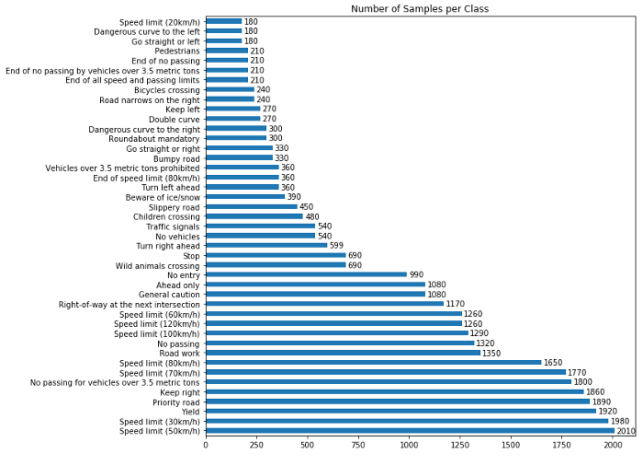

Fig. 4. Skew present in GTSRB dataset

To solve the problem with class skew, we had to assign weights to different classes when calculating the training and test loss. For improving contrast, we preprocessed the images

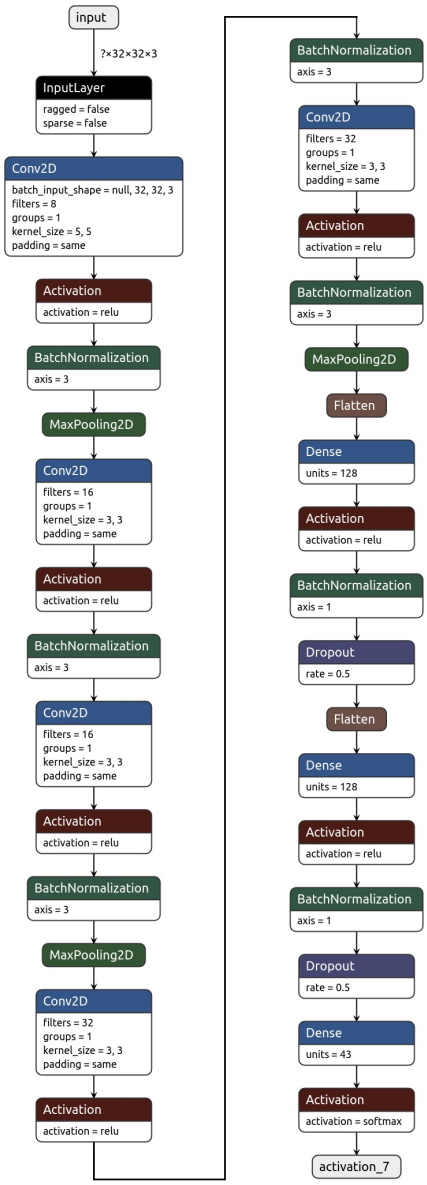

Fig. 5. Model architecture

using *Contrast Limited Adaptive Histogram Equalization* or CLAHE [5] in short. This improved the performance of the classifier.

The deep neural convolution neural network takes in 32×32 RGB images and passes it through 3 convolutional blocks. The first block has 8 convolution filters of size 5×5 with stride 1 and 'same' padding. This is followed by relu,batch normalization and a pooling layer of size 2×2 and stride specified to be 2 in both dimensions. The 2^{nd} convolutional block uses 16 3×3 filters with stride 1 followed by relu activation, batch normalization, another convolutional layer with 16 3×3 filters, relu, batch normalization and finally max pooling with filter size 2×2. 3^{rd} block is the same as 2^{nd} block but uses 32 convolutional filters instead of 16. The output tensor from this final convolutional block is flattened and given as input to a fully connected layer with 128 units. This is followed with another fully connected layer with 128 neurons, output of which are input to a softmax layer with 43 neurons. Please refer to fig.3 for better understanding.

The network has 107,147 parameters in total of which 106,427 are trainable.

Fig. 6. Detecting regions containing traffic signs. The detected region is marked with a yellow rectangle.

III. Implementation

We first implemented *Viola-Jones* [3] object detection algorithm. For our project, we trained the classifier to detect traffic signs using 500 positive and 500 negative images. The positive images are cropped out traffic signs whereas the negative signs are random images of objects like people, dogs, cats, scenery etc. Then, we use the learned features to extract ROI or Region of Interest from the input image. The detection algorithm will iteratively run a window over the whole image and find out the regions that gives the best similarity with the features. That means, given a full size of image of a street it can detect the boundary boxes that contains the traffic signs. OpenCV [6] library has easy to use implementations of this algorithm. The output of traffic sign detection is shown in figure 6.

