Freie Universität Berlin

Bachelor Thesis at The Department of Mathematics and Computer Science

Dahlem Center for Machine Learning and Robotics

Traffic Sign Detection and Classification for Autonomous Driving

Bashar Suleiman
Student ID: 5342743
bashas99@zedat.fu-berlin.de

Advisors:	Prof. Dr. Daniel Göhring and Nicolai Steinke
Examiner:	Prof. Dr. Daniel Göhring
Second Examiner:	Prof. Dr. Tim Landgraf

Berlin, August 15, 2022

Abstract

The field of Autonomous Driving has witnessed rapid improvement and innovation in recent years. This has been driven to a large degree by the major advances in Machine Learning and Object Detection of the previous decade. Deep Neural Networks enable the vehicle to find solutions to the myriad of complex and interconnected tasks associated with Autonomous Driving. One such essential task is the reliable detection and classification of traffic signs, which is also significant for advanced driver assistance systems.

This thesis will focus on a new approach utilizing the current state-of-the-art object detection algorithms trained on public datasets. The proposed system combines YOLOv7 with a custom classifier trained on the German Traffic Signs dataset. The results are evaluated on the GTSDB dataset and fisheye footage recorded by the robotic laboratory's prototype car. The final full system achieves high results in most metrics, scoring 99.0 on Recall, Precision, and mAP on the GTSDB test dataset. But it's limited by the number of traffic sign types the classifier can recognize.

Contents

1 Introduction

The field of Autonomous Driving has been subject to a revolution in both academia and industry in recent years. We have witnessed self-driving cars migrate from laboratories to public roads, and even enter the mainstream as ever more automobile brands integrate auto-pilot systems into their products. This has been driven to a large degree by the rapid advances and innovations in Artificial Intelligence and Computer Vision during the previous decade, which were triggered by the proliferation of innovative neural networks following the outstanding results of AlexNet at the ImageNet 2012 competition. Since then, Deep Neural Networks (NN) have come to match or even surpass human performance at certain tasks. We have concurrently witnessed computational power continue to increase in accordance with Moore's Law, allowing ever larger networks to be developed. With these advancements in mind, the goal of this study is to explore the possibility of designing a Traffic Sign detection and recognition system based on state-of-the-art Deep Neural Network algorithms.

Traffic Signs are among the most ubiquitous features of any streetscape. They encode vital information that enables drivers to navigate safely and efficiently, and are crucial to maintaining orderly traffic flow. In the absence of reliable and prevalent infrastructure to digitally communicate this information to cars, Traffic Sign detection and recognition through camera footage becomes a fundamental task for both Autonomous Driving and Advanced Driver Assistance Systems. Despite it being a trivial task for humans, finding a solution tailored for Autonomous Vehicles presents complex challenges.

We begin by defining three essential criteria for a viable system: real-time speed (30 frames per second), moderate hardware cost, and high accuracy. The latter pair of conditions are challenging, since Neural Networks trend towards deeper models to achieve higher accuracy. But an autonomous vehicle is constrained by the limited on-board hardware, which is shared with several other essential software modules. The system is therefore optimized to be compatible with the prototype autonomous vehicle designed by the Free University's Autonomos Lab, which is equipped with a fisheye front-camera (1280x800 resolution).

There are multiple potential approaches to detecting Traffic Signs, such as color segmentation, shape segmentation, and deep learning. While color segmentation methods are computationally inexpensive and fast, their real-world reliability is diminished by their sensitivity to various factors such as weather conditions, time of the day, as well as reflection, age and condition of the signs [30]. Shape segmentation is more robust, but suffers similar challenges as its color-based counterpart when faced with damaged, partially obscured, blurred, or distorted traffic signs [30]. The last factor is particularly problematic given the inherit distortion in fish-eye cameras. Neural Networks are capable of learning generalized representations of objects, but can only be as good as the data they train on. Therefore the focus of this thesis lies exclusively on deep learning approaches to the traffic sign recognition problem.

To train an accurate model capable of adapting to real-world variance, a large and varied dataset is required. The available public Traffic Sign datasets fall under either the detection or classification categories. Detection datasets contain images that each include multiple traffic signs and their respective bounding box coordinates. Classification datasets meanwhile contain small images of just the traffic signs, each with a class label. The proposed system is composed of a detector and a classifier each trained on a separate dataset. The final classification model is a custom deep version of LeNet5 [19] trained on the GTSRB dataset [10], while the final detection model is YOLOv7-W6 [35] trained on GTSDB [10] and Woodscape [33] datasets. Section 4.1 presents a flowchart of the proposed system. Sections 2 and 3 provide a review of related concepts to the system. Sections 4.2 and 4.3 will explain the process behind the choice of the model architectures and datasets, and will the training steps. Finally, a thorough evaluation of the full system is performed both quantitatively and qualitatively in section 5.

