COMPUTER VISION-I PROJECT

TEAM 11

TOPIC: TRAFFIC SIGN IDENTIFICATION

Made by

Abhishek Raj (671107236)

Aniket Kansara (655401422)

Gowtham Natarajan (651562708)

TABLE OF CONTENTS

1. Problem Statement & Introduction

Object identification has many applications in various fields like autonomous vehicles. In all over the world, important information about the road condition and its limitations are introduced to drivers as visual signals, such as traffic signs. Traffic signs are an important part of road infrastructure to provide information about the condition of road, warnings, prohibition, restriction, and other helpful information to the driver for navigation. They provide important information which can be interpreted by drivers. During inferior traffic or bad weather conditions, driver may not notice the signs directly or indirectly, which may lead to accidents or serious injuries. During such circumstances, if there is an automatic detection system for traffic signs, it can warn driver of such signs on the road and help him follow such signs and thus making driving safe. Advanced driver assistance system (ADAS) is one of the fastest growing fields in autonomous vehicle. ADAS technology is completely based upon vision system, active sensor technology and car data network. A vision-based road sign detection system is thus necessary to catch the driver's attention to avoid any accidents. However, there are many factors which make the road sign detection difficult such as lighting conditions poor or bright, deformation of signs, angle at which they are placed. Thus, our aim in this project is to write an algorithm for vision-based traffic sign Identification.

2. Implementation Details

As mentioned in the initial draft, this project is sticking with the same procedure as follows

a. Image Recognition - It is the first step where it receives the input image from Sensor (Camera) and process it to remove the objects. The common detection methods are Color-based and Shape-base methods. We tried using Template matching algorithm from OpenCV to recognize the traffic sign from the image.
b. Tracking - Due to the high robustness in the image recognition we plan to use a tracker to remove unwanted objects as suggested in the research papers. Instead of using complex algorithms this project involves non-overlapping algorithm to reduce capture of unwanted objects in the input image.

3. Data

The inputs used in our project are German Traffic Sign Detection Benchmark (GTSDB) and German Traffic Sign Recognition Benchmark (GTSRB). These datasets will be downloaded from INI benchmark Website. This is the large multi-category classification benchmark dataset. The database is German Traffic Sign Recognition Benchmark (GTSRB). The input will cover a variety of possible conditions such as the Images which are often low in resolution and of different scales. Furthermore, traffic sign image acquisition from a device in motion may generate blurry images and this is further complicated by the occlusion, affine transformation, varying lighting and bad weather conditions. The dataset contains 51,840 images of the 43 classes that are shown in Fig. 1. The relative class frequencies of the classes are shown in Fig. 2.

The GTSRB dataset was split into three subsets. We applied stratified sampling. The split was performed at random but considering class and track membership. This makes sure that (a) the overall class distribution is preserved for each individual set and that (b) all images of one traffic sign instance are assigned to the same set, as otherwise the datasets could not be considered stochastically independent.

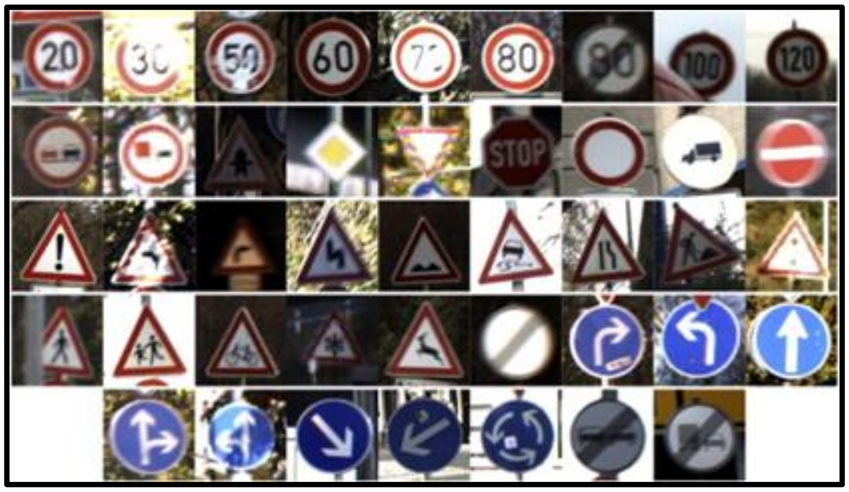


Fig 1: Random representatives of the 43 traffic sign classes in the GTSRB dataset.