After implementing and training the *Viola-Jones* [3] object detection algorithm, we focus on training the deep neural network dedicated for traffic sign classification shown in figure 5. The first problem we encounter is, the bounding boxes generated and cropped using the *Viola-Jones* object detection algorithm have no fixed size. Furthermore, the *GTSRB* [7] dataset containes images of varying sizes. Some images are as small as 25×25 while biggest are 166×152. While the deep neural network's input layer can only take in 32×32 images. Thus the images need to be preprocessed and resized before being used for training / testing or for inference. Furthermore,*Contrast Limited Adaptive Histogram Equalization* or CLAHE [5], which is a type of histogram equalization, was used for improving the overall contrast of the input image as an additional preprossessing step. We normalized the pixel values to be in a range [0-1] as it helps the training process to converge faster.

For the purpose of data augmentation, we randomly rotated the images in the training dataset in a range of $\pm 10°$, randomly

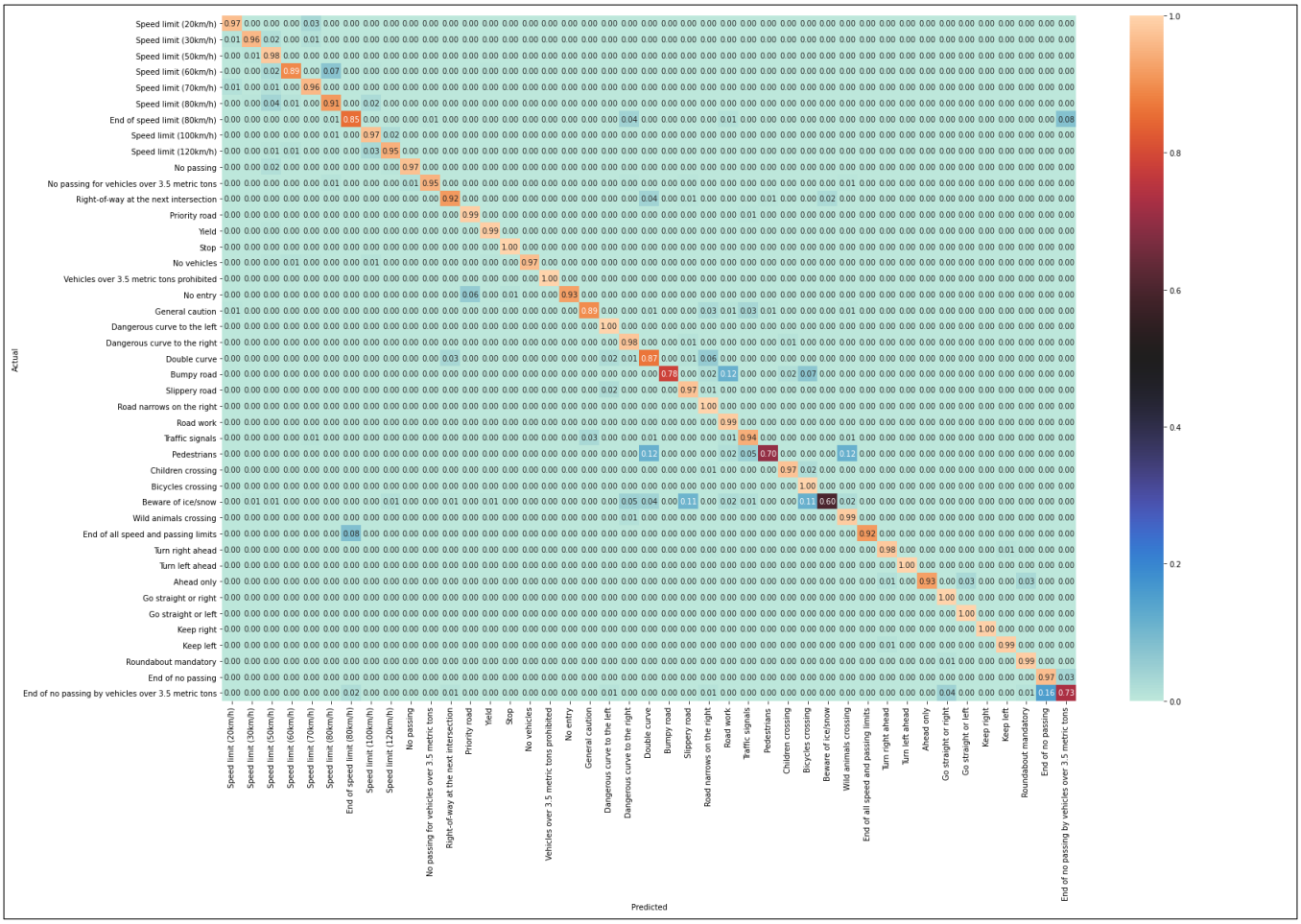

Fig. 7. Heat-map generated from confusion matrix on the test set.

zoomed, randomly shifted in width and height and applied sheer in the vertical direction in a random manner. All of this helped the classifier learn weights in a way that generalized to real world data.

We used categorical crossentropy as the loss function which is defined as

$$\ell(y_{pred}, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i log y_{pred,i} \tag{1}$$

The entire model is trained end to end using Adam optimizer [8] with a initial learning rate of 0.0001 and a learning rate decay of $\frac{0.0001}{numberofepoch \times 0.5}$. The model was trained for 50 epochs with mini batch size of 64. We used dropout with probability 0.5 in the 2 fully-connected layers to reduce overfitting. As mentioned previously, because