WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Detection of Traffic Sign Using Convolution Neural Network

Prof. Sejal Thakkar[1], Aditya Bhadauria[2], Kartik Darji[3], Jahanav Dixit[4], Fenil Khatri[5]

[1]Assistant Professor. CE Department, Indus University, Ahmedabad, India.
[2,3,4,5] Student, B.Tech CE, Indus University, Ahmedabad, India.

ABSTRACT

In modern times, traffic sign recognition is an essential aspect of driving safety. However, developing accurate and reliable traffic sign recognition systems requires advanced technological solutions. With the increasing availability of big data and machine learning techniques, the use of convolutional neural networks (CNN) has become a popular approach to addressing this problem. In this research paper, we propose a CNN-based traffic sign recognition system that ensures high accuracy and minimal errors. By leveraging the power of CNN and big data tools, our system can effectively recognise traffic signs and contribute to safer driving practises.

1. INTRODUCTION

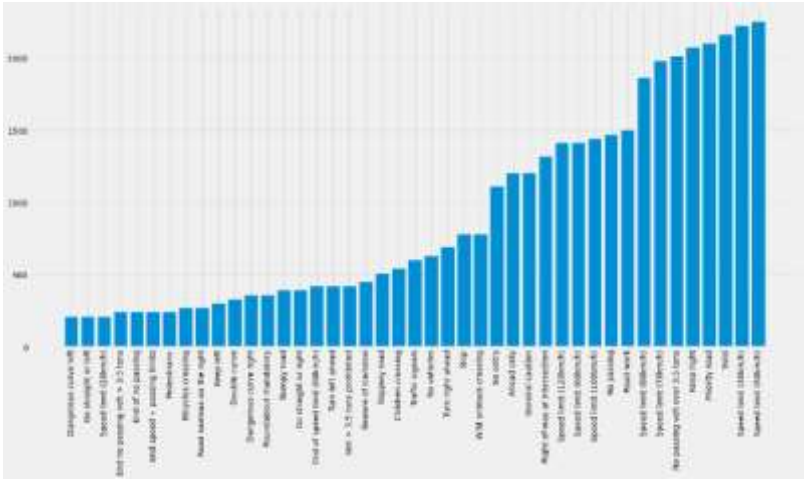
Traffic sign recognition is a crucial task in various applications, such as autonomous driving, mapping, navigation, and intelligent transportation systems. The recognition of traffic signs involves two major issues: traffic sign detection and traffic sign classification. The former aims to accurately localise the traffic signs in an image, while the latter intends to identify the labels of detected objects into specific categories and subcategories. The topic has attracted research interests in the computer vision community for many years, and it is generally regarded as challenging due to various complexities, such as the diversified backgrounds of traffic sign images. Recent advancements in deep learning have shown promising results in a range of different vision tasks, including traffic sign recognition.

In this researchpaper, we will present our implementation of a CNN-based traffic sign recognition system using the Keras library, which involves four main stages: dataset preparation and data augmentation, pre-processing, model architecture design, and model training and evaluation. We also discuss the performance of our model and provide some suggestions for future work.

The remainder of this paper is organised as follows: In Section 2, we introduce the dataset used in our project and the data augmentation techniques applied to increase the size and diversity of the training data. Section 3 presents the pre-processing techniques used to prepare the data for model training. In Section 4, we describe the design of our CNN architecture. Section 5 outlines the process of model compilation, training, and evaluation. Section 6 discusses the results and performance of our model, and Section 7 concludes the paper with some final remarks and suggestions for future research.

2. Dataset Description

The dataset used in our traffic sign recognition project is the German Traffic Sign Recognition Benchmark (GTSRB) dataset, which consists of more than 50,000 images divided into 43 classes. As shown in the histogram below, the dataset is imbalanced, with some classes having significantly fewer samples than others.



The histogram above shows the distribution of samples across the different classes in the GTSRB dataset. As we can see, some classes have more than 2000 samples, while others have less than 200 samples. This imbalance can lead to overfitting in the majority classes, resulting in poor performance in the minority classes. By applying data augmentation techniques, we increased the number of samples in each class, helping to improve the generalisation of the model and prevent overfitting.