2 Fundamentals

2.1 Supervised Learning

Supervised Learning is a machine learning method for statistically analyzing labeled training datasets to approximate a mapping function between the dataset's inputs and outputs. Elements of the training dataset are pairs composed of an exemplary input (commonly represented as feature vectors) and a desired output (label). The input and output should share some real-world relation. The goal of a Supervised Learning algorithm is to capture this relation in order to infer a model that fits the training dataset. The model should ideally be capable of generalizing the relation represented in the training dataset in order to correctly map unseen data points (of the same class as the training inputs) to the expected label. A Loss Function, which calculates the error margin between the model's predictions and ground-truth labels, is used to evaluate the model's performance on unseen data.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs), also known as Deep Feedforward Networks or Multilayer-Perceptrons, are an example of a supervised learning algorithm. ANNs seek to approximate the mapping function represented by the training dataset. The mapping is defined as $y=f(x; \theta)$, and the goal of the network is to learn the values of the parameters θ that result in the best function approximation. [11] ANNs are composed of multiple layers of nodes (artificial neurons). The first layer is the input layer, the final layer is the output layer, and all layers in-between are called hidden layers. A node is connected to other nodes by weighted connections. Each node passes the sum of its weighted inputs through a non-linear function, and then forwards the output to connected nodes in the next layer. [22]

The process of estimating the mapping function is called learning, and is most

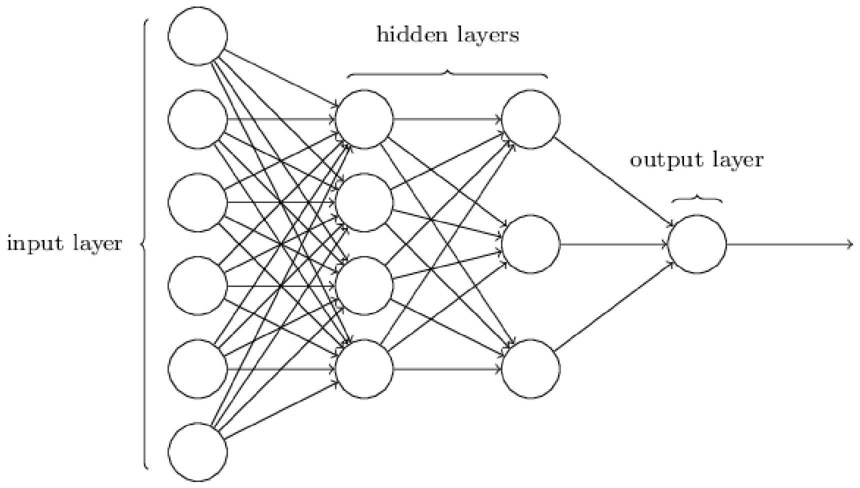


Figure 1: Example of a fully connected neural network. Image Source: [22]

commonly performed by some gradient descent algorithm. The two basic operations of the learning process are Feedforward and Backpropagation. Feedforward: When presented with some input (usually in the form of a tensor or vector), it passes all layers through their weighted connections and activation functions until it finally produces an output. Backpropagation: The output is then evaluated by some Loss Function, which computes an error value for the network's output. The Loss Function's gradient is then computed with respect to each weight by the partial derivation chain rule. The gradient is backpropagated one layer at at time from the last layer, updating the weights and biases at each layer it passes. The two operations are usually iterated many times. [22]

A Fully-Connected Network (FCNs) is an ANN in which each node is connected to every node in the next layer. The input must therefore be a vector of constant size, which makes it an inflexible model for image related tasks due to the loss of spatial information. FCNs also have many unnecessary weighted connections, which increases both the training and inference costs. To address these issues, another type of neural networks is usually used for image related tasks, the convolutional neural network.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special type of ANNs that excel at image processing and maintaining spatial information. CNNs usually consist of three types of layers, namely convolutional, pooling, and fully-connected layers. The convolutional layer aims to learn feature representations of the inputs, and is composed of several convolution kernels which are used to compute different feature maps [13]. This is conceptually done by convolving the learned kernels over a layer's input in a sliding window fashion. But in practice this process is replaced with more efficient operations such as multiplication with a Toeplitz matrix. The convolutions return high activation values when certain patterns (features) are detected, frequently with

ReLU (Rectified Linear Units) serving as the activation function. The parameters of convolutional filters are learned through backpropagation.

Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model [23]. The fully-connected layer is usually placed at the end of a CNN to perform regression. However some CNNs such as YOLOv3 [27] are fully convolutional, which allows a network to be invariant to input size.