there is a skew in the dataset, to account for it, we employed a weighting technique, where we assigned each class a weight inversely proportional to the percentage of training examples that belongs to it compared to the entire dataset. When calculating the loss and accuracy, we multiplied each training example's contribution with the weight of the class the example belongs to. This way, the model is penalised more for misclassifying an example that belongs to a class with small number of training examples than a class with a huge number of training example. This stops the model from learning to predict the most common class and achieve a high training accuracy in this way.

IV. Result Analysis

Our deep neural network classifier achieves competitive results. We initially ran the training for 50 epochs. However the models seems to overfit starting from epoch 45. Hence we employ early stopping and use the epoch 44 model as the final one as it achieves highest validation accuracy of 95% and training accuracy of 98.5%. Which indicated the model generalizes pretty well and is suited for most use cases. We did not use a separate test set. We used our test set as a validation set. By looking at the confusion matrix of the final trained model, we can further conclude that this model is indeed a good model. We can see the diagonal coxes have high number of elements. Which means almost 80%-90% have been classified properly. The classes that have poor precision (mi-classified as some other class) are either the classes that have comparatively low number of examples in the training set, or are ambiguous. For example *end of no passing* class

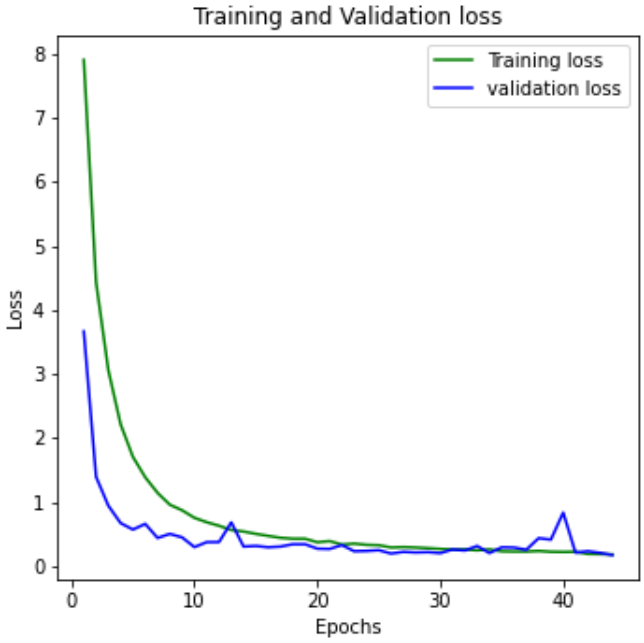

Fig. 8. Loss Vs epoch

Fig. 9. accuracy Vs epoch

has been properly classified only 73% of the time. 16% of the time it has been misclassified as *end of no passing by vehicles over 3.5 metric tons*. The ambiguity in this case is rather understandable.