3. Pre-Processing

Pre-processing of data is a crucial step in developing an accurate and reliable traffic sign recognition (TSR) model. Overfitting or underfitting of the model can lead to low accuracy and precision, which is unacceptable for critical tasks such as TSR. To avoid these issues, we implemented a series of pre-processing steps, including resizing, normalisation, and data augmentation. Resizing the images to a standard size and normalising the pixel values ensure the consistency and stability of the model during training. Moreover, we used data augmentation techniques, such as rotation, zooming, and shearing, to increase the diversity of the training dataset and reduce the risk of overfitting. These pre-processing steps helped us achieve high accuracy and precision in our TSR model.

3.1 Image Resizing

Image resizing for this research was necessary as the model required only the necessary data (provided in the form of image pixels here) to increase model efficiency. The GTSRB (German Traffic Sign Recognition Benchmark) dataset [1] consists of images of traffic signs with varying sizes, ranging from 15x15 pixels to 250x250 pixels. We have selected 30x30 pixels while resizing the images.

```
[ ] image_data = []
    image_labels = []

    for i in range(NUM_CATEGORIES):
        path = data_dir + '/Train/' + str(i)
        images = os.listdir(path)

        for img in images:
            try:
                image = cv2.imread(path + '/' + img)
                image_fromarray = Image.fromarray(image, 'RGB')
                resize_image = image_fromarray.resize((IMG_HEIGHT, IMG_WIDTH))
                image_data.append(np.array(resize_image))
                image_labels.append(i)
            except:
                print("Error in " + img)

    # Changing the list to numpy array
    image_data = np.array(image_data)
    image_labels = np.array(image_labels)

    print(image_data.shape, image_labels.shape)

    (39209, 30, 30, 3) (39209,)
```

Figure. Code Snippet for Resizing the images in the dataset.

3.2 Splitting data into a training set, a test set, and normalisation

Splitting the data for the project was done to assure the model's performance by evaluating it with unseen data during training. In this project, we have split the GSTRB dataset [1] into a ratio of 70:30, in the sense that 70% of the total number of images (i.e., 39199) [1] consist of the training data containing 27446 images of 30x30 pixels. The remaining 30% of the total number of images consists of test data containing 11763 images of 30x30 pixels. The shuffle parameter used in the train_test_split function is set to True, which shuffles the images before splitting, making sure that the data is not biassed on a particular subset [2].

We have also normalised the pixel data of images in both the training and test sets so that features in the image are similar in scale in both the training and evaluation stages.

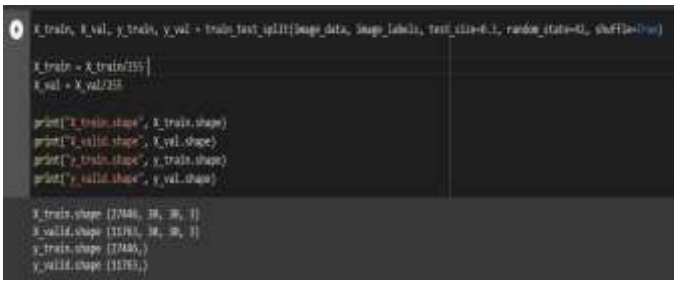


Figure. Code Snippet for train test split.

3.3 Encoding the labels

After splitting the data, we obtain both X (the input feature matrix) and Y (the target variable or label vector). We made use of the One Hot encoding for converting the values in vector Y from categorical labels to numeric format. Without one-hot encoding, the labels may be interpreted as continuous values, which may lead to inaccuracies.

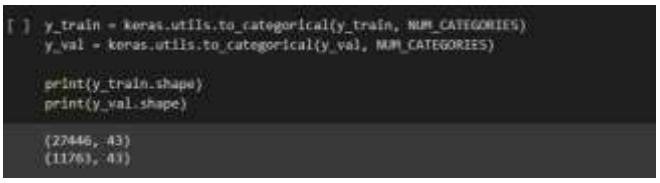


Figure. Code snippet for One – Hot encoding of labels contained in **Y** vector.

3.4 Data Augmentation

The technique of data augmentation involves using various methods to increase the size and quality of training datasets, which can lead to improved deep learning models. This survey explores a variety of image augmentation algorithms, including geometric transformations, colour space augmentations, kernel filters, mixing images, random erasing, feature space augmentation, adversarial training, generative adversarial networks, neural style transfer, and meta-learning [3].