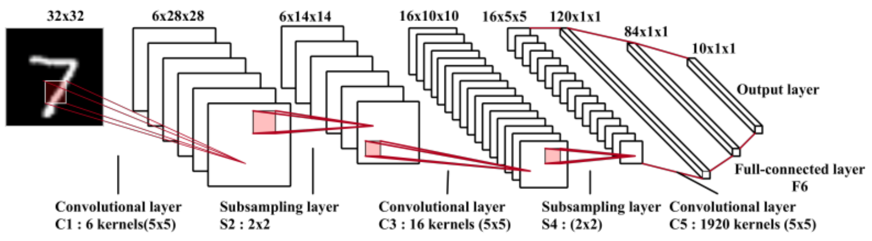


Figure 2: Convolutional Neural Network architecture example (LeNet5). Image Source: [13]

To mitigate the inefficiencies of fully-connected networks, every neuron in a convolutional layer is only connected to small region of the input. The dimensionality of this region is commonly referred to as the receptive field size of the neuron [23]. For example, the first layer of LeNet5 [19] uses a 5x5 kernel with an output depth of 6, resulting only in 180 trainable parameters.

The output of a convolutional layer will have a depth equal to the number filters, and a width and height of [(WK+2P)/S]+1 and [(HK+2P)/S]+1 where W and H are the input width and height, K is the kernel size, P is the padding, and S is the stride. Stride denotes the step width of the kernel, while padding increases the width and height dimensions of an input by wrapping it with additional cells (initialized as 0 or randomly).

2.4 Transfer Learning

Transfer learning is a supervised learning technique that reuses parts of a previously trained model on a new network tasked for a different but similar problem. For example, the pretrained weights of a model trained for detecting general objects can be finetuned for detecting traffic signs. The pretrained model can already extract the features describing a street image, so the finetuning process needs to only learn to differentiate between a traffic sign and background. This spares the new model from having to learn everything from scratch.

2.5 Object Detection

Object Detection is a subfield of Computer Vision and can be defined as the task of detecting all instances of visual objects of certain classes in an image [8]. The field began with traditional algorithms such as Scale Invariant Feature Transform (1999), Viola Jones (2001), Histogram of Oriented Gradients (2006), and Deformable Part Model (2008). But following the introduction of the Region-based Convolutional Neural Network (RCNN) in 2014 [3], Deep Neural Networks have come to dominate the field. Neural Network based object detection models are categorized either as one-stage or two-stage detectors. In two-stage object detectors (e.g. RCNN, SPPNet, Mask-RCNN), the approximate object regions are proposed using deep features before these features are used for the classification as well as bounding box regression for the object candidate. One-stage detectors (e.g. YOLO, SSD, RetinaNet) predict bounding boxes over the images without the region proposal step [8].

2.6 Object Detection Metrics

2.6.1 Confusion Matrix

To create a confusion matrix, four attributes are needed:
True Positives (TP): Predicted object matches a groundtruth sample.
False Positives (FP): Predicted object is not in groundtruth.
True Negatives (TN): No prediction is made, and there is no sample in groundtruth.
False Negatives (FN): A sample exists in the groundtruth, but was not predicted by the model.

In the context of object detection, a prediction is considered correct if its IoU and confidence are over certain thresholds. A confusion matrix helps visualize the cases in which the model fails. In the example table Fig.**??**, the x-axis represents the groundtruth and the y-axis represents the model's predictions. Ideally, the diagonal should have high numbers and everywhere else should be zeroes. In the example table the imaginary model mistakes a stop sign for a turn right sign twice, and a turn right sign with a stop sign once.

Stop Sign	1	0	32
Speed Limit 30	0	42	0
Turn Right	19	0	2
	Turn Right	Speed Limit 30	Stop Sign

Table 1: Exemplary confusion matrix.

2.6.2 Precision and Recall

Precision is the ratio between the correctly classified positive samples to the total number of samples classified as Positive. In short, Precision tells us how often the detector is correct when it makes predictions. For example, if a model detects 5 traffics sings, three of which are actually in the image, the precision is 3/5.

$$Precision = \frac{TP}{TP + FP}$$

Recall is the ratio between the correctly classified positive samples to the total number of groundtruth samples. Recall tells us whether the model predicted every time that a groundtruth sample was present. For example, if a model detects 3 out of 4 traffic signs in an image, the recall is 3/4.

$$Recall = \frac{TP}{TP + FN}$$

2.6.3 F1 Score

The F1-score combines precision and recall into a single metric by taking their harmonic mean. A higher F1-score signifies that precision and recall are high, while a lower F1-score signifies a low or imbalanced precision and recall (or lower precision and recall).