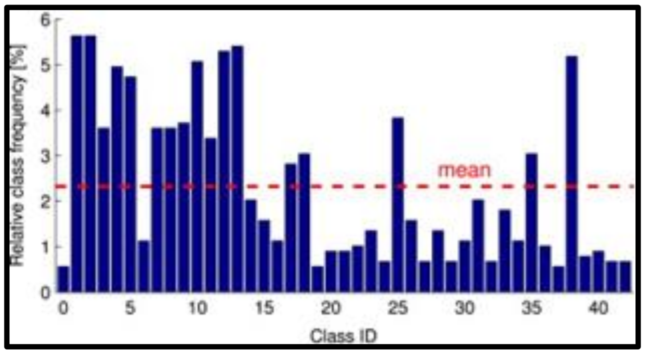


Fig 2: Relative Class Frequency

4. Algorithm

a. Template Matching Algorithm

Template Matching algorithm involves the process of searching and finding the location of template image in the input image. We must have the small template images of all the reference images before we start this process. The template images should have the objectives of traffic signs. Because the algorithm matches the input image with all the possible templates to recognize the image. So, we must have a proper template image data set. This algorithm can be done by the built-in function cv2.matchTemplate(). If input image is of size (W x H) and template image is of size (w x h), the output image will have a size of ((W-w+1) x (H-h+1)). Once result is obtained, we can use cv2.minMaxLoc() function to find where is the maximum/minimum value. We can take it as the top-left corner of rectangle and take (w,h) as width and height of the rectangle. That rectangle is our region of template which recognizes the traffic sign out of the input image.

5. Testing and Performance Measurement

The evaluation of the detection stage will be performed based on precision-recall curve, where the recall and precision values will be computed as follows:

i. Recall=(Number of correctly detected signs)/(Number of true signs) ×100
ii. Precision=(Number of correctly detected signs)/(Number of detected signs) ×100

The CCR and the average running time of the features used in the work will be calculated.

6. Method

a. cv2.TM_CCOEFF_NORMED:

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

There are six methods in template matching algorithm. But the method used in this algorithm is coefficient normalised.