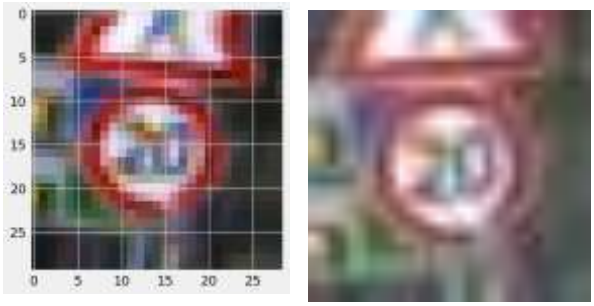


Figure : Augumented Image / Original Image from GSTRB[1].

We have applied various data augmentation techniques to the training dataset. Specifically, we used random rotations, width and height shifts, horizontal and vertical flips, and zooming to generate additional training images.

In this project, we have used the inbuilt data annotation module from Tensorflow, which generates batches of tensors from image data for training models [4].

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

Figure . Options used for data augumentation.

4. Model Architecture

Convolution is a fundamental operation in CNNs that allows the network to extract relevant features from the input images. Convolutional layers learn a set of filters or kernels that slide over the input image and perform local feature extraction through dot products. These learned features capture important visual patterns such as edges, corners, and textures and can be used for subsequent classification or regression tasks.

Pooling layers are often used in convolutional neural networks (CNNs) to reduce the dimensionality of the feature maps generated by the convolutional layers. By reducing the dimensionality, the number of parameters in the subsequent layers is reduced, making the model more efficient.

One of the benefits of pooling layers is that they help to simplify the extraction of features from the images. By taking a summary statistic of a group of neighbouring pixels, such as the maximum or average value, the pooling operation reduces the complexity of the feature maps. This can help to prevent overfitting and improve the generalisation performance of the model.

Therefore, by applying pooling layers, the usable features of the model are reduced in complexity, which can result in a more efficient and effective model for image recognition tasks.

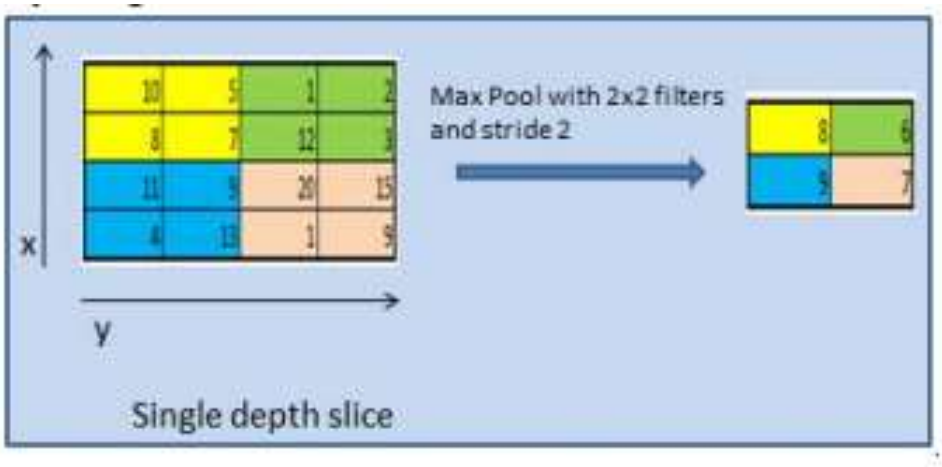


Figure . Max-Pooling Operation[5]

A fully connected layer in a neural network is a layer in which every neuron is connected to all neurons in the previous layer. This allows for high-level reasoning and abstraction. Unlike convolutional layers, which have a spatial arrangement of neurons and perform operations on local regions of the input, fully connected layers operate on the entire input. Therefore, it is not appropriate to apply a convolutional layer immediately after a fully connected layer. [5]