As it is quite evident out model generalizes quite well to the test data. In fact the model has a test accuracy of 95% and a weighted accuracy of 95% across 12630 test data images. For

TABLE I
CLASS WISE CLASSIFICATION REPORT ON THE TEST DATA

Class	Precision	Recall	f1-score	Support
Speed limit (20km/h)	0.77	0.97	0.86	60
Speed limit (30km/h)	0.98	0.96	0.97	720
Speed limit (50km/h)	0.91	0.98	0.95	750
Speed limit (60km/h)	0.97	0.89	0.93	450
Speed limit (70km/h)	0.98	0.96	0.97	660
Speed limit (80km/h)	0.92	0.91	0.92	630
End of speed limit (80km/h)	0.94	0.85	0.90	150
Speed limit (100km/h)	0.92	0.97	0.94	450
Speed limit (120km/h)	0.95	0.95	0.95	450
No passing	0.98	0.97	0.98	480
No passing for vehicles over 3.5 metric tons	0.99	0.95	0.97	660
Right-of-way at the next intersection	0.98	0.92	0.95	420
Priority road	0.96	0.99	0.98	690
Yield	0.99	0.99	0.99	720
Stop	0.98	1.00	0.99	270
No vehicles	1.00	0.97	0.98	210
Vehicles over 3.5 metric tons prohibited	1.00	1.00	1.00	150
No entry	1.00	0.93	0.96	360
General caution	0.98	0.89	0.94	390
Dangerous curve to the left	0.88	1.00	0.94	60
Dangerous curve to the right	0.79	0.98	0.88	90
Double curve	0.68	0.87	0.76	90
Bumpy road	1.00	0.78	0.88	120
Slippery road	0.86	0.97	0.91	150
Road narrows on the right	0.80	1.00	0.89	90
Road work	0.95	0.99	0.97	480
Traffic signals	0.88	0.94	0.91	180
Pedestrians	0.88	0.70	0.78	60
Children crossing	0.97	0.97	0.97	150
Bicycles crossing	0.76	1.00	0.87	90
Beware of ice/snow	0.90	0.60	0.72	150
Wild animals crossing	0.93	0.99	0.96	270
End of all speed and passing limits	1.00	0.92	0.96	60
Turn right ahead	0.97	0.98	0.98	210
Turn left ahead	1.00	1.00	1.00	120
Ahead only	1.00	0.93	0.96	390
Go straight or right	0.94	1.00	0.97	120
Go straight or left	0.83	1.00	0.91	60
Keep right	1.00	1.00	1.00	690
Keep left	0.97	0.99	0.98	90
Roundabout mandatory	0.83	0.99	0.90	90
End of no passing	0.79	0.97	0.87	60
End of no passing by vehicles over 3.5 metric tons	0.82	0.73	0.78	90
macro avg	0.92	0.94	0.93	12630
weighted avg	0.95	0.95	0.95	12630

quantitative analysis, we have attached some example outputs shown in figures 10 and 11 provided by the entire system (localization + classification). As we can see out system is quite adapted to detect signs regardless of its size in the input image. Furthermore, we can see in fig. 10, our system has no trouble detecting traffic signs even when multiple signs are present simultaneously.

The average detection + classification time is 1.77 seconds on a CPU system even without powerful GPU parallization . We believe this is fast enough to function as a real time system.

V. FEATURE MAP VISUALIZATION

Feature map visualization [9] can give us an idea about what the deep learning models are actually learning. The feature maps capture the result of applying the filters to a given input image. The idea of visualizing a feature map for a specific input image would be to understand what features of the input are detected or preserved in the feature maps. The expectation would be that the feature maps close to the input detect small or fine-grained detail, whereas feature maps close to the output of the model capture more general features. In order to explore the visualization of feature maps, we will provide our trained neural network with an image of traffic sign. Then, we will

Fig. 10. Example output:Dangerous curve to the right

Fig. 11. Example output: End of no passing by vehicles over 3.5 metric ton

extract the output from activation of the convolutional layers. Then we would be able to see the effect of the filters on our image. We provided fig. 12 as input to the already trained model. Figures 13,14,15,16 are the feature maps that we get from the outputs of 6th, 9th, 13th and 16th layers of our model. These clearly shows that the earlier layers learn low level features like edge, shape etc. The later layers learn high level features which are not interpretable from feature maps.

