

TRAFFIC SIGN RECOGNITION

Abstract: - Traffic signs are an integral part of our road infrastructure. They provide critical information, sometimes compelling recommendations, for road users, which in turn requires them to adjust their driving behavior to make sure they adhere with whatever road regulation currently enforced. Without such useful signs, we would most likely be faced with more accidents, as drivers would not be given critical feedback on how fast they could safely go, or informed about road works, sharp turn, or school crossings ahead. Traditionally, standard computer vision methods were employed to detect and classify traffic signs, but these required considerable and time-consuming manual work to handcraft important features in images. Instead, by applying deep learning to this problem, we create a model that reliably classifies traffic signs, learning to identify the most appropriate features for this problem by itself. I also got the accuracy of nearly 89% using some particular layers and algorithms.

Introduction: - Traffic signs play a crucial role in managing traffic on the road, disciplining the drivers, thereby preventing injury, property damage, and fatalities. Traffic sign management with automatic detection and recognition is very much part of any Intelligent Transportation System (ITS). In this era of self-driving vehicles, calls for automatic detection and recognition of traffic signs cannot be overstated. This paper presents a deep-learning-based autonomous scheme for cognizance of traffic signs in India. The automatic traffic sign detection and recognition was conceived on a Convolutional Neural Network (CNN). This traffic sign recognition project is done using Deep learning and its algorithms. The necessary dataset and libraries are downloaded and extracted to complete the project. I used different outputs like sequential model, maxpooling, convolutional 2D layers, flatten and dense layers are also used. An original dataset for Indian traffic signs was created using deep learning techniques. The datasets for the training and testing of the models were developed by capture of images and their segregation into different categories based on the traffic sign instances present in the image.

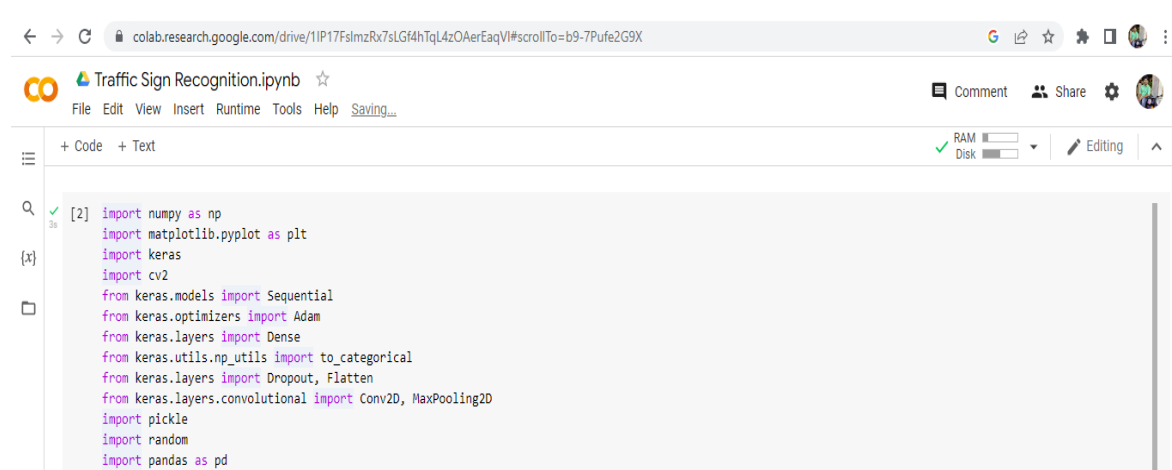
What is Deep Learning?

Deep learning is based on the branch of machine learning, which is a subset of artificial intelligence. Since neural networks imitate the human brain and so deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it is a machine learning class that makes use of numerous nonlinear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.

Since deep learning has been evolved by the machine learning, which itself is a subset of artificial intelligence and as the idea behind the artificial intelligence is to mimic the human behavior, so same is "the idea of deep learning to build such algorithm that can mimic the brain". Deep learning is implemented with the help of Neural Networks, and the idea behind the motivation of Neural Network is the biological neurons, which is nothing but a brain cell.