T(x,y) = Template Image (Small Image)
I(x,y) = Large Image
R(x,y) = Output Image

7. Flow of Code

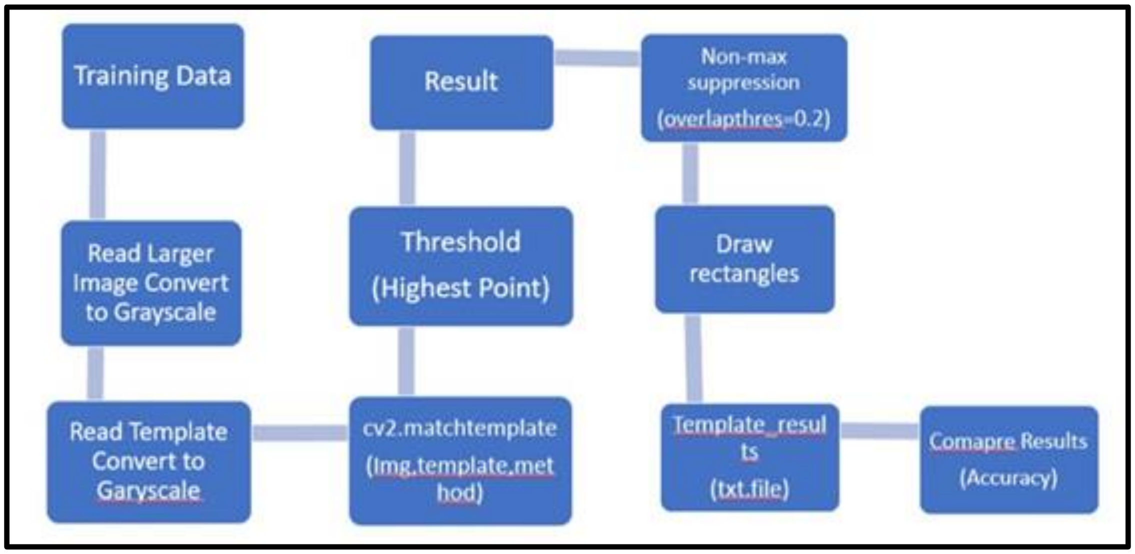

Training Data
Read Larger Image Convert to Grayscale
Read Template Convert to Garyscale
cv2.matchtemplate (Img,template,method)
Threshold (Highest Point)
Result
Non-max suppression (overlapthres=0.2)
Draw rectangles
Template_results (txt.file)
Comapre Results (Accuracy)

Fig 3: Flowchart of Code

The data is trained from German Traffic Sign Recognition Benchmark (GTSRB). We divide it into larger images, small images (templates) and annotation txt. file. The txt file are the actual coordinates of sign on the larger image. Then the larger image is read by using imread and convert from RGB image to grayscale image. Then the template image is read which are signs and then convert RGB image to grayscale image. After this cv2.matchtemplate is used from opencv library where template is slided over larger image from left to right and top to bottom at every pixel by using method of distance matrix. The method of distance matrix used is coefficient normalized method. Then to select the brightest point we use threshold. The threshold here used is 0.9. Then the images are stored in result matrix. But the results which are obtained are overlapped over each other. So to remove the overlaps non-max suppression method is used from imutils.object_detection. Here the overlap used is 0.2 (i.e. 20%) so if two rectangles exist between 20% only 1 rectangle is selected. Then rectangle is drawn over the traffic sign by using width and height of template. The coordinates of the rectangle are finally stored in template_results txt file. Finally this template_results txt file is compared with annotation

.txt file which are actual coordinates of traffic signs on the larger image from which accuracy is calculated.

8. CODE

a. Template Matching Code

```
import numpy as np
from matplotlib import pyplot as plt
import os
import cv2
from imutils.object_detection import non_max_suppression
from timeit import default_timer as timer

################## Function for Training DATA ########################
def read (mainFolder):
    xx = os.listdir(mainFolder)

    folder_list =[]
    file_list = []

    for name in xx:
        if ".ppm" in name:
            file_list.append(name)
        elif len(name)==2:
            folder_list.append(name)

    dataset= []

    for folder in folder_list:
        strTemp = mainFolder + folder
        xx=os.listdir(strTemp)

        count = 0
        for name in xx:
            img = plt.imread(strTemp + "//" + name)
            dataset.append([strTemp + "//" +  name, img])
            count = count+1
            if count > 10:
                break
        annotation = {}

    with open(mainFolder+"gt.txt","r") as inst:
        for line in inst:
