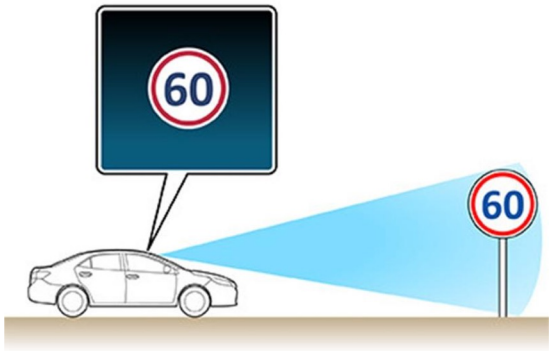
Traffic Sign Classification

Sahir Doshi and Yiran Wang

Problem Statement

Aim: How can we build a machine learning model that can classify traffic signs?

Inspiration: Interested in exploring how image classification works beyond only cats and dogs + wanted to test something more advanced and applicable to the real-world (i.e, autonomous electric vehicles)



Methods

1) General Data Set Up - packages, hyperparameters, input, splitting, one-hot encoding
2) CNN with Data Augmentation
3) CNN with Grayscale
4) CNN with Data Augmentation and Grayscale
5) CNN with Data Augmentation, Grayscale, and Random Noise
6) CNN with Data Augmentation, Grayscale, Random Noise, and Autofill
7) Transfer Learning using VGG16 network

Challenge

Problem: Could not explore original proposal extension idea

Solution: Focus on small differences between models using varying preprocessing functions and/or the presence of data augmentation

Results

Model	Loss Score	Accuracy
CNN w/ Data Augmentation	0.252	0.923
CNN w/ Grayscale	0.111	0.988
CNN w/ Data Augmentation and Grayscale	0.356	0.900
CNN w/ Data Augmentation, Grayscale, Random Noise	0.212	0.942
CNN w/ Data Augmentation, Grayscale, Random Noise, Autofill	0.342	0.928
CNN w/ Transfer Learning	0.048	0.985

CNN w/ Grayscale

Reduction in Complexity: Converting images to grayscale reduces the complexity of the input data.

Focus on Shape and Texture: Traffic sign recognition often relies more on the shape and texture of the sign than on its color.

Robustness to Variations in Lighting and Color: Color images can vary significantly under different lighting conditions, which can affect the performance of a CNN if it heavily relies on color features.

Conclusion

By the metric of accuracy, best model was a

CNN with Grayscale Only and CNN with Transfer Learning

Potential Reasons for Why:

1) Dataset is too easy to train on
2) The usage of two dense layers

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 96, 96, 60)        1560

 conv2d_10 (Conv2D)          (None, 92, 92, 60)        90060

 max_pooling2d_6 (MaxPoolin  (None, 46, 46, 60)        0
 g2D)

 conv2d_11 (Conv2D)          (None, 44, 44, 30)        16230

 max_pooling2d_7 (MaxPoolin  (None, 22, 22, 30)        0
 g2D)

 dropout_6 (Dropout)         (None, 22, 22, 30)        0

 flatten_3 (Flatten)         (None, 14520)             0

 dense_6 (Dense)             (None, 500)               7260500

 dropout_7 (Dropout)         (None, 500)               0

 dense_7 (Dense)             (None, 58)                29058

=================================================================
Total params: 7397408 (28.22 MB)
Trainable params: 7397408 (28.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Having Two Dense layers

The first dense layer captures high-level features and the second transforms these features into final outputs.

This separation results in better learning and generalization, because the model can refine its understanding of the data in progressive steps.

dense_6 (Dense)	(None, 500)	7260500
dropout_7 (Dropout)	(None, 500)	0
dense_7 (Dense)	(None, 58)	29058

Fun Fact

Some autonomic electric vehicles have machine learning algorithms that are so complex that they can sometimes detect brand new images in real-time using a technique called **online learning**. This can be done without the car's software being manually updated.