1. Importing the necessary libraries

We will be importing necessary libraries for building the model. The libraries are NumPy, Matplotlib, keras, convolutional layers 2D, Sequential model, optimizers like Adam, layers like Dense, Dropout, Flatten, Conv2D, MaxPooling2D, pickle, random, pandas.



```
[2] import numpy as np
import matplotlib.pyplot as plt
import keras
import cv2
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Dense
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import pickle
import random
import pandas as pd
```

2. Retrieving the images

We will retrieve the images and their labels. Then resize the images to (30,30) as all images should have same size for recognition. Then convert the images into NumPy array.

3. Dataset

We have taken the dataset from GTSRB - German Traffic Sign Recognition Benchmark

LINK: - [kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign](https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign)

The dataset consists

- Single-image, multi-class classification problem
- More than 40 classes
- More than 50,000 images in total

I also added dataset using clone tool. So that any of the dataset will run here during execution. The below screenshot contains about running dataset, training and testing the dataset.



```
[4] !git clone https://bitbucket.org/jadslim/german-traffic-signs

Cloning into 'german-traffic-signs'...
Unpacking objects: 100% (6/6), done.

[5] with open('german-traffic-signs/train.p', 'rb') as f:
    train_data = pickle.load(f)
    with open('german-traffic-signs/valid.p', 'rb') as f:
        val_data = pickle.load(f)
    with open('german-traffic-signs/test.p', 'rb') as f:
        test_data = pickle.load(f)

[6] X_train, y_train = train_data['features'], train_data['labels']
    X_val, y_val = val_data['features'], val_data['labels']
    X_test, y_test = test_data['features'], test_data['labels']

[7] print(X_train.shape)
    print(X_val.shape)
    print(X_test.shape)

(34799, 32, 32, 3)
(4410, 32, 32, 3)
(12630, 32, 32, 3)
```

✓ 0s completed at 6:03 PM

4. Images and Distribution

You can see below a sample of the images from the dataset, with labels displayed above the row of corresponding images. Some of them are quite dark so we will look to improve contrast.

5. Pre-Processing Steps

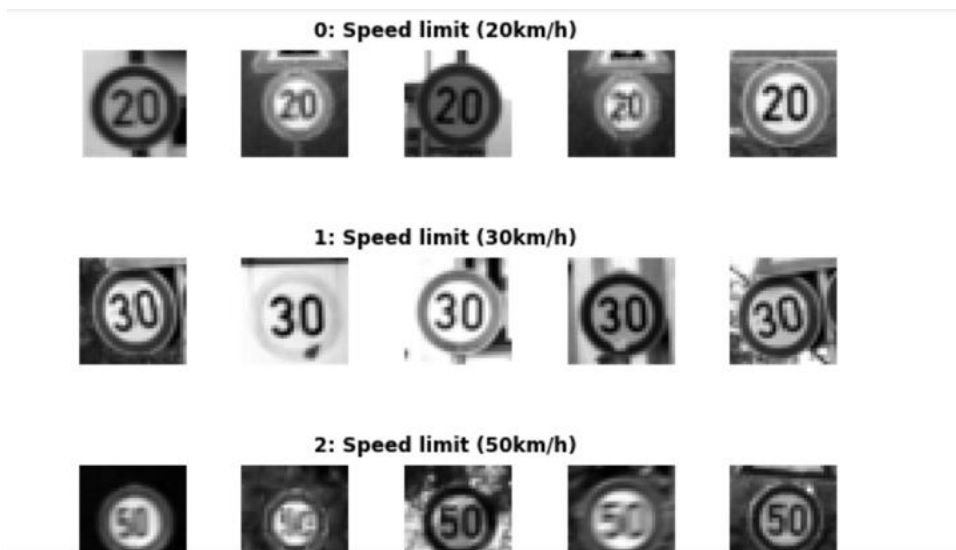
Converting images into gray color using Grayscale.

