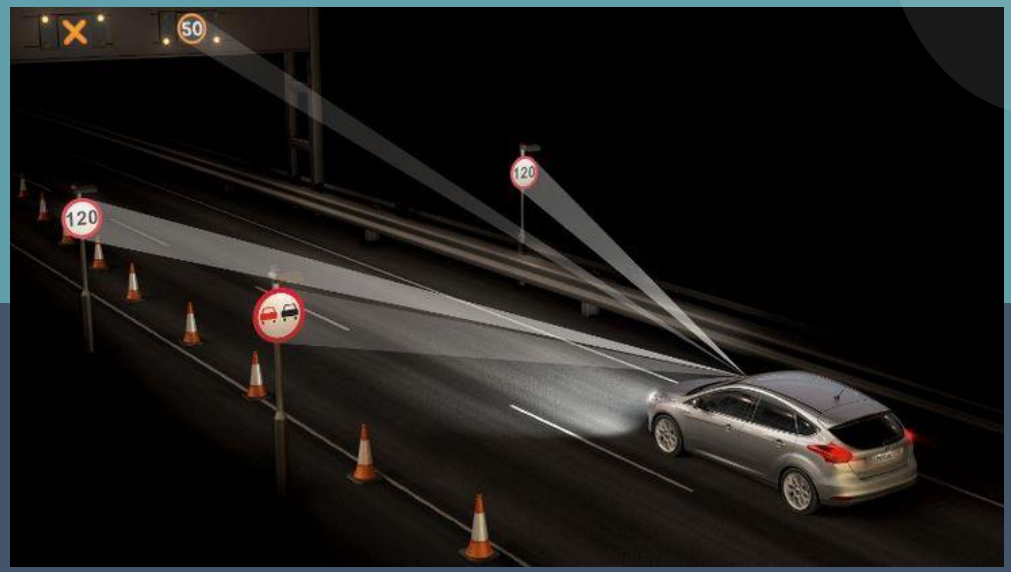
Traffic Sign Detection & Classification using Deep Learning Techniques



Team Members:

Kalyan Roy
Veerabhadra Rao M
Vishwas Bhushan Basuru
Mohammed Uvais

Agenda

- Introduction.
- Project Goals.
- Proposed Model Architecture & Technical Stacks.
- The GTSDB Dataset.
- Methodology- Data Loading and Pre-Processing.
- Methodology- Pre-Trained Object Detection Models Comparison: Faster R-CNN & RetinaNet.
- Methodology- Pre-Trained Classification Models Comparison: MobileNetV2 & ResNet50.
- Methodology- Custom Classification CNN Model-TSNet.
- Methodology- Custom Detection CNN Model- TSDetector.
- Methodology- Training, Implementation, and Evaluation.
- Results (TSDetector+ResNet50).
- Future Work & Conclusion.
- References.

Introduction

Driverless cars use machine learning, a type of artificial intelligence, to identify road signs and other things they encounter, like pedestrians and other vehicles.

- What if a sign is splattered with mud, partially covered with snow, or marked with graffiti?
- What if it is nighttime, so the lighting in the picture is different and the sign appears to be a slightly different color?
- Or what if the picture of the sign is taken at an odd or extreme angle,

All of these factors make identifying road signs and other objects a challenging task for driverless cars (also called autonomous or self-driving vehicles). To operate safely, they must be able to identify signs and objects in a wide variety of conditions and from many different angles.

In this project, you will take pictures of different types of road signs and use machine learning to identify the signs.

Project Goals

Objectives

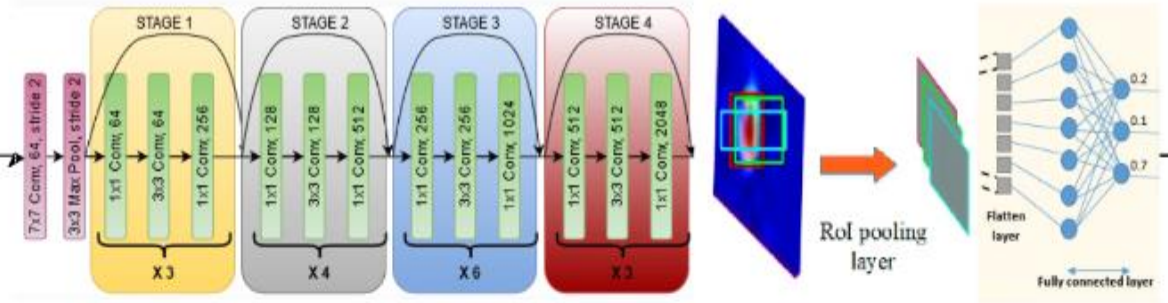
- This project focuses on showcasing the development and evaluation of two novel custom CNN architectures, TSNet and TSDetector, trained on the GTSDB dataset, intending to enhance traffic sign classification and detection accuracy.
- The presentation will highlight the comparison of our custom CNN models against established pre-trained models like MobileNetV2, ResNet50, FasterRCNN, and RetinaNet, emphasizing the potential of custom deep-learning architectures for advancing autonomous vehicle technology and improving road safety.