Fig. 12. Provided Image for Feature Map Visualization

VI. CONCLUSION

In this machine learning project, we implemented a two stage traffic sign detection. In the first stage it finds out the region of interests on the given image which are likely to contain a traffic sign. This is done using a machine learning

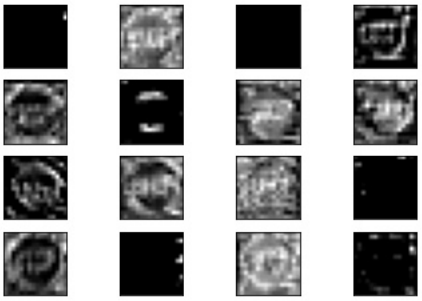

Fig. 13. Output of 6th layer

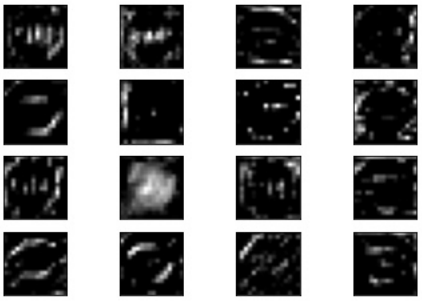

Fig. 14. Output of 9th layer

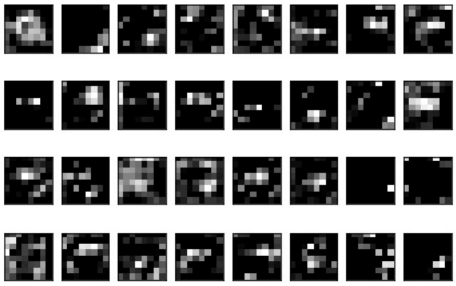

Fig. 15. Output of 13th layer

approach called *Viola-Jones* [3] object detection algorithm. Then, in the second stage a deep convolutional neural network classifies the ROI of the given image into a type of traffic sign. So, our project actually leverage both classical machine learning algorithm and modern deep learning model. It gives reasonable accuracy. But, recent deep learning approaches can localize and recognize the objects at the same time. For example, Faster RCNN [10] can detect and recognize traffic signs very efficiently. *You only look once* (YOLO) [11] is a state-of-the-art, real-time object detection system which can

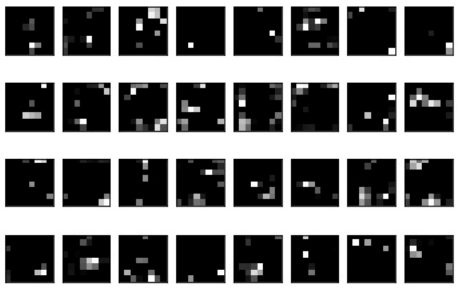

Fig. 16. Output of 16th layer

detect multiple objects at a single pass making it a extremely fast approach. So, in a real life scenerio, we should implement these modern approaches as they are faster and more accurate.

REFERENCES

[1] S. Soo, "Object detection using haar-cascade classifier," *Institute of Computer Science, University of Tartu*, pp. 1–12, 2014.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.

[4] T.-K. An and M.-H. Kim, "A new diverse adaboost classifier," in *2010 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 1. IEEE, 2010, pp. 359–363.

[5] E. D. Pisano, S. Zong, B. M. Hemminger, M. DeLuca, R. E. Johnston, K. Muller, M. P. Braeuning, and S. M. Pizer, "Contrast limited adaptive histogram equalization image processing to improve the detection of simulated spiculations in dense mammograms," *Journal of Digital imaging*, vol. 11, no. 4, p. 193, 1998.

[6] G. Bradski and A. Kaehler, "Opencv," *Dr. Dobb's journal of software tools*, vol. 3, 2000.

[7] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: a multi-class classification competition," in *The 2011 international joint conference on neural networks*. IEEE, 2011, pp. 1453–1460.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[9] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

APPENDIX A
CODE SNIPPETS

■ trafficsignNet.py

```
class TrafficSignNet:
   @staticmethod
   def build(width,height,depth,classes):
      #build and initialize the model
      model=Sequential()
      inputshape=(height,width,depth)
      chanDim=-1
      #CONV => RELU => BN => POOL
      model.add(Conv2D(8,(5,5),padding="same",
              input_shape=inputshape))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(MaxPooling2D(pool_size=(2,2)))

      #(CONV => RELU => CONV => RELU) => POOL

      model.add(Conv2D(16,(3,3),padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(Conv2D(16,(3,3),padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(MaxPooling2D(pool_size=(2,2)))

      model.add(Conv2D(32,(3,3),padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(Conv2D(32,(3,3),padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(MaxPooling2D(pool_size=(2,2)))

      #Two sets of fully connected layers and a softmax classifier

      model.add(Flatten())
      model.add(Dense(128))
      model.add(Activation("relu"))
      model.add(BatchNormalization())
      model.add(Dropout(0.5))

      model.add(Flatten()) #is this even necessary?
      model.add(Dense(128))
      model.add(Activation("relu"))
      model.add(BatchNormalization())
      model.add(Dropout(0.5))

      model.add(Dense(classes))
      model.add(Activation("softmax"))

      return model
```