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * (TP + FP + FN)}$$

2.6.4 Intersection over Union

IoU is a metric used in object detection to compare a predicted bounding box (P) with the groundtruth bounding box (B). It is measured as area of overlap over area of union:

$$IoU = \frac{P \cap B}{P \cup B}$$

IoU represents how well the predicted bounding box fits the object. An IoU threshold is usually chosen, under which a bounding box is no longer considered a correct prediction of an object. The higher the IoU, the better the bounding box represents the detected object.

2.6.5 Average Precision

AP is a metric represents the area under the Recall-Precision curve at different confidence thresholds. An object detector is considered good if the precision and recall are not impacted by different confidence thresholds, and if the precision does not decrease as recall increases. While an integral can be used to measure the area under the curve, in practice the VOC interpolation method is used to measure an approximation. This is done by calculating the interpolated precisions at 11 recall levels: [0, 0.1, 0.2, . . . , 1], and then taking the average of their sum:

$$AP = \sum_{r=[0,0.1,...,1]} P_{interpolated}(r)$$

Mean Average Precision (mAP) is the average of the APs of each class. Since we will be using a single class for our detector, AP and mAP can be used interchangeably. It should be noted that the CoCo metric differs slightly from VOC's. First, the AP metric uses a uses 101-point interpolated AP instead of VOC's 11. It also introduces the mAP@[.5:.95], which averages the mAPs at different IoU threshold levels from 0.5 to 0.95 with a step size of 0.05.

2.7 Non-Maximum Suppression

Most object detection algorithms generate multiple bounding boxes for a single object. It's therefore necessary to be able to select a single bounding box per object. To do this, Non-Maximum Suppression (NMS) is used to select the best bounding box for each object and to discard the rest.

The algorithm is simple, its inputs are the set of proposed bounding boxes, their corresponding confidence scores, and a threshold. The proposal with the highest confidence is selected and inserted into the output set, and is then compared with all the other proposals. If their IoU is higher than the specified threshold, the proposal is removed from the set. This process is repeated until no boxes remain in the input set. [18]

3 Evolution of YOLO

You Only Look Once (YOLO) is a one-stage object detection algorithm introduced by Joseph Redmon in 2015, and further developed by Alexey Bochkovskiy after YOLOv3. It was the first object detection model to achieve real-time speed while maintaining an average precision comparable to that of Fast-RCNN. YOLOv7 is the center-piece of the proposed Traffic Sign Detection system.

Since each version builds on its predecessor, it would be difficult to understand YOLOv7's architecture without historical context. Therefore, a comprehensive review of YOLO's evolution is presented while trying to remain as concise as possible. It should be noted that YOLOv5/v6 were released by different authors, and aren't connected to YOLOv7's branch of development.

3.1 YOLOv1 [28]

YOLO reframes object detection as a single regression problem. A single convolutional network inputted with an image simultaneously predicts multiple bounding boxes and class probabilities for those boxes. The detection network has 24 convolutional layers followed by 2 fully connected layers. (Fig.3)

The input image is divided into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes, and each bounding box consists of 5 predictions: x, y, width, height, and confidence (defined as Pr(obj) * IOU). The (x,y) coordinates define the object's center coordinates relative to their grid cell, so their values always fall between 0 and 1. The width and height values are normalized by the image's width and height to also keep their value between 0 and 1. Along with bounding boxes, a grid cell contains a single set of C conditional class probabilities Pr(class_i | obj), with C being the number of classes. Predictions with confidence lower than a certain threshold are ignored, and then Non-Maximum Supression is performed to eliminate duplicate bounding boxes.

