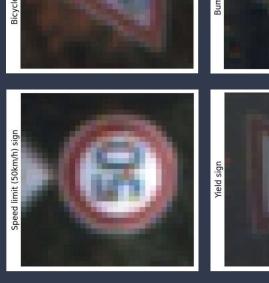
# Recognizing Traffic Signs

**Evan Hollier** 

### Problem







The goal of this project is to build a model capable of determining the type of traffic sign that is displayed in an image captured under different real-life conditions and showing obstructions, poor lighting, or even the sign being far away from the camera.

32x32 images

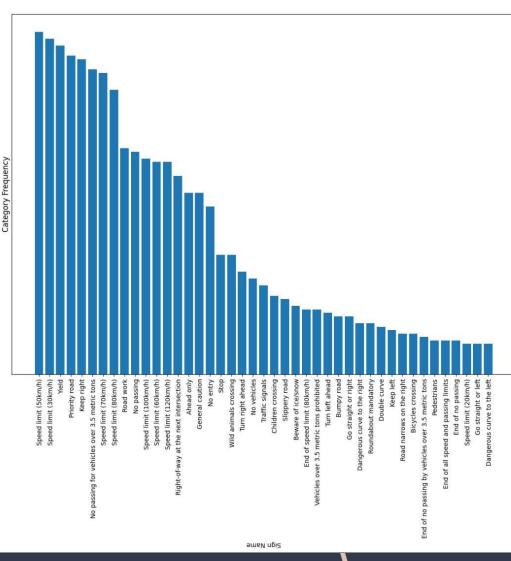
43 labels

34,799 train, 4,410 validation, 12,630 test  $\sim$  67-9-24 %

51,839 total

Dataset itself is clean

#### Yield -Priority road -Keep right -No entry -Stop -Wild animals crossing Turn right ahead Go straight or left Dangerous curve to the left Speed limit (50km/h) Speed limit (30km/h) No passing for vehicles over 3.5 metric tons Speed limit (70km/h) Speed limit (80km/h) Road work No passing Speed limit (100km/h) Speed limit (60km/h) Speed limit (120km/h) Right-of-way at the next intersection Ahead only General caution No vehicles Traffic signals Children crossing Slippery road End of speed limit (80km/h) Vehicles over 3.5 metric tons prohibited Turn left ahead Bumpy road Go straight or right Dangerous curve to the right Roundabout mandatory Double curve Keep left Road narrows on the right Bicycles crossing End of no passing by vehicles over 3.5 metric tons Pedestrians End of all speed and passing limits Speed limit (20km/h) Beware of ice/snow End of no passing Sign Name Data distribution Looks somewhat unbalanced Smallest category ~0.5% Largest category ~5.8%



3000

2500

2000

1500 Frequency

1000

200

## **Model Preparation**

Images needed to be upscaled in order to be fed into

pre-trained models - 128x128

One-hot encode labels

### Available models

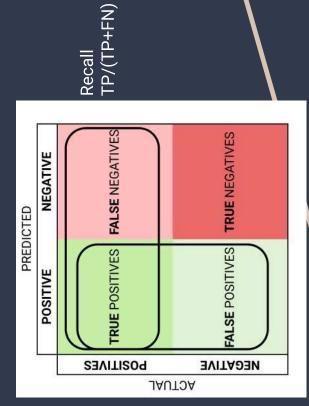
Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters Depth	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	%0.06	143.7M	19	84.8	4.4
ResNet50	86	74.9%	92.1%	25.6M	107	58.2	4.6
DAMINEDVO	90	700 JL	טט טעע	DE CNA	100	אב <i>כ</i>	4.4
							\

In order to form a baseline and choose pre-trained model, trained several different pre-trained models with only a softmax activation layer added (no custom layers)

Pre-trained models selected from https://keras.io/api/applications/with imagenet weights

EfficientNetB0 performed the best

#### Metrics



Because dataset is unbalanced, accuracy can be misleading. Favors more frequent categories.

Recall and Precision are better in this case.

In a multi-class scenario, generate Confusion Matrix for each class. Binary "Class X / Not Class X"

Confusion Matrix for "Stop sign" class	Predicted: Stop sign Predict	TP	Actual: Speed limit 100 FP TN	FP
	Predicted: Speed limit 100			
	Predicted: Yield sign	FN	NL	N

Recall minimize false-negatives

Precision minimize false-positives

F1 score is the harmonic mean of both.

$$\mathsf{TP}_{A} + \mathsf{TP}_{B} + \dots \mathsf{TP}_{N}$$

$$\mathsf{TP}_A + \mathsf{FN}_A + \mathsf{TP}_B + \mathsf{FN}_B + \dots \mathsf{TP}_N + \mathsf{FN}_N$$

Micro-average

Recall =

Precision TP/(TP+FP)

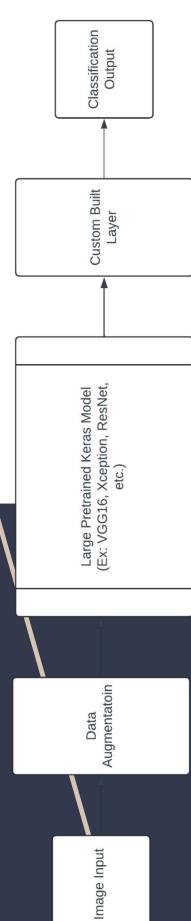
### Model tuning

Baseline validation recall: 71.70% Baseline validation accuracy: 72.04%

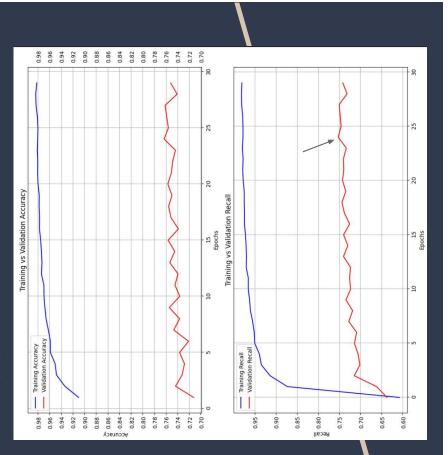
Performed a Grid Search for neurons & dropout of custom Dense layer

Overfitting was an early problem. Addressed by trying: Early Stopping, Data Augmentation, L2 Regularization, Batch Normalization Tried different hyperparameters: optimizer, number of neurons, more training time (epochs & patience)

Decaying Learning Rate



#### Results



Final model used:

Data Augmentation on 4x upscaled images
EfficientNetB0 pre-trained model as base
Batch Normalization
Custom 32-neuron Dense layer with 0.2 Dropout
Adam optimizer

97.78% training recall 98.01% training accuracy 75.24% validation recall 76.30% validation accuracy

60.11% test recall 61.05% test accuracy