



AI-341 PROJECT REPORT

TRAFFIC SIGNS RECOGNITION BY USING CNNs

Members:

- Ahsham Abdullah -2020057
- Ahmad Hassan -2020053
- M. Bin Murad -2020904

Introduction:

Traffic Signs are used to maintain the stable flow of traffic and to prevent disruption of traffic and accidents. These traffic signs should be recognizable by the computers for the traffic safety cameras functioning and moreover the autonomous vehicles must also recognize these signs to ensure the smooth journey of driver-less cars. In our project, we have made our computer recognize the traffic signs to give road sense to the computer.

Problem Statement:

For this project, we must train the model on some sample traffic sign images in order to train our machine and we use these images as our training data to train our Convolutional Neural Network. After the training, we can feed any image to the model, and it will predict what sign category does it belongs to.

Dataset:




- We have used the [German Traffic Signs Recognition Benchmark \(GTSRB\)](#) dataset in order to train our model. The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. We cordially invite researchers from relevant fields to participate: The competition is designed to allow for participation without special domain knowledge. Our benchmark has the following properties:
- Single-image, multi-class classification problem
- More than 40 classes
- More than 50,000 images in total
- Large, lifelike database

This dataset gives us more than enough signs to train our model according to modern day traffic requirements and make it compatible enough to work on real-life data.



Figure represents some of the training samples

Our dataset also contains csv files maintaining the metadata i.e., the information of the images in training data, given as

▲ Path	🔍 ClassId	🔍 ShapeId	🔍 ColorId	▲ SignId
Path to image	Image class ID	Shape of sign (0-red, 1-blue, 2-yellow, 3-white)	Color of sign (0-triangle, 1-circle, 2-diamond, 3-hexagon, 4-inverse triangle)	Sign ID (by Ukrainian Traffic Rule)
43 unique values				<div>3.29 19%</div> <div>3.3 5%</div> <div>Other (33) 77%</div>
Meta/27.png	27	0	0	1.32
Meta/0.png	0	1	0	3.29
Meta/1.png	1	1	0	3.29
Meta/10.png	10	1	0	3.27
Meta/11.png	11	0	0	1.22
Meta/12.png	12	2	2	2.3
Meta/13.png	13	4	0	2.1

Methodology:

1. We used the Convolutional Neural Network to train on these images. First, we assign classes to each category so it would be easy to classify them.
2. We load these images and then normalize them i.e., divide the RGB values of the images by 255 to normalize data between zero and one. This is one of the key steps of our data preprocessing.
3. Next step, we train our Convolutional Neural Network, we used multiple layers of different filter sizes and neurons to train our model. We used the 'softmax' activation function for the output layer and it has 43 classes so it predicts the probability of each class as output, the output with highest predicted probability will be the output given from the model.

Training the Convolutional Neural Network

```
In [105]: model = keras.models.Sequential([
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), activation='relu', input_shape=(height,width,3)),
    keras.layers.Conv2D(filters=32, kernel_size=(5,5), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),
    keras.layers.Dropout(rate=0.25),
    keras.layers.Flatten(),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.25),

    keras.layers.Dense(43, activation='softmax')
])
```

4. We then prepare and preprocess the test data in order to feed it to our trained model. The test dataset is then tested and it gives an accuracy of around 80%.

```
]: def map_pred(pred):
    return [int(list(train_dataset.class_indices.keys())[i]) for i in pred]
test_df = pd.read_csv('Test.csv')

test_labels = test_df["ClassId"].values
imgs = test_df["Path"].values
data = []

for img in imgs:
    try:
        image = cv2.imread(test_path + img)
        image_fromarray = Image.fromarray(image, 'RGB')
        resize_image = image_fromarray.resize((height, width))
        data.append(np.array(resize_image))
    except:
        print("Error in " + img)
X_test = np.array(data)
X_test = X_test/255
```

Figure showing test data preprocessing

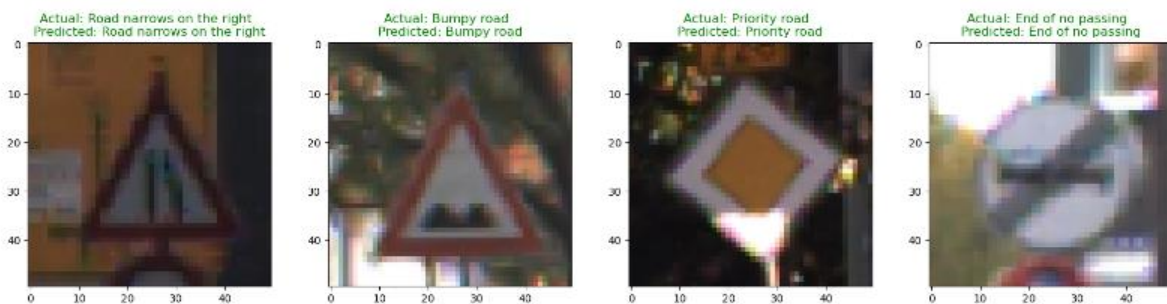
Results:

We come to the result that we get nearly 80% accuracy on test data and predict our results on test data .

```
: from sklearn.metrics import accuracy_score
pred = model.predict(X_test)
pred = map_pred(pred.argmax(axis=-1))
print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)

395/395 [=====] - 7s 17ms/step
Test Data accuracy:  78.701504354711
```

After this, we displayed some samples from test data with their actual labels and the predicted labels just to compare them as follows.



Conclusion:

In conclusion, we came to know that CNNs are great predictors for traffic signs and have achieved great accuracy. But in order to apply them to a larger scale, i.e., on roads for practical purposes, we have to use Regional CNN algorithms or YOLO as they can detect such signs from the whole image and identify them from a larger image containing different objects. Otherwise, CNN works fine for sign predictions.

[GitHub Repository Link](#)