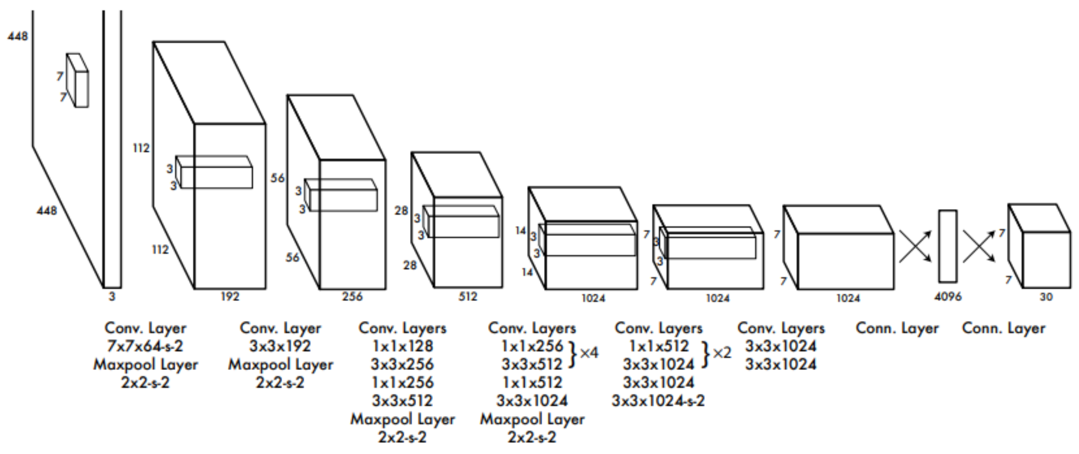


Figure 3: YOLOv1 network architecture [28]. The final layer in the network produces a tensor of dimensions SxSx((B*5)+C) with S=7, B=2, and C=10.

YOLO achieved 63.4 mAP and 45 FPS on the Pascal VOC dataset, making it the most accurate real-time detector at the time, but it was less accurate than Fast-RCNN at 70.0 mAP.

3.2 YOLOv2 [26]

YOLOv2 builds on YOLO to increase accuracy while maintaining speed. The classification network Darknet-19 (Appendix Fig. 14) is used as a backbone for feature extraction, and is pretrained on the ImageNet-1000 dataset at 224x224 resolution. The classifier is then finetuned at resolution 416x416 to adjust to detection input resolution. To transform the classifier into a detector, the last convolutional layer is removed and instead three 1024x3×3 convolutional layers are added followed by a final 1×1 convolutional layer with the number of outputs needed for detection. Each bounding box now has its own set of class probabilities, so the output will be a tensor of shape SxSx(B*(5+C)). Note that the detection head is now fully convolutional with no fully-connected layers.

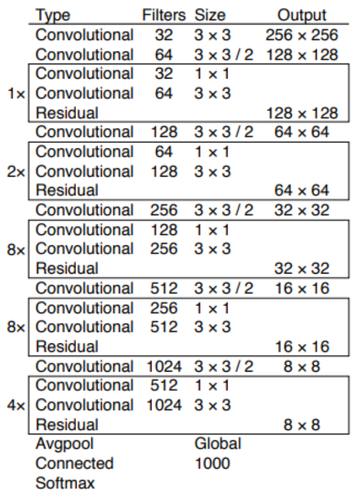
To further improve accuracy, batch normalization is added to every convolutional layers. Anchor Boxes are also introduced, but are only used in calculating the bounding box' width and height. The object center coordinates are still directly predicted in relation to their grid cell, with no influence from the anchor boxes. To find good Anchor Box priors, the authors run k-means clustering on the training set bounding boxes with k=5, but replace euclidean distance with d(box, centroid) = 1 −IOU(box, centroid).

YOLOv2 also uses fine-grained features from earlier layers with higher resolution features to help detect small objects. Instead of making predictions on the final 13x13 feature maps like YOLO, a passthrough layer is added to a previous layer to bring features at 26x26 resolution. Each 26x26 feature map is transformed to a stack of four 13x13 resolution features, and are then concatenated with the low resolution features from the final layer.

The fully convolutional architecture also makes the network invariant towards input dimensions. The network downsamples by a factor of 32, therefore every 10 batches a new multiple of 32 from the set: {320, 352, ..., 608} is chosen as input size. This forces the network to learn to predict well across a variety of input dimensions.

3.3 YOLOv3 [27]

YOLOv3 adds a few changes to YOLOv2 to improve performance. Most importantly, they replace Darknet-19 with the much deeper Darknet-53 as a backbone.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4: Darknet-53 network architecture [27]

Additionally, they now perform predictions on 3 scales each with 3 anchors. The features of the final layer with a resolution of 13x13 are upsampled by a factor of 2 and are concatenated with features from a layer with resolution 26x26. These features are in turn upsampled by a factor of 2 and concatenated with feature from a layer of resolution 52x52. This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature maps. Detections are performed on the feature maps of all 3 scales, and the results are filtered through non-maximum suppression.

3.4 YOLOv4 [7]

YOLOv4 reframes the network as 3 modules: backbone, neck, and head. The backbone is a classification network that is repurposed to extract features, the neck collects feature maps from different layers and aggregates them to enhance the receptive field, and the head is the part of the network actually responsible for detection.

CSPDarknet-53 is chosen as the new backbone, which integrates Cross Stage Partial (CSP) connections [2] to the Darknet-53 network used in YOLOv3. Spatial Pyramid Pooling, a Path Aggregate Network, and a Spatial Attention Module are used for the neck. The YOLOv3 head is still used for detection, but "Mish" is now used as the main activation function and DIoU-Non-Maximum-Suppression is used to filter bounding boxes.