Constraints

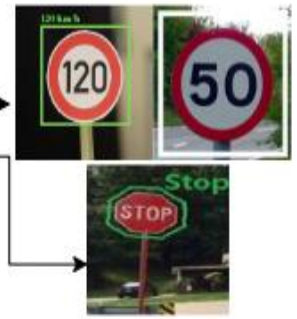
- Hardware infrastructure and processing power limitations.
- Technical limitations (lighting conditions, image quality etc).
- Transparency and user consent for data collection.
- Accessibility considerations for diverse users.

Proposed Model Architecture and Technical Stacks

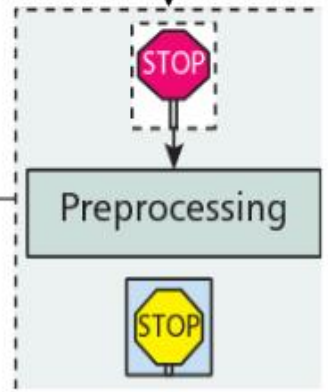
Traffic Sign Input Database



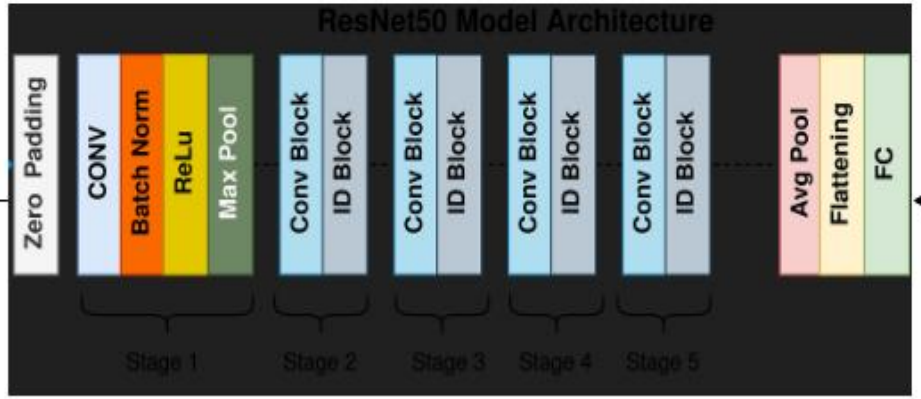
Custom Detection CNN Model TSDetector Utilizes Pre-Trained ResNet50 (Replace Final FC with ROI Pooling)



Detection of Traffic Sign



Pre-Processing the detected Symbols to input size of (32x32x3)



Pre-Trained ResNet 50 Model

Output of Classified Traffic Symbols

List out all the Software, tools, and environments used:

- **IDE**: Jupyter Notebook/Google Colab
- **Programming language**: Python-3.11
- **GPU**: T4 GPU

The GTSDB Dataset

- We employed the German Traffic Sign Detection Benchmark dataset in our exploration of traffic sign detection and classification with PyTorch. This dataset primarily caters to researchers interested in developing image-based driver assistance systems. It bears a strong resemblance to the German Traffic Sign Recognition Benchmark dataset, sharing identical classes. Additionally, the GTSDB dataset was featured in the International Joint Conference on Neural Networks (IJCNN) 2013.

Statistics of our GTSDB dataset:

- It contains a total of **900 images**. **600** - **training** images and **300** are for **evaluation**. Download the mandatory dataset files via this link like the TrainIJCNN2013.zip, TestIJCNN2013.zip, and gt.txt.
- The **TrainIJCNN2013** directory first contains the **43 class** folders which contain a few sample images belonging to the respective classes.
- **gt.txt** file which holds the ground truth annotations and classes.
- The **TestIJCNN2013** directory directly contains the 300 test images without **any ground truth information.**