■ **train.py**

```
import matplotlib
matplotlib.use("Agg")

from TrafficSignNet import TrafficSignNet
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import callbacks

from sklearn.metrics import classification_report
from skimage import transform
from skimage import exposure
from skimage import io
import numpy as np
import matplotlib.pyplot as plt
import argparse
import random
import os



def load_split(basePath,csvPath):
   data=[]
   labels=[]
   rows=open(csvPath).read().strip().split("\n")[1:]
   random.shuffle(rows)
   for (i,row) in enumerate(rows):
      if i>0 and i % 1000 == 0:
         #break
         print("preprocessed total %d images"%(i));
      (label,imagePath) = row.strip().split(",")[-2:]
      imagePath=os.path.sep.join([basePath,imagePath])
      image=io.imread(imagePath)
      image=transform.resize(image,(32,32))
      image=exposure.equalize_adapthist(image,clip_limit=0.1)
      data.append(image)
      labels.append(int(label))
   data=np.array(data)
   labels=np.array(labels)
   return (data,labels)




ap = argparse.ArgumentParser()
ap.add_argument("-d","--dataset",required=True,help="path to input dataset")
ap.add_argument("-m","--model",required=True,help="path to output model")
ap.add_argument("-p","--plot",required=True,help="path to output plot")
ap.add_argument("-e","--epoch",required=True,help="number of training epochs")
args = vars(ap.parse_args())



NUM_EPOCHS = int(args['epoch'])
INIT_LR = 1e-3
Batch_siz = 64
signature = "trial_run_1"

filepath = os.path.join(args["model"],"saved-model-{epoch:02d}.h5")

checkpoint = callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
    save_best_only=False, save_weights_only=False, mode='auto', period=1)
```

```
labelnames =
    open("/content/traffic-sign-recognition-tutorial-code/signnames.csv").read().strip().split("\n")[1:]
labelnames = [ l.split(",")[1] for l in labelnames]


trainPath = os.path.sep.join([args["dataset"],"Train.csv"])
testPath = os.path.sep.join([args["dataset"],"Test.csv"])

print("[INFO] loading and preprocessing the data")

(trainX,trainY) = load_split(args["dataset"],trainPath)
(testX,testY) = load_split(args["dataset"],testPath)

trainX = trainX.astype("float32")/255.0
testX = testX.astype("float32")/255.0

numLabels = len(np.unique(trainY))

trainY = to_categorical(trainY,numLabels)
testY = to_categorical(testY, numLabels)

classTotal = trainY.sum(axis=0)
print("classtotal ",classTotal)
classWeight = classTotal.max()/classTotal
classWeight = {i : classWeight[i] for i in range(len(classWeight))}


aug = ImageDataGenerator(
   rotation_range=10,
   zoom_range=0.015,
   width_shift_range=0.1,
   height_shift_range=0.1,
   shear_range=0.15,
   horizontal_flip=False,
   vertical_flip=False,
   fill_mode="nearest")

print("[INFO] compiling model......")
opt = Adam(lr=INIT_LR,decay=INIT_LR/(NUM_EPOCHS*0.5))
model = TrafficSignNet.build(width=32,height=32,depth=3,classes=numLabels)
model.compile(loss="categorical_crossentropy",optimizer=opt,metrics=["accuracy"])
print("[INFO] training the model.....")
H = model.fit(
      aug.flow(trainX,trainY,batch_size=Batch_siz),
      validation_data=(testX,testY),
      steps_per_epoch=trainX.shape[0]//Batch_siz,
      epochs=NUM_EPOCHS,
      class_weight=classWeight,
      verbose=1,
      callbacks= [checkpoint]
      )


print("[INFO] testing the model.....")
predictions = model.predict(testX,batch_size=Batch_siz)
print(classification_report(
      testY.argmax(axis=1),
      predictions.argmax(axis=1),
      target_names=labelnames,
      ))

np.save(os.path.join(args["plot"],'run{}history_final.npy'.format(signature)),H.history)

print("[INFO] saving network to '{}'...".format(args["model"]))
model.save(args["model"])
plt.style.use("ggplot")
plt.figure()
```

```
N = np.arange(0, NUM_EPOCHS)
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.plot(N, H.history["accuracy"], label="train_acc")
plt.plot(N, H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

■ **detectionPlusRecognition.py**