YOLOv4 also introduces many new techniques to improve training, called "Bag of Freebies" because they don't affect inference time. For training the classifier backbone, Mosaic and CutMix are used for data augmentation, as well as Label Smoothing and DropBlock for regularization. For training the detection network, many techniques were added to the Bag of Freebies: CIoU-loss, Cross-Iteration mini Batch Normalization (CmBN), DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, eliminating grid sensitivity, using multiple anchors for a single ground truth, cosine annealing scheduler, optimal hyperparameters, and random training shapes.

3.5 Scaled YOLOv4 [34]

Cross Stage Partial connections [2] are used to enhance the architecture of YOLOv4 creating YOLOv4-CSP. This model is then scaled up and down to create different versions of different sizes, from smallest to largest: YOLOv4 P4, P5, and P6. Scaled up models are more accurate but have less inference speed, while scaled down models are faster but less accurate.

3.6 YOLOR [36]

YOLOR (You Only Learn One Representation) combines Explicit and Implicit knowledge into a unified model. The authors define Explicit Knowledge as knowledge that directly corresponds to an observation (e.g. input). Implicit Knowledge has nothing to do with observation or input, and can be viewed as accumulated general knowledge about a task. Explicit Knowledge can be modelled by a normal neural network, the authors use YOLOv4-CSP as the explicit model. Implicit Knowledge meanwhile is a constant set of latent code that is applied directly on the next layer, but isn't connected to any previous layers. To integrate Implicit Knowledge, the Unified Model is proposed: y = $f_\Theta(x) * g_\phi(z)$ where x is the observation, Θ is the parameters of the neural network f_Θ, z is the implicit knowledge latent code, g_ϕ is a task specific operation that selects relevant implicit knowledge, and $\star$ is some operator (addition or multiplication). Implicit Knowledge can be modelled as a vector, a neural network, or a matrix. The values of the implicit model are learned through backpropagation, and their initial values are sampled from a Gaussian Distribution if no prior knowledge exists.

The authors propose and test three usecases for Implicit Knowledge, all of which improve YOLOv4-CSP's mAP. The first is feature alignment in Feature Pyramid Networks (FPN). Implicit Knowledge latent code is added to the feature map of each FPN layer, which helps align the positions of large and small objects at the different scales of the pyramid. The second usecase is the refinement of the network's predictions. Multiplying a separate set of latent code with the network's predictions improves their accuracy. This is because the latent code can learn the patterns of each anchor box during training. The third usecase improves the network's performance on multiple tasks. Since we use YOLOR for a single task only, this usecase won't be reviewed.

3.7 YOLOv7 [35]

YOLOv7 builds on YOLOR by introducing two "Trainable Bag of Freebies (BoF)". The first BoF method is Module-Level Re-parameterization. It involves replacing some convolutional layers with computational blocks that aid learning during training, that are then collapsed back into a single convolutional layer during inference. For example, the 3x3 convolutional layers in CSPDarknet-53 blocks can be replaced with a computational block. The output of this block is the sum of the outputs of the original 3x3 convolution, a 1x1 convolution, and an identity connection each followed by batch normalization. Adding this block was found to improve gradient backpropagation during training, but increases the inference cost. Therefore, the three operations are then merged into a single 3x3 convolutional layer for inference. This is done by summing the weights and biases of the three layers and integrating the mean and variance from their batch normalization layers. It should be noted that the identity connection is only used if no other skip connection already points to the block's output. Module-level re-parameterization is also used to merge Implicit knowledge operations from YOLOR with convolutional layers. Implicit Knowledge can be simplified to a vector and then combined with the bias and weight of the previous or subsequent convolutional layer.

The second BoF method is adding a second detection head to the middle of the network. The detection head at the end of the network is still responsible for all predictions, and is called the "Lead Head". The new detection head at the middle is called the "Auxiliary Head", and is meant to help backpropagate detection-loss gradients to the shallow layers. To achieve this, a "Label Assigner" compares the Lead Head's prediction with the ground truth to generate soft labels. The Label Assigner then passes soft Fine-Labels (all classes) to the Lead Head, and soft Coarse-Labels (only superclasses) to the Auxiliary Head. The Auxiliary Head is stripped from the network after training.

A couple of architectural changes are also introduced, such as integrating "Efficient Layer Aggregation" (ELAN), which replaces Cross Stage Partial connections at several positions of the system. The ELAN blocks (Appendix Fig. 15) increase efficiency by shortening the longest path gradients need to travel. Another architectural change is the "Compound Model Scaling" approach, in which model width and depth are scaled in coherence. The new scaling approach delivers better performance than the methods used for Scaled-YOLOv4.