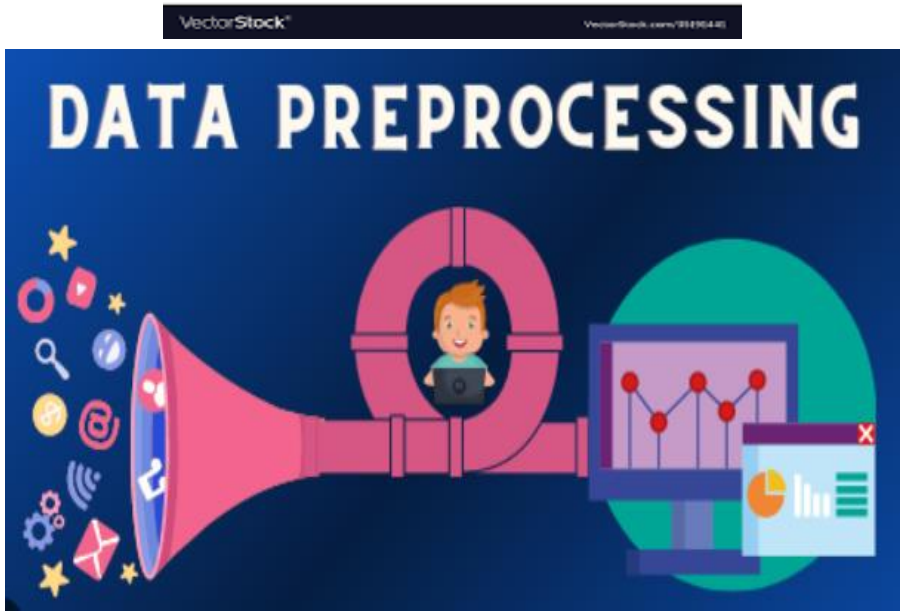

```
TrainIJCNN2013/
├── 00
├── 01
...
├── 41
├── 42
├── 00000.ppm
├── 00001.ppm
├── 00002.ppm
...
├── 00597.ppm
├── 00598.ppm
├── 00599.ppm
├── ex.txt
├── gt.txt
└── ReadMe.txt
TestIJCNN2013/
└── TestIJCNN2013Download
    ├── 00000.ppm
    ├── 00001.ppm
    ...
    ├── 00298.ppm
    ├── 00299.ppm
    └── ReadMe.txt
```

ClassID	Description	Type
0	Speed limit 20	Prohibitory
1	Speed limit 30	Prohibitory
2	Speed limit 50	Prohibitory
3	Speed limit 60	Prohibitory
4	Speed limit 70	Prohibitory
5	Speed limit 80	Prohibitory
6	Restriction ends 80	Other
7	Speed limit 100	Prohibitory
8	Speed limit 120	Prohibitory
9	No overtaking	Prohibitory
10	No overtaking (trucks)	Prohibitory
11	Priority at next intersection	Danger
12	Priority road	Other
13	Give way	Other
14	Stop	Other
15	No traffic both ways	Prohibitory
16	No trucks	Prohibitory
17	No entry	Other
18	Danger	Danger
19	Bend left	Danger
20	Bend right	Danger
21	Bend	Danger
22	Uneven road	Danger
23	Slippery road	Danger
24	Road narrows	Danger
25	Construction	Danger
26	Traffic signal	Danger
27	Pedestrian crossing	Danger
28	School crossing	Danger
29	Cycles crossing	Danger
30	Snow	Danger
31	Animals	Danger
32	Restriction ends	Other
33	Go right	Mandatory
34	Go left	Mandatory
35	Go straight	Mandatory
36	Go right or straight	Mandatory
37	Go left or straight	Mandatory
38	Keep right	Mandatory
39	Keep left	Mandatory
40	Roundabout	Mandatory
41	Restriction ends (overtaking)	Other
42	Restriction ends (overtaking (trucks))	Other

Methodology- Data Loading and Pre-Processing

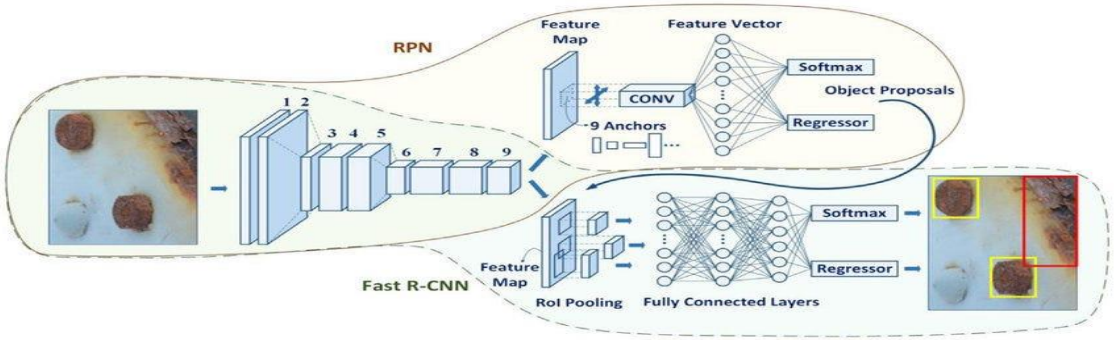
In this slide, we will discuss how we loaded and pre-processed the German Traffic Sign dataset for use in our custom CNN models. We will explore the reasons behind our choices and how they impacted the model's performance.