We convert our 3-channel image to a single grayscale image. We are interested in detecting white or yellow lines on images, which show a particularly high contrast when the image is in grayscale. Remember that the road is black, so anything that is much brighter on the road will come out with a high contrast in a grayscale image. The conversion from HSL to grayscale helps in reducing noise even further. This is also a necessary pre-processing step before we can run more powerful algorithms to isolate lines. Here the dataset is also trained and tested. And specific size and shape is given to all the images so that they look all together.

Colored Images in dataset: -



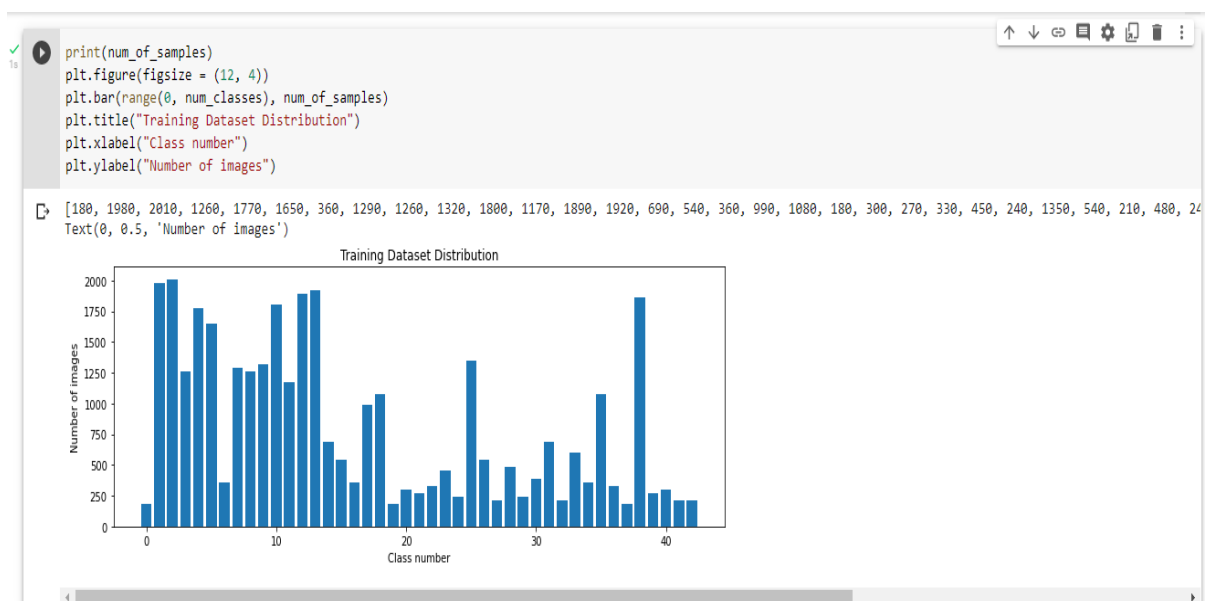
Converting the above image into Gray colored images using Grayscale: -



Data Visualization: - Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

Data Preprocessing: - Data Preprocessing is a technique that is used to convert raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

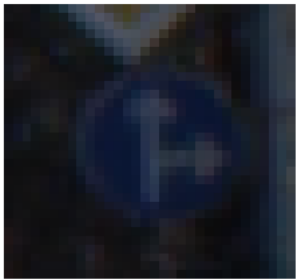
Training dataset distribution code and graph: -



Screenshot of running Data Preprocessing code: -

```
▼ Data Preprocessing
[11] plt.imshow(X_train[1000])
     plt.axis('off')
     print(X_train[1000].shape)
     print(y_train[1000])

(32, 32, 3)
36
```



```
[12] def grayscale(img):
     image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     plt.axis('off')
     return image
```

```
Traffic Sign Recognition.ipynb ☆
File Edit View Insert Runtime Tools Help Saving...

+ Code + Text

[16] img = grayscale(X_train[1000])
     plt.imshow(img, cmap = 'gray')
     print(img.shape)

(32, 32)
```



```
[13] def equalize(img):
     img = cv2.equalizeHist(img)
     return img

img = equalize(img)
plt.imshow(img, cmap = 'gray')
plt.axis('off')
print(img.shape)
```