4 Creating the System

4.1 Proposed System Architecture

First, a ROS node subscribes to the frontcamera topic and recieves 1280x800 images. The images are passed on to the detection module, which detects bounding box parameters for all Traffic Signs in the image. There will be several bounding boxes per traffic sign, so non-maximum suppression is used to filter the overlapping boxes. The traffic sign inside the bounding box will be cropped, resized to 32x32, and passed on to the classifier. The classifier will assign the each traffic sign one of 43 labels, and pass it back to the detector. The detector can then forward the label and bounding box coordinates back to the ROS node for publishing.

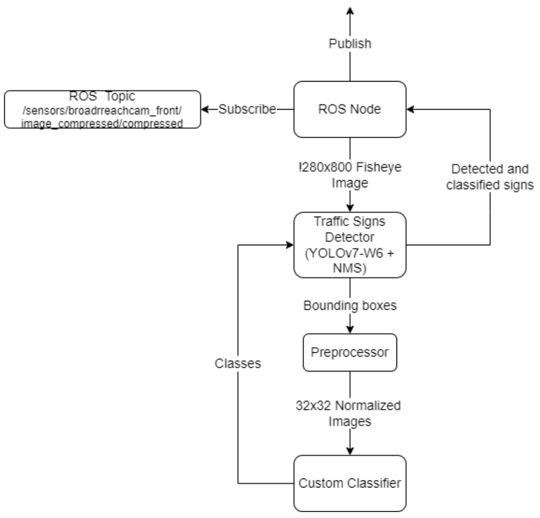

Publish
ROS Topic /sensors/broadrreachcam_front/ image_compressed/compressed
Subscribe
ROS Node
!280x800 Fisheye Image
Detected and classified signs
Traffic Signs Detector (YOLOv7-W6 + NMS)
Bounding boxes
Preprocessor
Classes
32x32 Normalized Images
Custom Classifier

Figure 5: Flowchart of the proposed system architecture

4.2 Classification Module

4.2.1 Classification Dataset [10]

There are several options for the classification dataset, the table below (Table **??**) lists some popular European traffic sign datasets. GTSRB stands out as the best choice for having the most samples and no foreign traffic signs. Additionally, GTSRB has a complementary detection dataset (GTSDB) that is annotated with the same class labels, making it very practical for the development of a combined detection and classification system.

The German Traffic Sign Recognition Benchmark (GTSRB) dataset contains 43 classes of traffic signs and over 50,000 images with one traffic sign each. A list of all the classes and their numerical labels can be found in the Appendix. The dataset is divided between 39,209 training images and 12,630 test images. Image sizes vary between 15x15 to 250x250 pixels and contain a border of 10 % around the actual traffic sign (at least 5 pixels).

Dataset	Country	Images	Classes	Source
GTSRB	Germany	50000+	43	[10]
BTSC	Belgium	4591	62	[6]
DITS	Italy	8000+	58	[16]

Table 2: Three potential classification training datasets.

GTSRB images come in various and diverse lighting conditions and forms. The 43 classes cover the most important and frequent traffic signs. A notable exception is the "No Stopping" traffic sign, which isn't present in the dataset. The images also aren't distributed equally across the classes, the following image illustrates the dataset distribution by classes:

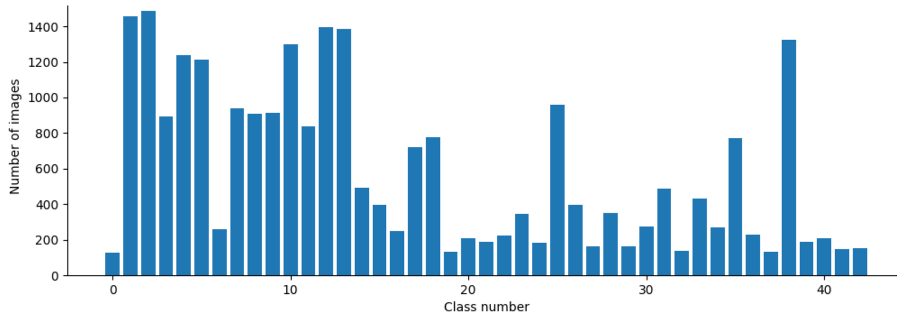


Figure 6: GTSRB training dataset distribution by class.

All images were transformed from the original PPM format to PNG, then resized to 32x32 to give us a constant training size. A dataloader python program extracts labels from an associated CSV file then transforms all images to tensors and matches them with the appropriate label. The dataset is saved as two PTH files used for training and testing.