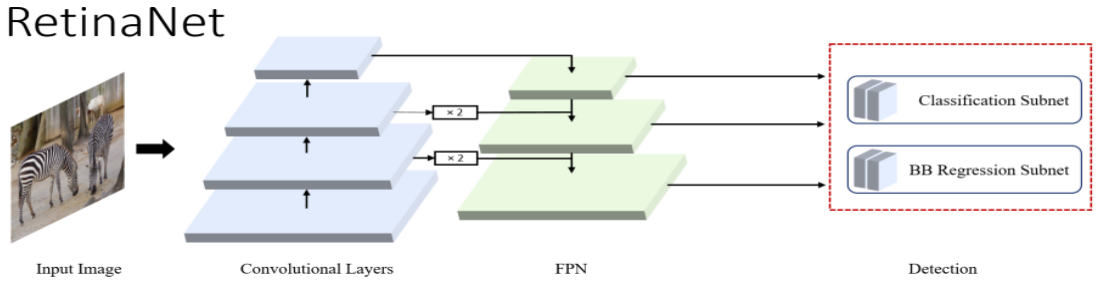
- **Custom Dataset Class**: We built a custom class (GermanTrafficSignDataset) to handle our dataset's structure (subdirectories by class labels).
- **Data Transformations**: Images are resized (32x32), converted to tensors, and normalized for consistency.
- **No Data Augmentation**: We preserved the original image orientation to aid traffic sign classification.
- **Data Splitting**: The dataset is split into training (80%) and testing (20%) sets.
- **Data Loaders**: DataLoader objects manage data with a batch size of 64 (shuffled for training).

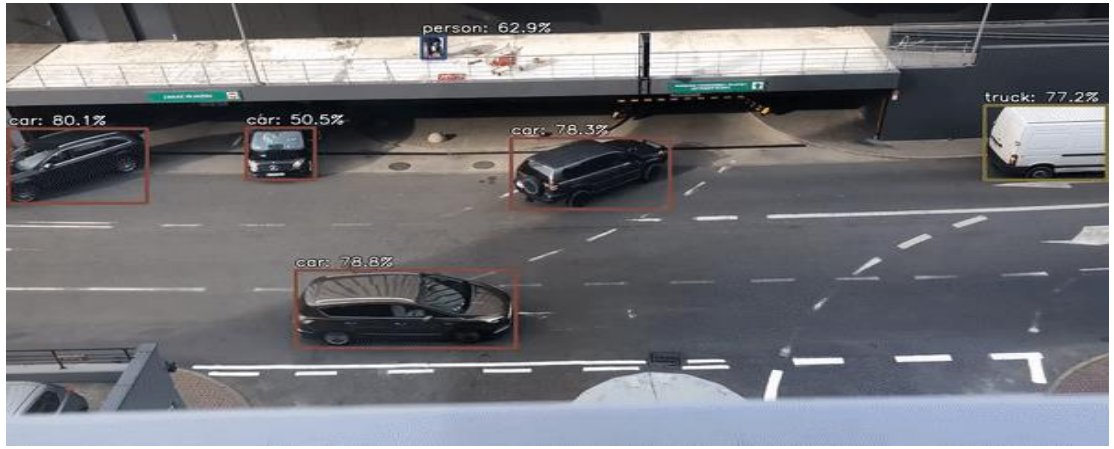


Comparison-Object Detection Models: Faster R-CNN, RetinaNet

Aspect	Faster R-CNN	RetinaNet
Model Type	Various, including ResNet50	ResNet+FPN is a common backbone
Architecture	Two-stage (region proposal network + classification/regression)	FPN-based CNN with focal loss for object detection
Speed	Slower than one-stage detectors due to two-stage process	Slower due to complex architecture
Accuracy	Very high, precise object localization	High, focal loss improves performance on hard examples
Size	Generally larger than RetinaNet	Slightly smaller than Faster R-CNN
Use Case	High accuracy object detection, commonly used when accuracy is paramount	State-of-the-art object detection, outperforms two-stage detectors
Main Application	Object detection with highest possible accuracy	Object detection with balanced speed and accuracy