Here, the images are divided into batches. The image is preprocessed in different types. Like the image is given zoom range, shear range, rotation range, height, width. The image size is also given using subplots.

```

[22] from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(width_shift_range = 0.1,
                             height_shift_range = 0.1,
                             zoom_range = 0.2,
                             shear_range = 0.1,
                             rotation_range = 10)
datagen.fit(X_train)

[23] batches = datagen.flow(X_train, y_train, batch_size = 20)
X_batch, y_batch = next(batches)

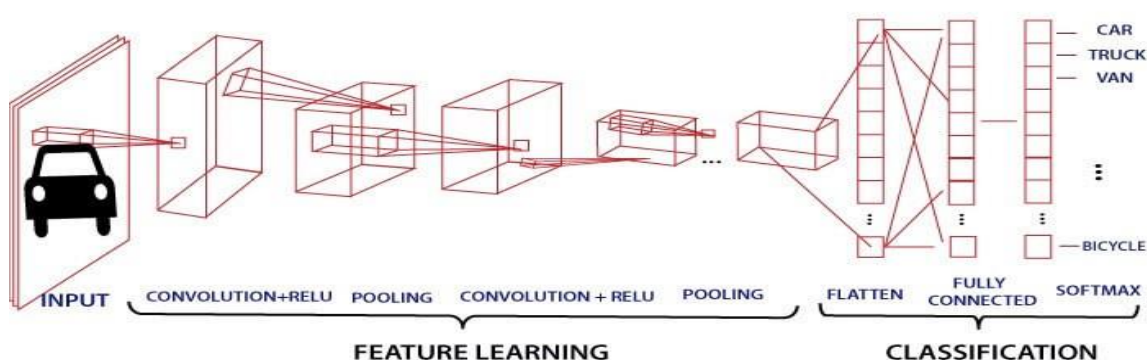
fig, axs = plt.subplots(1, 15, figsize = (20, 5))
fig.tight_layout()

for i in range(15):
    axs[i].imshow(X_batch[i].reshape(32, 32))
    axs[i].axis('off')

y_train = to_categorical(y_train, 43)
y_val = to_categorical(y_val, 43)
y_test = to_categorical(y_test, 43)

```

Convolutional Neural Network (CNN): - Convolutional neural networks are a form of artificial neural network that uses the mathematical operation convolution instead of ordinary matrix multiplication in at least one of its layers. They are employed in image recognition and processing and are especially built to handle pixel data. An input layer, hidden layers, and an output layer make a convolutional neural network. Any middle layers in a feed-forward neural network are referred to be hidden because their inputs and outputs are hidden by the activation function and final convolution. The hidden layers of a convolutional neural network include convolutional layers. Typically, this comprises a layer that does a dot product of the convolution kernel and the input matrix of the layer. This is often the Frobenius inner product, and its activation function is generally ReLU. The convolution operation generates a feature map as the convolution kernel slides along the input matrix for the layer, which then contributes to the input of the following layer. Other layers such as pooling layers, fully - connected layers, and normalization layers follow.



Max-pooling: - Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

Flattening: - Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image.

Dense Layer: - In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer. This layer is the most commonly used layer in artificial neural network networks.

The dense layer's neuron in a model receives output from every neuron of its preceding layer, where neurons of the dense layer perform matrix-vector multiplication. Matrix vector multiplication is a procedure where the row vector of the output from the preceding layers is equal to the column vector of the dense layer. The general rule of matrix-vector multiplication is that the row vector must have as many columns like the column vector.

Dropout layer: - The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons. Dropout layers are important in training CNNs because they prevent overfitting on the training data.