```
import numpy as np
import cv2
import time
from tensorflow.keras.models import load_model
from skimage import transform
from skimage import exposure
import os


#cascade.xml contains features needed to find traffic signs
cascade = cv2.CascadeClassifier('cascade.xml')

img = cv2.imread('fullsizeimagesformllabproject_/00003.jpg')
full_image = np.array(img)
#gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # no need to convert to gray

#loading conv model
print("[INFO] loading model....")
#set model path here
print(os.path.exists('pretrained_models/saved-model-44.h5'))
model = load_model('pretrained_models/saved-model-44.h5')
labelnames = open("signnames.csv").read().strip().split("\n")[1:]
labelnames = [l.split(",")[1] for l in labelnames]

#detection
tic = time.time()
boxes = cascade.detectMultiScale(img, scaleFactor = 1.01, minNeighbors = 7, minSize= (24,24),
    maxSize=(128,128))
tac = time.time()
print('detection time : ', (tac - tic),'seconds')

#drawing boundary boxes on input image
for (x,y,w,h) in boxes:
   print(x,y,w,h)
   img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
   #model.predict
   cropped_image = full_image[ y:y+h,x:x+w, : ]
   cropped_image = transform.resize(cropped_image,(32,32))
   cropped_image = exposure.equalize_adapthist(cropped_image, clip_limit=0.1)
   cropped_image = cropped_image.astype("float")/255.0
   cropped_image = np.expand_dims(cropped_image, axis=0)
   preds = model.predict(cropped_image)
   j = preds.argmax(axis=1)[0]
   label = labelnames[j]
   print(" j:",j," max_pred:",preds.max(), " label:",label)
   cv2.putText(img, label, (x, y), cv2.FONT_HERSHEY_SIMPLEX,0.45, (0, 255, 255), 2)

#saving output file
cv2.imwrite('output1.jpg', img)
```

■ **featureMapVisualization.py**

```
import numpy as np
import cv2
import time
from skimage import io
from tensorflow.keras.models import load_model
from skimage import transform
from skimage import exposure
from tensorflow.keras.models import Model
from matplotlib import pyplot as plt
from numpy import expand_dims
import argparse


### HOW TO RUN
# python featureMapVisualization.py --model MODEL_PATH --image INPUT_IMAGE_PATH

### WHAT IT DOES
# it will run the model with given image and plot features maps from certain layers and also
    save them as images


ap = argparse.ArgumentParser()
ap.add_argument("-m","--model", required=True, help="path to the model")
ap.add_argument("-i","--image", required=True, help="path to the image")
args = vars(ap.parse_args())


image = io.imread(args["image"])
plt.imshow(image)
plt.title('input image')
plt.show()
image = transform.resize(image,(32,32))
image = exposure.equalize_adapthist(image, clip_limit=0.1)
image = image.astype("float")/255.0
image = np.expand_dims(image, axis=0)



model1 = load_model(args["model"])
indexes = [6,9,13,16]
# index of the layers at the end of each conv block (Conv>Activation>BN) of trafficSignNetModel
num_features = [4,4,8,8]
outputs = [model1.layers[i].output for i in indexes]
model2 = Model(inputs=model1.inputs, outputs=outputs)



feature_maps = model2.predict(image)



for ind,fmap in enumerate(feature_maps):
   ix = 1
   figtitle = 'output_of_layer_'+ str(indexes[ind],) + '_' + model1.layers[indexes[ind]].name
   print(figtitle)
   for _ in range(4):
      for _ in range(num_features[ind]):
         ax = plt.subplot(4,num_features[ind],ix)
         ax.set_xticks([])
         ax.set_yticks([])
         plt.imshow(fmap[0,:,:,ix-1], cmap = 'gray')
         ix += 1
   plt.savefig(figtitle + ".jpg")
   plt.show()
```