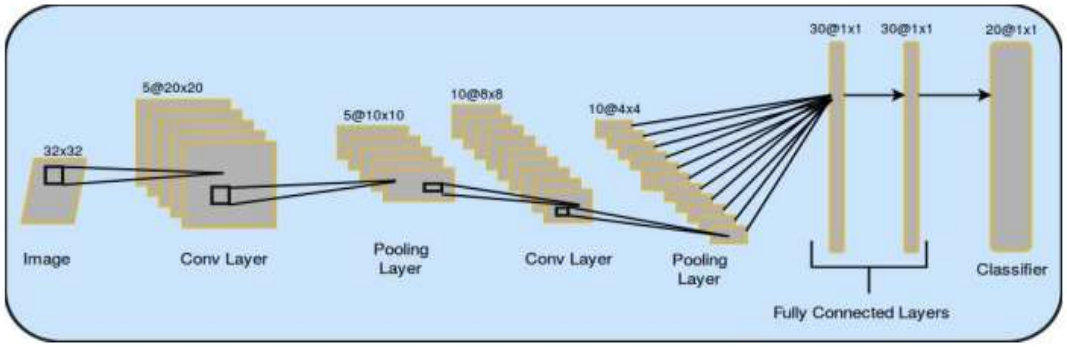


Figure . Basic ConvNet Architecture used for classification tasks [5]

In our model architecture, we have used two convolutional layers with filter sizes of 16 and 32, respectively, followed by max pooling layers. Afterward, two more convolutional layers with filter sizes of 64 and 128 have been added, followed by another max pooling layer. Batch normalisation layers are used to normalise the activations of the previous layers, and a flatten layer is used to convert the output of the convolutional layers into a one-dimensional vector. Then, a dense layer with 512 neurons and the rectified linear unit (ReLU) activation function is used, followed by a batch normalisation layer and a dropout layer with a rate of 0.5. Finally, a dense layer with 43 neurons and the softmax activation function is used as the output layer.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        448

 conv2d_1 (Conv2D)           (None, 26, 26, 32)        4640

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 batch_normalization (BatchN  (None, 13, 13, 32)       128
 ormalization)

 conv2d_2 (Conv2D)           (None, 11, 11, 64)        18496

 conv2d_3 (Conv2D)           (None, 9, 9, 128)         73856

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 batch_normalization_1 (Batc  (None, 4, 4, 128)        512
 hNormalization)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 batch_normalization_2 (Batc  (None, 512)              2048
 hNormalization)