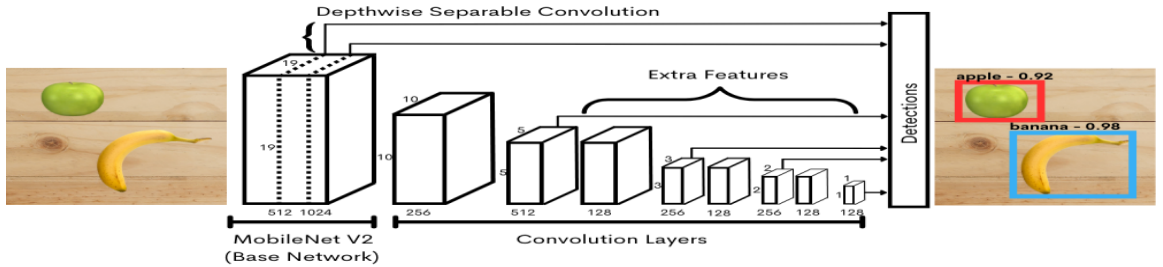


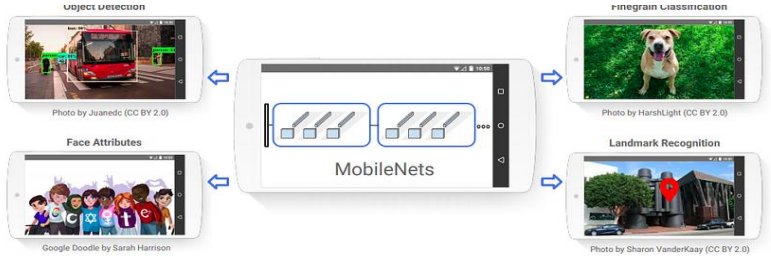


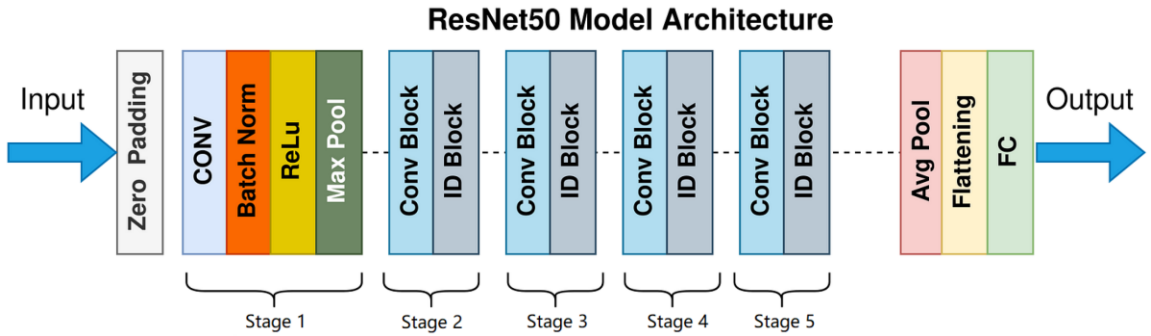


Comparison - Classification Models: MobileNetV2 & ResNet50

Aspect	MobileNetV2	ResNet50
Model Type	CNN-based Backbone Network (for feature extraction)	CNN-based Backbone Network (for feature extraction)
Architecture	Lightweight, depthwise separable convolutions	Deep CNN with 50 layers, uses skip connections
Speed	Fast, designed for efficiency	Moderate, deeper network requires more computation
Accuracy	Moderate, trades off accuracy for speed	High, benefits from deep architecture
Size	Smaller and Lighter	Larger and Deeper
Use Case	Real-time applications on resource-constrained devices	General-purpose, strong performance in image classification
Main Application	Image Classification and Feature Extraction	Image classification, high-accuracy object detection