Rectified linear activation function or ReLU: - The usage of ReLU helps to prevent the exponential growth in the computation required to operate the neural network. If the CNN scales in size, the computational cost of adding extra ReLUs increases linearly.

Neural Network: -

Neural Network

```
[61] def neural_model():
    model = Sequential()
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(30, (3, 3), activation = 'relu'))
    model.add(Conv2D(30, (3, 3), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2, 2)))

    #model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(500, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = 'softmax'))
    model.compile(Adam(lr = 0.001), loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model
```

Summary of neural network: -

This gives the output of layers, shapes and params while using a network. It shows the values of convolutional 2D network, flatten, hidden layers values, maxpooling, dropout and dense values.

```
model = neural_model()
print(model.summary())
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 60)	1560
conv2d_5 (Conv2D)	(None, 24, 24, 60)	90060
max_pooling2d_2 (MaxPooling 2D)	(None, 12, 12, 60)	0
conv2d_6 (Conv2D)	(None, 10, 10, 30)	16230
conv2d_7 (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d_3 (MaxPooling 2D)	(None, 4, 4, 30)	0
flatten_1 (Flatten)	(None, 480)	0
dense_2 (Dense)	(None, 500)	240500
dropout_1 (Dropout)	(None, 500)	0
dense_3 (Dense)	(None, 43)	21543

=====

Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0

Epoch test: - An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine

learning model. Another way to define an epoch is the number of passes a training dataset takes around an algorithm.

```
[66] history=model.fit(X_train, y_train, epochs=10, batch_size=10)

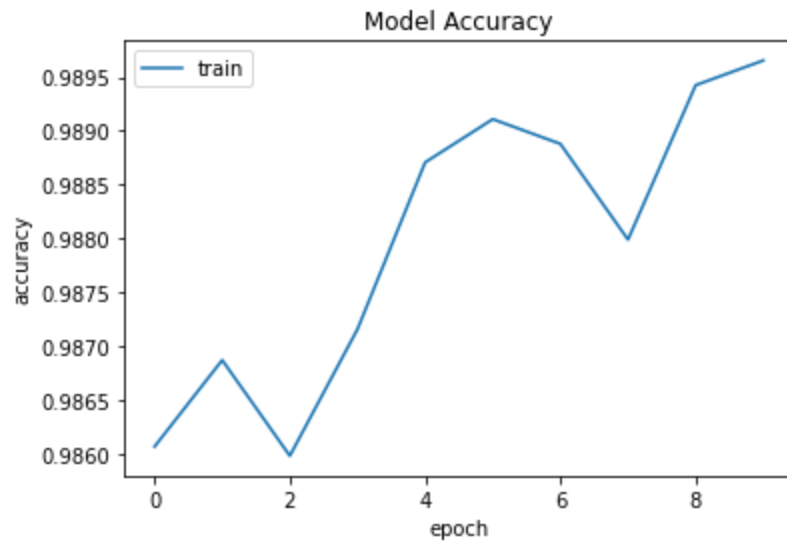
Epoch 1/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0516 - accuracy: 0.9861
Epoch 2/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0489 - accuracy: 0.9869
Epoch 3/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0520 - accuracy: 0.9860
Epoch 4/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0485 - accuracy: 0.9872
Epoch 5/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0454 - accuracy: 0.9887
Epoch 6/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0446 - accuracy: 0.9891
Epoch 7/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0456 - accuracy: 0.9889
Epoch 8/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0524 - accuracy: 0.9880
Epoch 9/10
3480/3480 [=====] - 17s 5ms/step - loss: 0.0426 - accuracy: 0.9894
Epoch 10/10
3397/3480 [=====>.] - ETA: 0s - loss: 0.0439 - accuracy: 0.9895
```

Quantitative Analysis (Graphs) :-

Printing the graph of Accuracy

```
history.history??

