Traffic Sign Detection In Foggy Weather

Aditya Raj
Ahmedabad University
AU1920177

Dhruvanshu Parmar
Ahmedabad University
AU194016

Abstract— This project presents a traffic sign detection and recognition system based on a Convolutional neural network. In this project we are using EfficientNetB4 neural network architecture to create a model based on our own custom dataset. Training model in this project involves preprocessing the images, so that it can handle the system even in the foggy weather condition.

This project is focused on two subproblems first, since the data in the foggy condition is hard to get for the training, steps are taken to mitigate the problem by manipulating the images. And the second subproblem is repeating the process of training and testing to achieve the accepted result.

I. Introduction

In this project we are trying to detect traffic signs and recognize them as well and we are going further in this project and taking care that it can also be recognized in the foggy weather condition. To implement this detection and recognition we need a model which can be trained using machine learning algorithms like SVM(support vector machine) or using deep neural networks. Since we are detecting the traffic sign in foggy conditions, chances of detection become low but we are trying to increase the accuracy as much as we can.

II. Literature survey

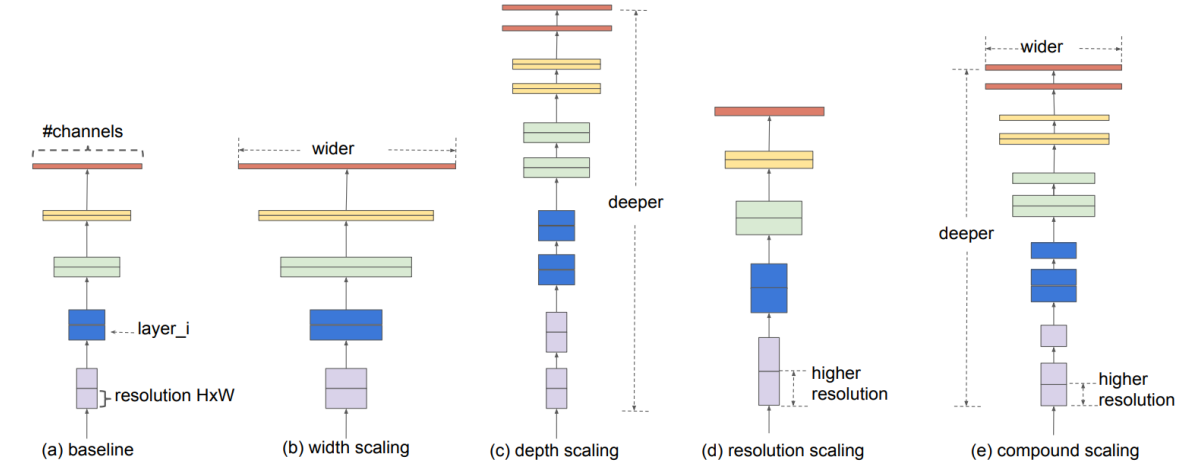
We Explored different Models for object detection which can be fit for our project. The models which we explored are:

- MobileNet
- RetinaNet
- Faster-RCNN
- SqueezeNet
- GhostNet
- SSD (Single Shot detection)

In these Models we are mainly focused on studying the mechanism, model size, and Inference Time of the model. We Come to the conclusion that depending on the depth of the neural network model size and inference time depends.

we are able to explore efficientNet neural network architecture developed by google AI. After Reading the Official Paper on *EfficientNet* [1], we are convinced that we are going to use this architecture in our Project.

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks



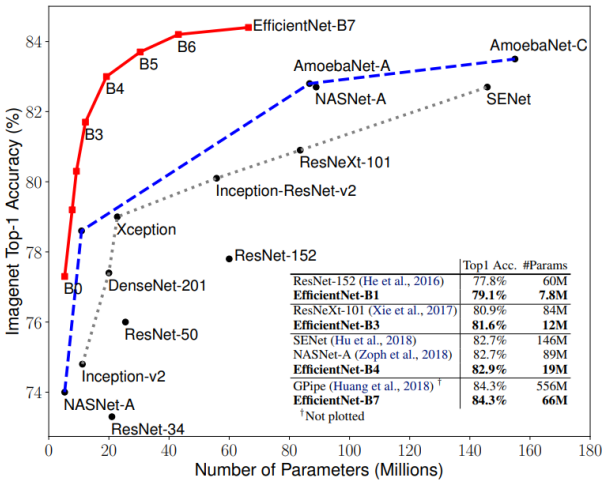


Fig: Model vs. ImageNet Accuracy

In this paper[1] they systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, they propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient.

They propose a new compound scaling method, which use a compound coefficient φ to uniformly scales network width, depth, and resolution in a principled way:

$$\text{depth: } d = \alpha^{\varphi}$$
$$\text{width: } w = \beta^{\varphi}$$
$$\text{resolution: } r = \gamma^{\varphi}$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

where α, β, γ are constants that can be determined by a small grid search. Intuitively, φ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively.

.