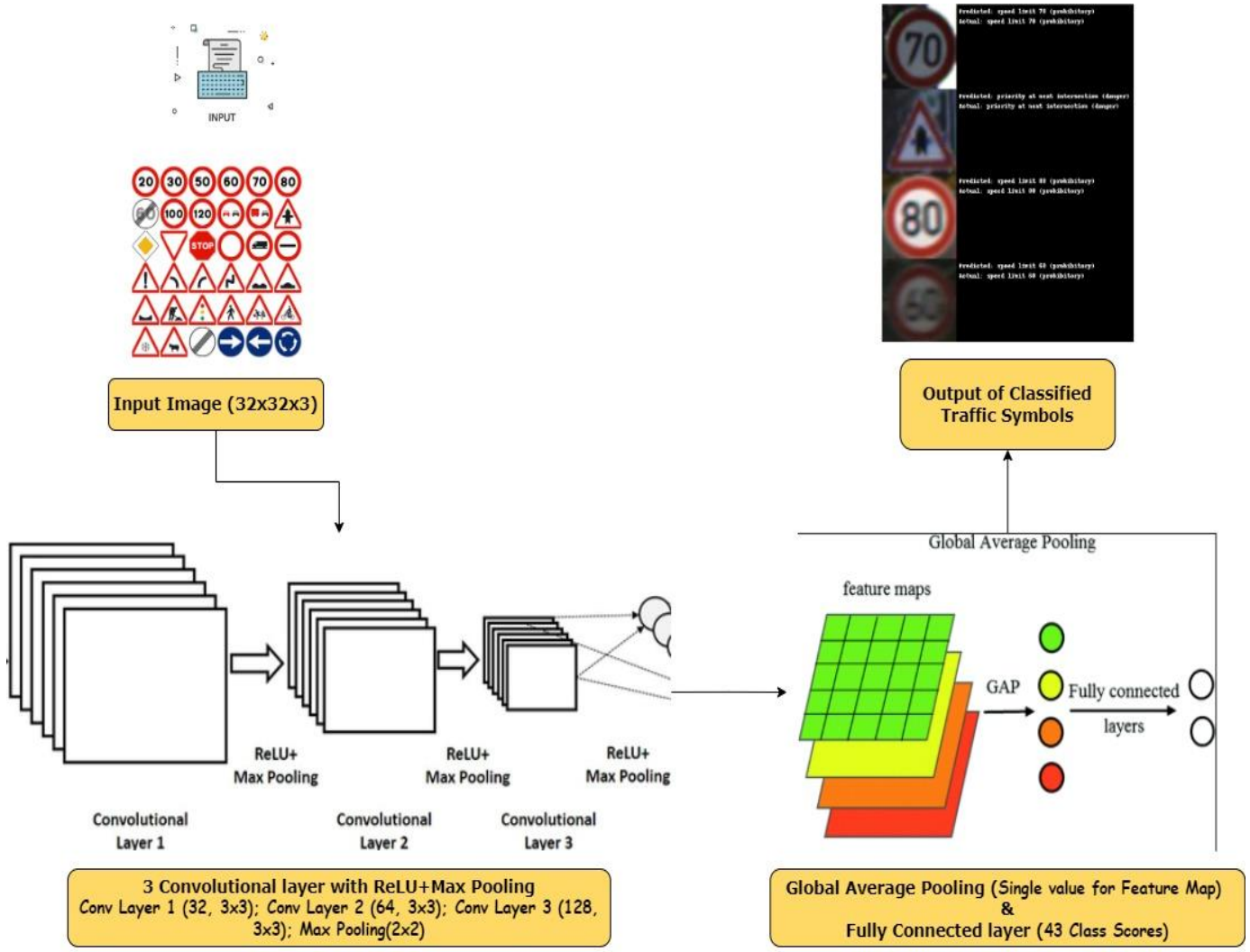
Methodology- Custom Classification CNN Model- TSNet

Focus: Efficient traffic sign classification through key feature extraction.

Architecture:

- **Convolutional Layers (3):** Extract features with increasing complexity (32, 64, 128 filters).
- **Batch Normalization & ReLU Activation**: Improve training stability and introduce non-linearity.
- **Max Pooling Layers:** Reduce feature map size and computation.
- **Global Average Pooling**: Capture key information from each feature map.
- **Fully Connected Layer:** Classify into 43 traffic sign categories.

Benefits: Optimized balance between efficiency and accuracy for traffic sign recognition.



Methodology- Custom Detection CNN Model- TSDetector

Backbone (ResNet50):

- The TSDetector utilizes a pre-trained ResNet50, discarding its final layer to leverage its feature extraction capabilities for complex patterns.

Pooling Layers:

- Leverages ROI pooling in the forward pass for targeted feature extraction from various-sized regions of interest, eliminating the need for explicit pooling layers within the model definition.