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 43)                22059

=================================================================
Total params: 1,171,275
Trainable params: 1,169,931
Non-trainable params: 1,344
_________________________________________________________________
None
```

Figure5. Model Summary

The above model consists of 1,169,931 trainable parameters and 1,3444 non-trainable parameters for the TSR (Traffic Sign Recognition) classification model.

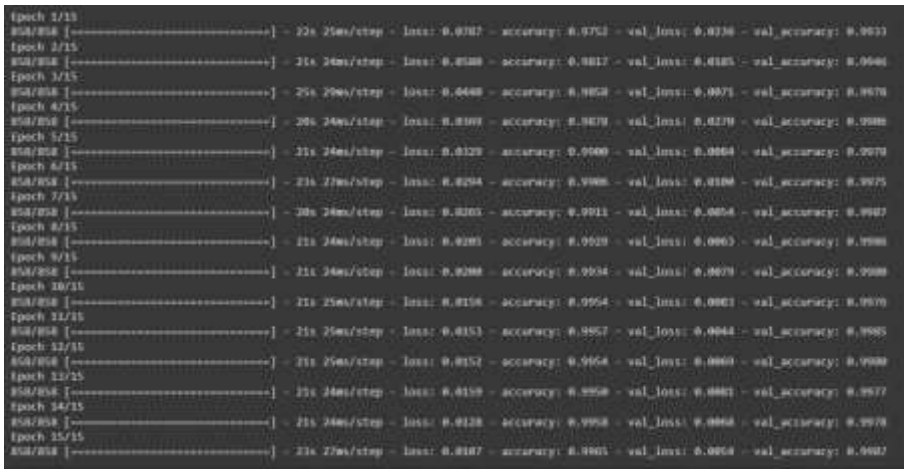
5. Model Compilation and Training

The process of compiling and training the model for traffic sign recognition involves creating a convolutional neural network (CNN) architecture using Keras, selecting an optimizer, and defining the loss function and metrics. The CNN model architecture consists of multiple layers, including convolutional layers, max pooling layers, batch normalisation layers, flattening layers, and dense layers.

The optimizer used in this model is the Adam optimizer with a learning rate of 0.001 and a decay of 0.5 over 15 epochs. The loss function used is categorical cross-entropy, and the metric used to evaluate the performance of the model is accuracy.

The data is augmented using the ImageDataGenerator class in Keras. Data augmentation techniques such as rotation, zoom, shift, shear, and flip are applied to increase the size and diversity of the training set. The model is then trained using the augmented data and the validation set. The training is performed using a batch size of 32 and 15 epochs. The history of the training process is recorded for analysis and further improvement of the model.

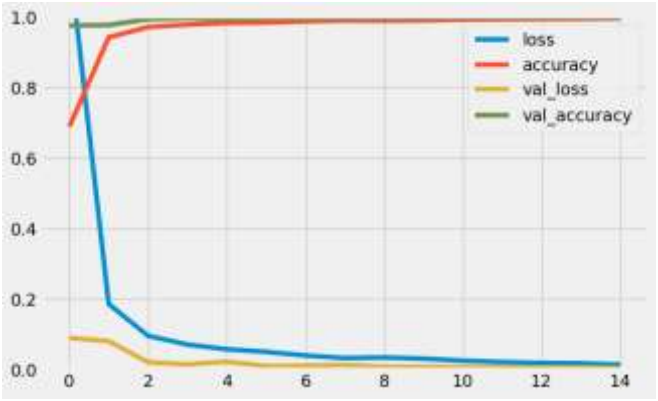
After the model was trained, it was evaluated on a separate test dataset to measure its accuracy and generalisation performance.



6. Model Evaluation

Evaluating the performance of a traffic sign recognition model is a crucial step in the process of creating a reliable and accurate system. This involves testing the model against a set of known images of traffic signs to determine its effectiveness in correctly identifying and classifying the signs. The evaluation process helps measure important metrics like precision, recall, and accuracy, which help us understand how well the model is performing. Just like a driver needs to pay attention to traffic signs on the road, a good traffic sign recognition model needs to accurately recognise signs to ensure safe driving.

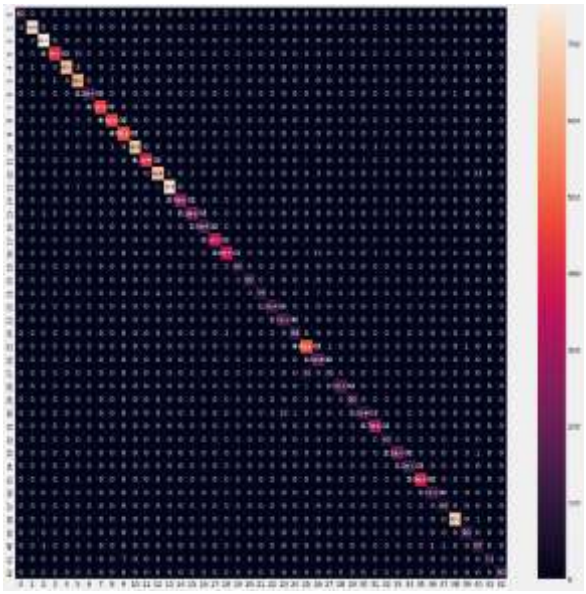
In evaluating the performance of a machine learning model, one common technique is to use visualisation tools such as plots. We are using the Pandas dataframe to plot the history of the model's training process. By visualising the model's training progress, we can assess how well the model is learning from the data and improving its accuracy over time. The figure's y-axis is limited between 0 and 1, indicating that the model's performance is being measured using a metric that ranges between 0 and 1, such as accuracy or loss. Overall, this visualisation provides an easy-to-understand representation of the model's training progress and can help us identify potential issues and areas for improvement.



We test the model on a separate dataset. By comparing the predicted labels to the actual labels in the test dataset, we can calculate the model's accuracy score, which measures how many traffic signs it correctly classified.