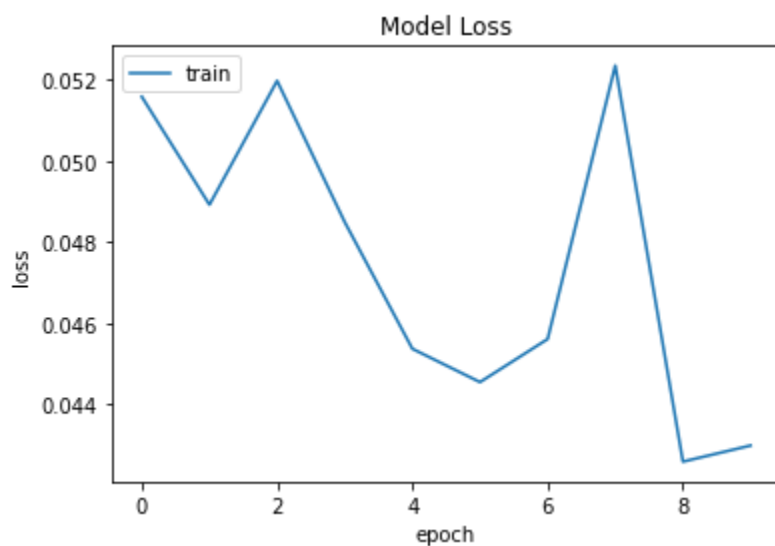
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



Printing the Graph of Loss: -

```
plt.plot(history.history['loss'])  
plt.title('Model Loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train'], loc='upper left')  
plt.show()
```

OUTPUT: -



Test Score and Test Accuracy Output: -

```
[ ] score = model.evaluate(X_test, y_test, verbose = 1)
    print('Test Score', score[0])
    print('Test Accuracy', score[1])

395/395 [=====] - 2s 4ms/step - loss: 0.3794 - accuracy: 0.9508
Test Score 0.3794045150279999
Test Accuracy 0.9508313536643982
```

Testing: -

▼ Testing

```
import requests
from PIL import Image
url = 'https://c8.alamy.com/comp/A0RX23/cars-and-automobiles-must-turn-left-ahead-sign-A0RX23.jpg'
r = requests.get(url, stream=True)
image = Image.open(r.raw)
plt.axis('off')
plt.imshow(image, cmap=plt.get_cmap('gray'))
```

<matplotlib.image.AxesImage at 0x7f9791acb4d0>



Data Augmentation: - Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples. Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image. Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more.

Some of the most common data augmentation techniques used for images are:

Position augmentation

- Scaling

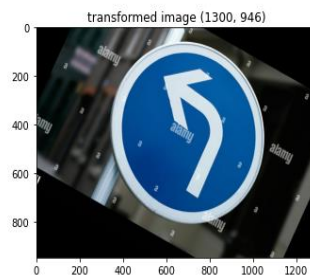
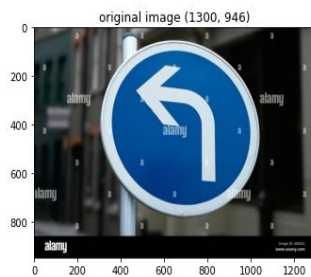
- Cropping
- Flipping
- Padding
- Rotation
- Translation
- Affine transformation

POSITION AUGMENTATION

```
[30] import PIL.Image
import matplotlib.pyplot as plt
import torch
from torchvision import transforms
def imshow(img, transform):
    img = PIL.Image.open(img)
    fig, ax = plt.subplots(1, 2, figsize=(15, 4))
    ax[0].set_title(f'original image {img.size}')
    ax[0].imshow(img)
    img = transform(img)
    ax[1].set_title(f'transformed image {img.size}')
    ax[1].imshow(img)
```

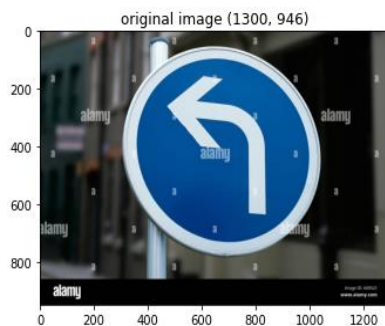
ROTATION

```
[33] loader_transform = transforms.RandomRotation(95)
imshow('direction.jpg', loader_transform)
```