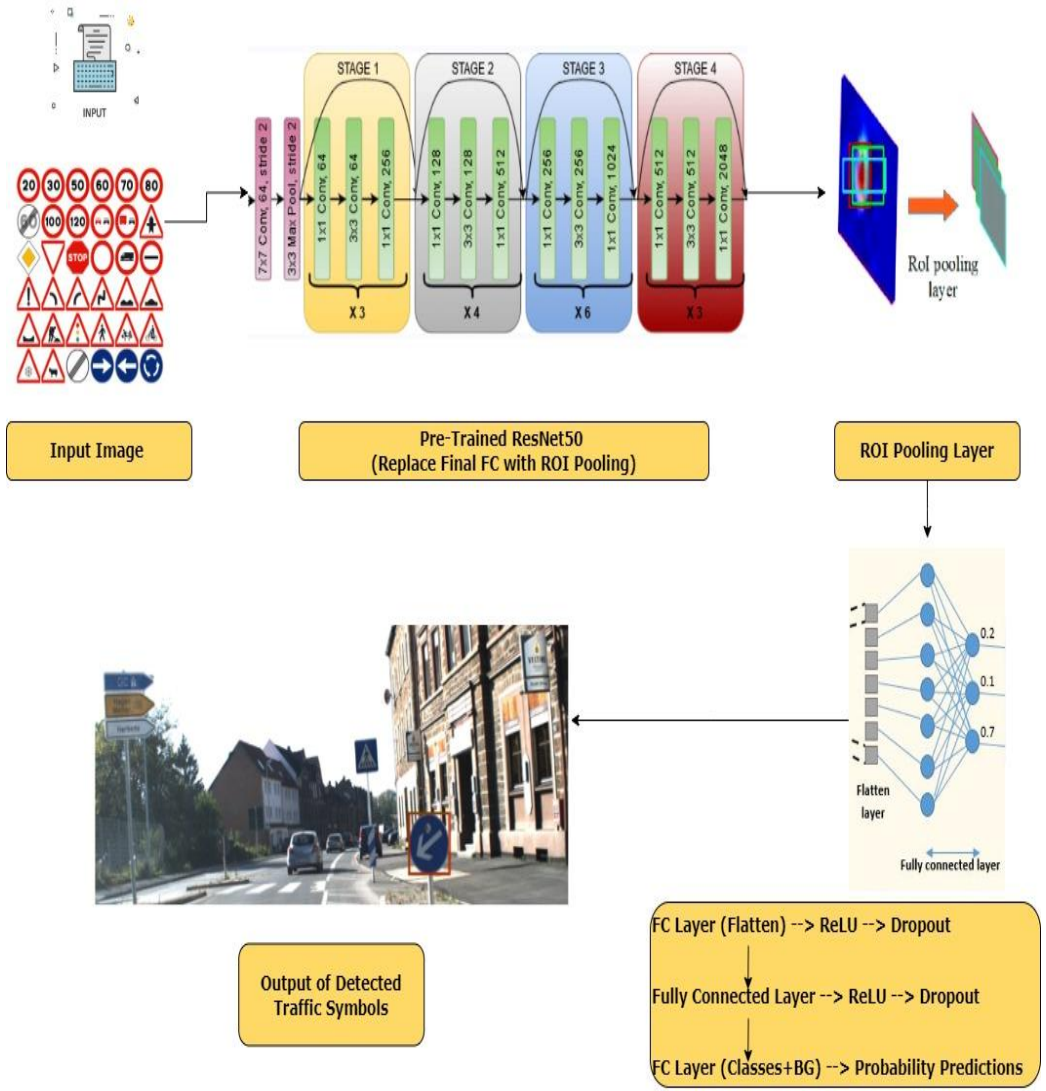
Global Average Pooling:

- Not used in this model architecture.

Fully Connected Layers:

- The flattened features are fed through two consecutive fully connected layers with ReLU activation and dropout for final class probability predictions (including background).

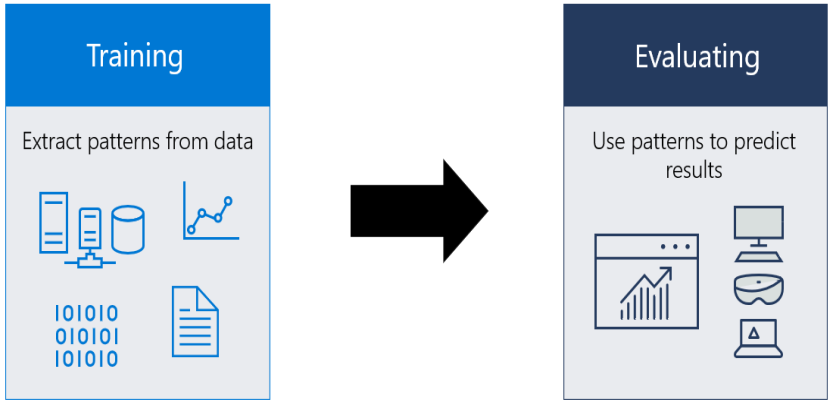
Benefits: TSDetector uses a pre-trained ResNet50 to extract features from image regions that might contain traffic signs. It skips unnecessary processing and predicts traffic sign types directly, making it efficient for object detection tasks.