            filename,x1,y1,x2,y2,t = line.split(";")

            if filename in annotation:
                annotation[filename].append([int(x1),int(y1),int(x2),int(y2)])
            else:
                annotation[filename] = [int(x1),int(y1),int(x2),int(y2)]

    return dataset, file_list, annotation


###################### FUNCTION FOR TEMPLATE MATCHING #################
def detect(img,template,method):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


    template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

    w, h = template.shape[::-1]

    res = cv2.matchTemplate(img_gray,template,method)
    threshold = 0.9


    loc = np.where(res>=threshold)

    results=[]
    for pt in zip(*loc[::-1]):
        results.append([pt[0] ,pt[1] ,pt[0]+w,pt[1]+ h])

    return results

############### READ DATA    ####################################
mainFolder = "C:/Users/Aniket/Desktop/a/FullIJCNN2013//"
dataset, file_list, annotation = read(mainFolder)

####################  DATASET ####################################
print("length of dataset: ",len(dataset))
################## Display COordinates and results ################
```

```
f=open("template-results.txt","w+")

################## Different Methods #############################
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
          'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']

method = eval(methods[2])

########### START TIMER /// DETECT For every 5 IMAGES ##############
for i in range(0,4): #len(file_list)):
   start=timer()
   print(file_list[i])
   img = cv2.imread(mainFolder + file_list[i])

   results = []
   for name, template in dataset:
############ TEMPLATE MATCHING    #################################
      results_temp = detect(img.copy(),template,method)

      if(len(results_temp)>0):
         results.extend(results_temp)

##########  REMOVE OVERLAP   ######################################
      rects = np.array([[x1,y1,x2,y2]for (x1,y1,x2,y2) in results])
      pick = non_max_suppression(rects, probs=None, overlapThresh=0.2)


########## Detect OBJECT ############################################
      for x1,y1,x2,y2 in pick:
         cv2.rectangle(img, (x1,y1), (x2,y2), (0,0,255), 3)
######################## Inserting data in Saving File ################
         f.write(file_list[i]+";"+str(x1)+";"+str(y1)+";"+str(x2)+";"+str(y2)+"\n")

      cv2.imshow("results", cv2.resize(img,(448,448)))
################# END TIMER ######################################
   end = timer()
   print ("elapsed time: ", end-start)
   print("results: ", pick)

################ Display IMAGE ####################################
   k = cv2.waitKey(30) & 0xff
```

```
        if k == 27:
            break

        if(len(results)>0):
            cv2.waitKey(0)

    f.close()
```