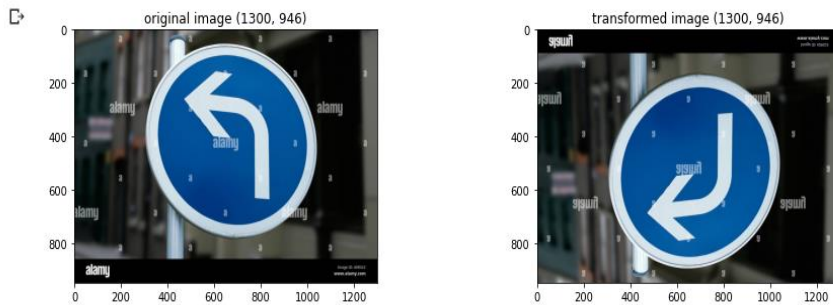
SCALING

```
[31] loader_transform = transforms.Resize((400, 400))
imshow('direction.jpg', loader_transform)
```



FLIPPING

```
loader_transform = transforms.RandomVerticalFlip(p=1)
imshow('direction.jpg', loader_transform)
```



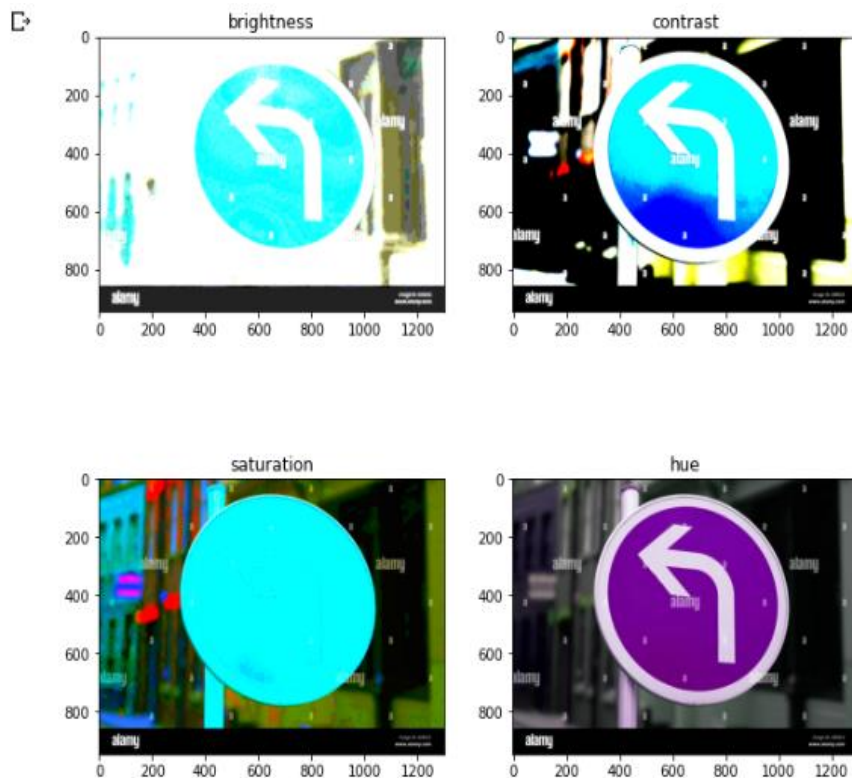
Color Augmentation: - Another augmentation method is changing colors. We can change four aspects of the image color: brightness, contrast, saturation, and hue. In the example below, we randomly change the brightness of the image.