Methodology- Training, Implementation and Evaluation

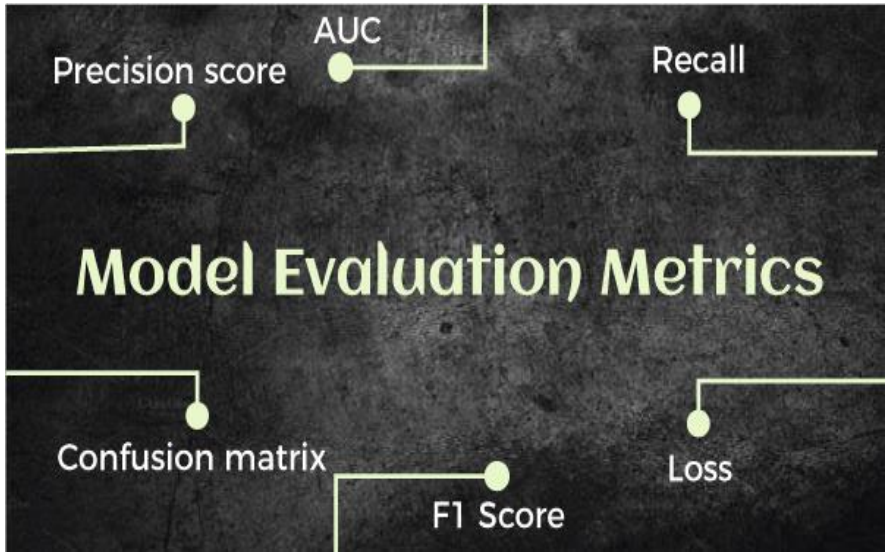
Training Our Models:

- We train with a batch size of 64 and Adam optimizer (LR: 0.001). Pre-trained models like ResNet50 and MobileNetV2 run 25 epochs, while custom models TSNet and TSDetector run 100 epochs with GPU acceleration. Training includes batch processing, cross-entropy loss computation, and test loader validation per epoch to prevent overfitting and ensure real-world applicability.



Implementation:

- PyTorch framework is used for its flexibility and efficiency in deep learning.
- Code and datasets are publicly available for reproducibility and further research.



Evaluation Metrics-Detection & Classification:

- **Accuracy:** Ratio of correct classifications to total outcomes.
- **Precision:** Proportion of true positives among predicted positives.
- **Recall:** Ratio of true positives identified by the model.
- **F1-Score:** Harmonic mean of precision and recall (balances both).
- **IoU:** Overlap between predicted and real object boxes.
- **mAP:** Average accuracy across all object classes.
- **Confusion Matrix:** Visualizes model performance for each class.

Results – Final Model (TSDetector+ResNet50)

Comparison of Classification Model Performance

Metric	ResNet50	MobileNet	TSNet
Precision	0.973	0.957	0.743
Recall	0.971	0.957	0.663
F1 Score	0.969	0.957	0.645
Accuracy	97.12%	95.88%	81.07%

Comparison of Detection Model Performance

Metric	Faster RCNN	RetinaNet	TSDetector
Precision	0.44	0.42	0.47
Recall	0.44	0.42	0.47
F1 Score	0.44	0.42	0.47
Average IoU	0.11	0.01	0.26

Integrated Output (TSDetector + ResNet50)

The integrated TSDetector + ResNet50 model demonstrates great promise for handling both traffic sign detection and classification tasks in one go. Although there's room for improving metrics, their use in these tasks provides valuable insights for future research and development in the field.

Future Work & Conclusion

Future Work:
Future work will focus on addressing their limitations, such as improving TSNet ability to recognize a wider range of signs and enhancing TSDetector's better precision and recall. Additionally, we will design future models specifically for real-world complexities, ensuring they are resilient and adaptable to diverse and challenging conditions.

Conclusion:
Our research emphasizes the importance of model selection for traffic sign recognition. We presented custom models (TSNet and TSDetector) alongside established models, demonstrating the potential of custom architectures. By building upon the insights gained from this research, we can pave the way for the development of more precise, efficient, and robust models for real-world traffic sign recognition and detection.

References

- https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.html
- https://debuggercafe.com/traffic-sign-detection-using-pytorch-and-pretrained-faster-rcnn-model/


THANK YOU
Any Questions?