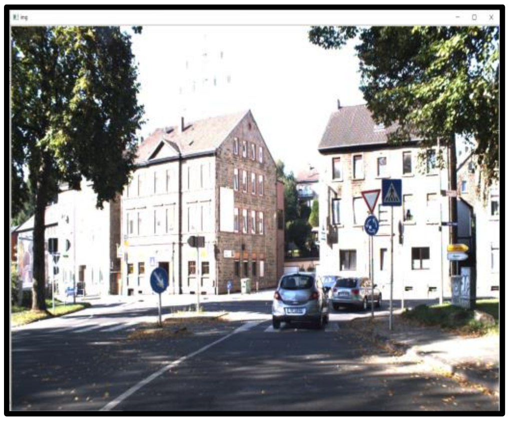
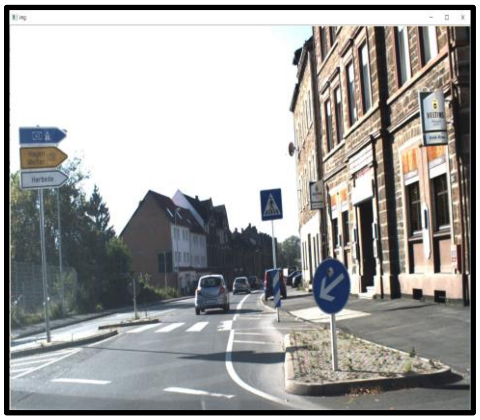
b. Accuracy Code

```
import numpy as np
import cv2
import os
from matplotlib import pyplot as plt

############## Function for Training DATA ########################

def read (mainFolder):
    xx = os.listdir(mainFolder)

    folder_list =[]
    file_list = []

    for name in xx:
        if ".ppm" in name:
            file_list.append(name)
        elif len(name)==2:
            folder_list.append(name)

    dataset= []

    for folder in folder_list:
        strTemp = mainFolder + folder
        xx=os.listdir(strTemp)

        count = 0
        for name in xx:
            img = plt.imread(strTemp + "//" + name)
            dataset.append([strTemp + "//" +  name, img])
            count = count+1
            if count > 10:
                break
        annotation = {}

    with open(mainFolder+"gt.txt","r") as inst:
        for line in inst:
            filename,x1,y1,x2,y2,t = line.split(";")

            if filename in annotation:
                annotation[filename].append([int(x1),int(y1),int(x2),int(y2)])
            else:
                annotation[filename] = [int(x1),int(y1),int(x2),int(y2)]

    return dataset, file_list, annotation
def area(a):
  x = abs(a[0]-a[2])
  y = abs(a[1]-a[3])
  return x*y
################# Function for Finding Matches ########################

def findMatches(listGT, lista):
    matchCount = 0
    missMatchCount = 0
    for x in lista:
        isMatched = False
        for gt in listGT:
              intersect = intersection(x,gt)
              if len(intersect)>0:
               r = area(intersect)
                isMatched = True

        if isMatched:
            matchCount = matchCount + 1
        else:
            missMatchCount = missMatchCount + 1

    return matchCount, missMatchCount
############### Function for Counting Rectangles #####################

def draw_rects(img, listRect, color):

    for x1,y1,x2,y2 in listRect:
        cv2.rectangle(img, (x1,y1), (x2,y2), color, 2)
```

```
   return img
########### Function for reading results from File #######################

def read_results(filename):

   results = {}
   with open(filename, "r") as ins:
      for line in ins:
         filename,x1,y1,x2,y2 = line.split(';')

         if filename in results:
            results[filename].append([int(x1),int(y1),int(x2),int(y2)])
         else:
            results[filename] = [[int(x1),int(y1),int(x2),int(y2)]]

   return results
############################ Training DATA ########################

mainFolder = "C:/Users/rajes/Downloads/FullIJCNN2013/FullIJCNN2013//"

dataset, file_list, annotation = read(mainFolder)

results = read_results("template-results.txt")
########################## Initialize Functions ########################

total = 0
totalMatched = 0
totalMissMatched = 0
totalMissed = 0
########################### Reading From File ########################

for key in annotation:

   listGT = annotation[key]

   if key not in results:
#       total = total + len(listGT)
      continue

   lista = results[key]
 ###################### Printing Output Results ########################
```

```
    matchCount, missMatchCount = findMatches(listGT, lista)
    missed = len(listGT)-matchCount
    print(key, " total:", len(listGT) ," matched: ", matchCount, " miss-matched: ",
missMatchCount, " missed: ",missed)

    img = cv2.imread(mainFolder + key)
    img = draw_rects(img, listGT, (0,0,255))
    img = draw_rects(img, lista, (255,0,0))
    cv2.imshow("img",img)
  ######################### Counting Rectangles ########################

    total = total + len(listGT)
    totalMatched = totalMatched + matchCount
    totalMissMatched = totalMissMatched + missMatchCount
    totalMissed = totalMissed + missed

    cv2.waitKey(0)


############### Printing Accuracy and Matching Result ########################

print ("total: ", total)
print("matched: ", totalMatched)

accuracy = (totalMatched / total)*100
print("accuracy (%): ", accuracy)
```