- Brightness
- Contrast
- Saturation
- Hue

Code for Color Augmentation: -

COLOR AUGMENTATION

```
[34] img = PIL.Image.open('direction.jpg')
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
#BRIGHTNESS
loader_transform1 = transforms.ColorJitter(brightness=50)
img1 = loader_transform1(img)
ax[0, 0].set_title(f'brightness')
ax[0, 0].imshow(img1)
#CONTRAST
loader_transform2 = transforms.ColorJitter(contrast=50)
img2 = loader_transform2(img)
ax[0, 1].set_title(f'contrast')
ax[0, 1].imshow(img2)
#SATURATION
loader_transform3 = transforms.ColorJitter(saturation=25)
img3 = loader_transform3(img)
ax[1, 0].set_title(f'saturation')
ax[1, 0].imshow(img3)
fig.savefig('color augmentation', bbox_inches='tight')
#HUE
loader_transform4 = transforms.ColorJitter(hue=0.5)
img4 = loader_transform4(img)
ax[1, 1].set_title(f'hue')
ax[1, 1].imshow(img4)
fig.savefig('color augmentation', bbox_inches='tight')
```



Research gaps in traffic sign recognition using Machine Learning (ML) and Deep Learning (DL) can be identified in several areas:

1. **Robustness to Variations:** Current models often struggle with variations in illumination, weather conditions, occlusions, and sign degradation over time. Research can focus on developing more robust models that generalize well across diverse real-world conditions.
2. **Small Data and Few-shot Learning:** Gathering labeled data for all possible traffic signs can be challenging and expensive. Exploring techniques like few-shot learning, transfer learning, and semi-supervised learning could help in effectively utilizing limited annotated data.
3. **Real-time Performance:** Many applications require real-time processing of traffic signs (e.g., in autonomous vehicles). Improving the efficiency of models to achieve real-time performance without compromising accuracy is a significant research challenge.
4. **Adaptability to New Sign and Environments:** Models should be able to adapt quickly to new traffic signs or changes in existing signs, as well as different geographical environments where signs may vary in appearance or meaning.

5. Interpretability and Explainability: Deep learning models, especially convolutional neural networks (CNNs), are often considered black boxes. Research can focus on making these models more interpretable and explainable, especially in safety-critical applications like autonomous driving.

6. Edge Computing and Resource Constraints: Deploying traffic sign recognition systems on edge devices with limited computational resources poses a challenge. Optimizing models for deployment on low-power devices while maintaining accuracy is an emerging area of research.

8. Privacy and Security: Ensuring the privacy and security of data collected by traffic sign recognition systems, especially in the context of connected vehicles and smart cities, requires robust methods for data anonymization and protection against adversarial attacks.

9. Benchmark Datasets and Evaluation Metrics: Developing standardized benchmark datasets and evaluation metrics specific to traffic sign recognition can facilitate fair comparisons between different models and methodologies.

10. Human-Centric Design: Considering human factors such as the understanding and interpretation of traffic signs by drivers and pedestrians can lead to more effective design and deployment of recognition systems.

Addressing these research gaps can contribute significantly to the development of more accurate, reliable, and practical traffic sign recognition systems using ML and DL techniques.

Conclusion: -

We conclude that on implementing Traffic Sign Recognition using deep learning algorithms, I have understood the basic concepts of deep learning, Convolutional neural networks, Layers like hidden layers, Relu layer, dense layers. We also understood how to install different types of libraries that is used to work in the project. And we also learnt about the implementation using codes. And finally, after running the neural networks we have got the maximum accuracy of 94% after data cleaning but we will work forward to increase this accuracy. Thus, this project helps us in understanding the methods in Machine Learning and deep learning.

References: -

<https://towardsdatascience.com/traffic-sign-recognition-using-deep-neural-networks-6abdb51d8b70>

<https://iq.opengenus.org/data-augmentation/>

<https://analyticsindiamag.com/a-beginners-guide-to-neural-network-pruning/>

<https://www.analyticsvidhya.com/blog/2021/12/traffic-signs-recognition-using-cnn-and-keras-in-python/>

<https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/code>

<https://www.javatpoint.com/artificial-neural-network>