III. Implementation.

The idea behind implementation is that first, acquire appropriate data for your model, in our case it is similar to fogged traffic signs. Second, preprocess the data for input, in another word every model has their own minimum requirement like shape of the image, so to accommodate that difference we need to create a custom dataset particular for that model. Third, selecting the optimizer and the loss function for your model, in this case we need trial and error to get which optimizer and loss function is working best with your model. And last is to start training your model and evaluate your model and make changes accordingly.

Now, The implementation we are following is exactly the same as explained above. So we will start by data

Data

The CURE-TSR[2] (challenging unreal and real environments for traffic sign recognition) dataset is taken from IEEE DataPort.

Fig: Types of traffic sign in dataset

Traffic sign images in the CURE-TSR dataset were cropped from the CURE-TSD[3] dataset, which

incorporates around 1.7 million real-world images and simulator images with more than 2 million traffic sign instances. Real-world images were obtained from the BelgiumTS video sequences and the simulated images were generated with the Unreal Engine 4 game development tool. Sign types include speed limit, goods vehicles, no overtaking, no stopping, no parking, stop, bicycle, hump, no left, no right, priority to, no entry, yield, and parking. Unreal and real sequences were processed with visual effect software Adobe(c) After Effects to simulate challenging conditions, which include rain, snow, haze, shadow, darkness, brightness, blurriness, dirtiness, colorlessness, sensor and codec errors.

Total size of the unzipped data is around 10GB. There are 14 types of traffic signs with different challenging conditions. From the dataset we only took the Haze and GaussianBlur challenging condition data for our dataset because we are only simulating foggy weather conditions.

Data Preprocessing

For our purpose we need the image which is similar to foggy in nature so, from the large dataset we have, we have taken GaussianBlur and Haze challenging condition images from it.

In this step, we needed to create the dataset for the model EfficientNetB0, in this model the specific requirement that we needed is that image shape should be (224 x 224). To achieve this we took advantage of the image_dataset_from_directory function of kearas which takes images from a folder in a specific format of hierarchy to recognize classes and convert it into tensor for model input.

Optimizer and loss function

Due to time constraint we are only able to test on only two combinations, first is Adam optimizer and mean square error loss function. And Second is Adam optimizer and CategoricalCrossentropy loss function. Mean Square error is a regression loss function and CategoricalCrossentropy is a probabilistic loss function.

Training and testing

Due to very large data and less GPU time available on google colab we are able train for few epochs for the above combination of optimizer and loss function. The model we used here is EfficientNetB0, this model takes less computational power and less inference time according to the official paper. For this it can be used for mobile devices which has low computational power

IV. Results

After training the model we have following accuracy and predictions:

```
# img_path = '/content/data/testing_data/no_entry/01_12_07_02_0039.bmp'
img_path = '/content/data/testing_data/parking/01_14_07_01_0340.bmp'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = tf.keras.applications.efficientnet.preprocess_input(x)

preds = model.predict(x)
decode_prediction(preds)
```

```
bicycle :  3.6252032e-08
goods_vehicles :  2.823617e-09
hump :  2.8937202e-06
no_entry :  9.3588545e-07
no_left :  2.9769234e-09
no_overtaking :  3.4253966e-10
no_parking :  5.4118814e-06
no_right :  7.3399214e-08
no_stopping :  6.1461475e-10
parking :  0.9999869
priority_to :  1.16485204e-07
speed_limit :  1.697523e-06
stop :  1.9330953e-06
yield :  6.275533e-08
predicted output is : parking
```

Above result is showing the prediction of a blurred image of parking and the statistics of the other classes. From the last line we can observe that the predicted output is parking and the model is correctly predicted.

```
img_path = '/content/data/testing_data/no_entry/01_12_07_02_0039.bmp'
# img_path = '/content/data/testing_data/parking/01_14_07_01_0340.bmp'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

x = tf.keras.applications.efficientnet.preprocess_input(x)

preds = model.predict(x)
decode_prediction(preds)
```

```
predicted output is : no_entry
```

This is the prediction with another image and the model is correctly predicting even with very low quality of images.

V. Conclusion

For our project here we used 72918 images of 14 classes to train our EfficientNetB0 models and 33340 images for validation. The fourteen classes are speed limit, goods vehicles, no overtaking, no stopping, no parking, stop, bicycle, hump, no left, no right, priority to, no entry, yield, and parking. Upon training we used two loss functions: first is mean square error (regression loss function), and second is CategoricalCrossentropy (probabilistic loss function). Model is correctly predicting even with very low quality of images.

References

[1] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in International conference on machine learning PMLR, 2019, pp. 6105–6114.

[2] Dogancan Temel, Gukyeong Kwon, Mohit Prabhushankar, Ghassan AlRegib, October 10, 2019, "CURE-TSR: Challenging Unreal and Real Environments for Traffic Sign Recognition", IEEE Dataport, doi: https://dx.doi.org/10.21227/n4xw-cg56.

[3] Dogancan Temel, Tariq Alshawi, Min-Hung Chen, Ghassan AlRegib, October 13, 2019, "CURE-TSD: Challenging Unreal and Real Environment for Traffic Sign Detection", IEEE Dataport, doi: https://dx.doi.org/10.21227/en9z-mq69.