9. Input Images

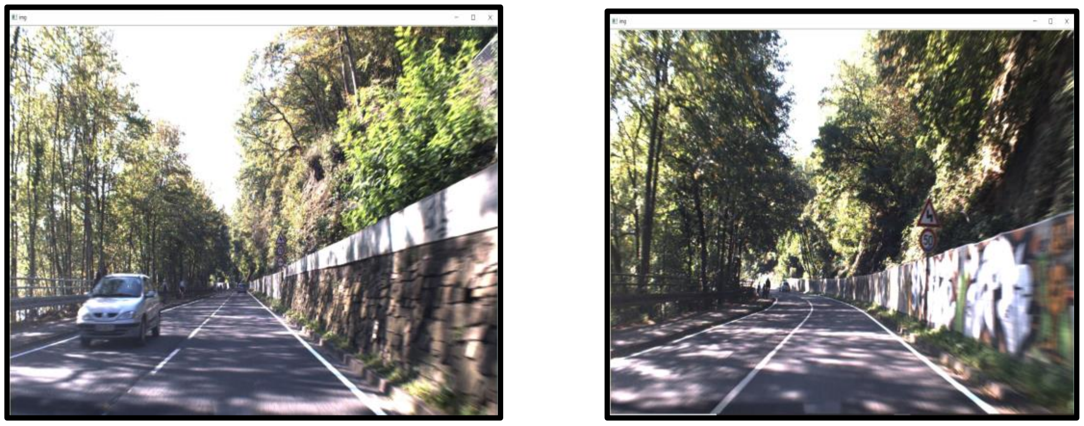


Fig 4: Input Image

10. Results

a. Traffic sign detection

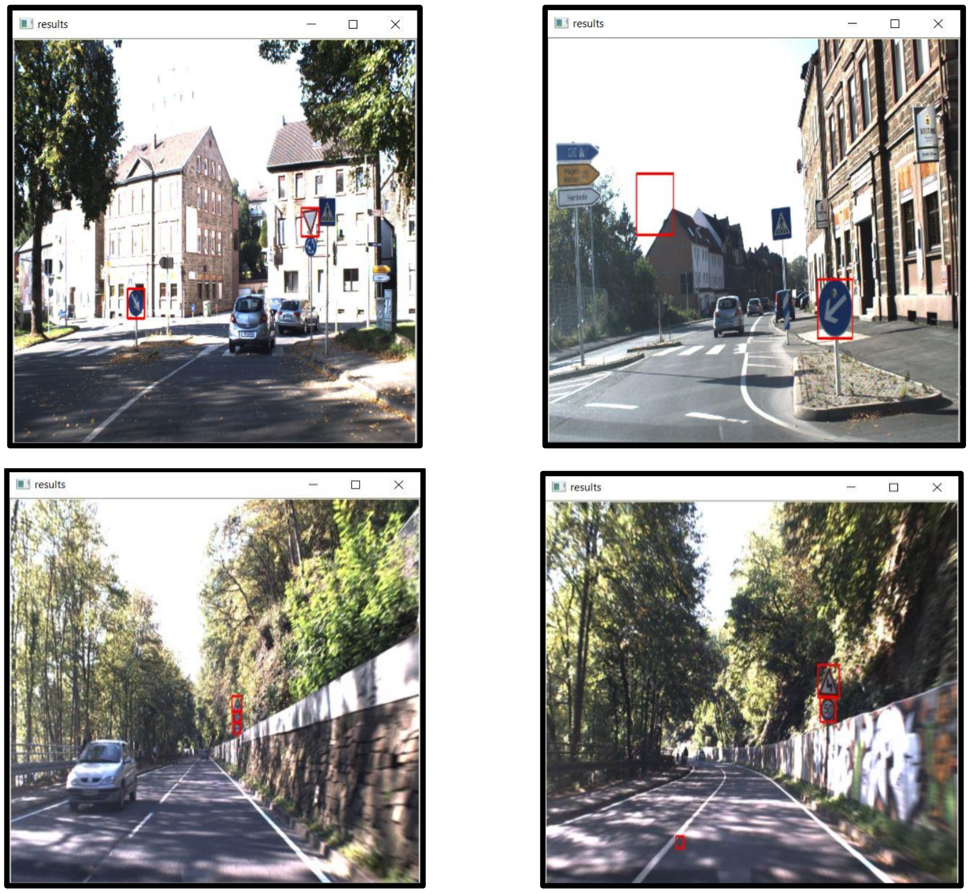


Fig 4: Traffic sign detection

b. Result Interpretation

- The algorithm successfully detects the traffic sign from the input image.
- The accuracy of the traffic sign detection was found to be around 85%.
- The computation time for detecting sign in single image is 15 seconds.
- It detects some mismatched points as shown in figure (4).