```
395/395 [==============================] - 1s 3ms/step
Test Data accuracy:  98.23436262866193
```

To gain a more in-depth understanding of the model's performance, we use a confusion matrix and a classification report. These tools show us how many traffic signs the model classified correctly and incorrectly and how it performed for each type of traffic sign.



	precision	recall	f1-score	support
0	1.00	1.00	1.00	60
1	0.99	1.00	1.00	720
2	0.99	1.00	0.99	750
3	1.00	0.96	0.98	450
4	1.00	0.99	1.00	660
5	0.96	0.99	0.97	630
6	1.00	0.89	0.94	150
7	1.00	1.00	1.00	450
8	0.99	0.99	0.99	450
9	0.99	1.00	0.99	480
10	1.00	1.00	1.00	660
11	1.00	0.98	0.99	420
12	0.99	0.96	0.97	690
13	1.00	1.00	1.00	720
14	0.98	1.00	0.99	270
15	0.99	1.00	0.99	210
16	1.00	1.00	1.00	150
17	1.00	0.98	0.99	360
18	0.99	0.93	0.96	390
19	1.00	1.00	1.00	60
20	0.96	1.00	0.98	90
21	0.90	0.88	0.89	90
22	1.00	1.00	1.00	120
23	0.87	1.00	0.93	150
24	0.99	0.98	0.98	90
25	0.93	1.00	0.96	480
26	0.89	1.00	0.94	180
27	0.97	0.50	0.66	60
28	1.00	0.99	1.00	150
29	0.94	1.00	0.97	90
30	0.99	0.80	0.89	150
31	0.98	1.00	0.99	270
32	1.00	1.00	1.00	60
33	1.00	0.98	0.99	210
34	1.00	1.00	1.00	120
35	1.00	1.00	1.00	390
36	0.99	1.00	1.00	120
37	0.98	1.00	0.99	60
38	1.00	1.00	1.00	690
39	0.88	1.00	0.94	90
40	0.78	0.97	0.86	90
41	1.00	0.88	0.94	60
42	0.96	1.00	0.98	90
accuracy			0.98	12630
macro avg	0.97	0.97	0.97	12630
weighted avg	0.98	0.98	0.98	12630

To visualise the model's predictions, we can create a grid of traffic sign images with their actual and predicted labels. If the model correctly predicts a traffic sign, we can label it in green text, and if it makes a mistake, we can label it in red text.

After putting in a lot of effort to train and test our traffic sign recognition model, we have successfully minimised errors and achieved the highest possible accuracy. Our model achieved an impressive final accuracy of 98.23% on the test data, indicating its high performance in recognising traffic signs. It's a great feeling to know that our model can accurately classify traffic signs, which is an important task for ensuring safe driving.

7. CONCLUSION

In this research paper, we present a CNN-based traffic sign recognition system using the German Traffic Sign Recognition Benchmark dataset. We applied various data augmentation techniques to address the issue of limited data and increased the number of samples in each class to prevent overfitting. The pre-processing techniques used to prepare the data for model training and the CNN architecture design were described in detail. The model was trained and evaluated, and the performance of our model was discussed.

The results showed that the combination of data augmentation, pre-processing, and the CNN architecture design was effective in improving the accuracy of traffic sign recognition. Our model achieved good results on the GTSRB dataset and outperformed some previous methods. Future work can focus on improving the efficiency of the algorithm by introducing parallel algorithms to speed up the process time and introducing sparsity in extracting features.

In summary, our work demonstrates the potential of deep learning-based methods for traffic sign recognition and provides insights into the design and implementation of a CNN-based traffic sign recognition system. This research contributes to the advancement of intelligent transportation systems and has practical implications for autonomous driving and other related fields.

8. REFERENCES

[1] GSTRB Benchmark

[2] Sklearn test train split

[3] Shorten, Connor, and Taghi M. Khoshgoftaar. "A survey on image data augmentation for deep learning." *Journal of big data* 6.1 (2019): 1-48.

[4] Image Generator Tensorflow

[5] Aloysius, Neena, and M. Geetha. "A review on deep convolutional neural networks." *2017 international conference on communication and signal processing (ICCSP)*. IEEE, 2017

[6] https://www.researchgate.net/profile/Zhao-Wang-11/publication/304290449_Robust_chinese_traffic_sign_detection_and_recognition_with_deep_convolutional_neural_network/links/5c9dba4545851506d731bc31/Robust-chinese-traffic-sign-detection-and-recognition-with-deep-convolutional-neural-network.pdf

[7] Traffic Sign Detection based on Convolutional Neural Networks (xlhu.cn)