11. Challenges and Future direction

- Initially we were able to detect more mismatched features in the test image, which were rectified by varying the threshold value.
- The computation time seems to be long as 15 seconds on average, so it's quite difficult to install in real world applications specifically automobiles.
- To overcome this we can use neural network theory to reduce computation time and increase the detection accuracy.
- The commonly used neural network algorithm is Convolutional Neural Network (CNN). The figure gives the overall concept of Convolutional neural network.
- CNN is able to detect any kind of feature in the input image. For eg. If the model is trained to detect cat, it detects the cat feature wherever it is in the image.

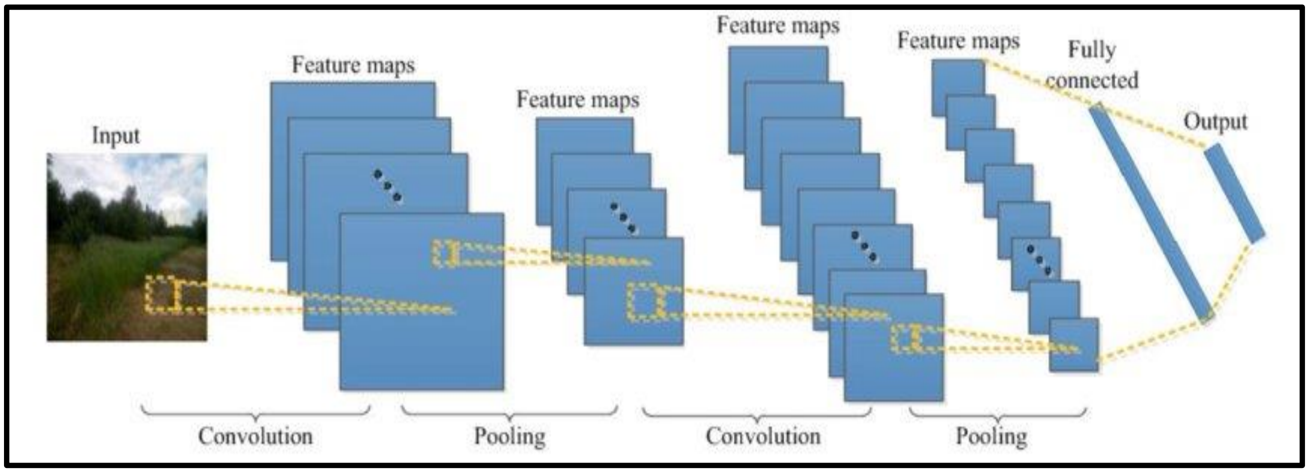


Fig 5 : Convolutional Neural Network

12. Conclusion

We detected traffic signs from the image using a template matching algorithms. We used a database called the German Traffic Sign Recognition Benchmark. We were able to correctly detect the region in the image containing the traffic sign and added a red square around that region. We then calculated the accuracy of the code by matching with the annotation in the original image present in the database. Our accuracy was 85% using the current method. The processing time was 15sec. We aim to use Convolutional Neural Network in the future to improve the result accuracy and reduce the processing time by implementing GPU accelerated Pytorch algorithms.

13. References

a. *Link of Dataset: https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.html*

b. *Code Reference: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html*

c. *ROAD AND TRAFFIC SIGN DETECTION AND RECOGNITION Hasan FLEYEH1, Mark DOUGHERTY2*

d. *Traffic Sign Detection and Recognition using Features Combination and Random Forests (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 7, No. 1, 2016*

e. *Sensors 2019, 19, 2093; doi:10.3390/s19092093*

f. *Road Sign Recognition from a Moving Vehicle Bjorn Johansson*

g. *Peyman Hosseinzadeh Kassani, Andrew BengJin Teoh, 'A new sparse model for traffic sign classification using soft histogram of oriented gradients.*

h. *Traffic Sign Detection and Recognition using Features Combination and Random Forests(IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 7, No. 1, 2016.*

i. *Vision-Based Trac Sign Detection and Recognition Systems: Current Trends and Challenges*

j. *ROAD AND TRAFFIC SIGN DETECTION AND RECOGNITION, Hasan FLEYEH1, Mark DOUGHERTY2*