4.2.2 Classification Model Design

To establish a baseline, custom PyTorch implementations of LeNet5 [19] and ResNet18 [15] were trained after processing the dataset. The goal was to compare the performance of a small network against a relatively large network. The training was done with 1e-3 learning rate, 50 epochs, and 256 batch-size. Adam and StepLR were used as optimizer and scheduler, and Cross Entropy Loss was used as the criterion. All tests were performed on an Nvidia RTX-3080 Laptop GPU.

The results make it immediately apparent that complex models are not the way forward for this task. LeNet5 is one of the smallest and oldest convolutional neural networks ever designed, but is nevertheless able to hold its ground against ResNet18 despite being smaller by a factor of 172 times (parameter-wise). LeNet5 was originally designed for 32x32 images [19], which is the reason it was chosen as a baseline and

Model	Training Acc	Test Acc	Parameters	Speed (Batch 1)	Size
LeNet5	100.000	93.74505 (11840/12630)	64,811	0.447ms	0.37MB
ResNet18	99.997	96.650 (12207/12630)	11,198,763	3.911ms	44.44MB

Table 3: LeNet5 and ResNet18 results.

helps explain its high performance. In fact, several successful traffic sign classification models were trained on LeNet5 variants [12]. The results present an opportunity to design a fast and near-perfect classifier.

To achieve that, a custom network heavily inspired by LeNet5 called CustomTS was designed for this task. The design strategy was to increase the network depth (both in filters and layers) and to integrate modern best-practices (LeNet5 is decades old). To increase depth, a third convolutional block was added. Since the input dimensions are so small, using maxpool with stride=2 quickly reduces the feature map dimensions to 1x1, thus all strides are removed. The optimal kernel sizes and output filters were determined by trial and error. The best results were obtained from the configuration of kernel sizes 5x5->3x3->3x3 and output filters (96, 96*2, 96*3).

Additionally, BatchNorm2d was added to every convolutional block and Dropout was added to linear layers. The activation function used by LeNet5 is Tanh(), which is somewhat outdated and prone to the vanishing gradients problem. It was therefore replaced with the more robust Mish() [21], which avoids the problem due to being unbounded from above but bounded from below. Mish() also has the favorable features of being self-regularized and continuously differentiable, and has been reported to increase accuracy compared to ReLU [21].

$$Mish(x) = x * tanh(ln(1 + e^x))$$

The following table illustrates the changes in architecture:

CustomTS	LeNet5
Conv2d(in=3, out=96, kernel=5)	Conv2d(in=3, out=6, kernel= 5)
Mish() -> MaxPool2d(k=2) -> BatchNorm2d	Tanh() -> MaxPool2d(k=2, stride=2)
Conv2d(in=96, out=192, kernel=3)	Conv2d(in=6, out=16, kernel=5)
Mish() -> MaxPool2d(k=2) -> BatchNorm2d	Tanh() -> MaxPool2d(k=2, stride=2)
Conv2d(in=192, out=288, kernel=3)	-
Mish() -> MaxPool2d(k=2) -> BatchNorm2d	-
Dropout(0.6)	-
Linear(1152, 400)	-
Mish()	-
Dropout(0.5)	-
Linear(400,120)	Linear(400, 120)
Mish()	Tanh()
Linear(120,43)	Linear(120, 84) -> Tanh() -> Linear(120, 43)
Mish()	Tanh()

CustomTS is still a relatively lightweight network, but is nevertheless able to beat ResNet18 by a decent margin at test accuracy:

Model	Training Acc	Test Acc	Parameters	Speed (Batch 1)	Size
CustomTS	**99.354**	**97.846 (12358/12630)**	**1,187,003**	**0.8ms**	**6.61MB**
LeNet5	100.000	93.74505 (11840/12630)	64,811	0.441ms	0.37MB
ResNet18	99.997	96.650 (12207/12630)	11,198,763	3.911ms	44.44MB

To improve accuracy even further, a spatial transformer is integrated and the model is retrained on normalized and heavily augmented data.

1) Spatial Transformers

The network that won the original GTSRB competetion was "CNN with 3 Spatial Transformers" by the team DeepKnowledge Seville [9][10]. They were able to achieve 99.71 test accuracy on GTSRB through the use of Spatial Transformers. We will therefore explore the concept of STNs and integrate it to CustomTS.

Spatial Transformers (STN) [17] are a popular way to increase spatial invariance of a model against transformations such as translation, scaling, rotation, cropping. This is achieved by adaptively transforming their input into the expected pose, leading to better classification performance. The appropriate transformation is applied depending on the input's deviation from the expected pose. The prediction of the appropriate transformation is produced by a neural network called the localisation network that learns its parameters through backpropagation [31].

STN is composed of three modules: the localisation network, the grid generator, and the sampler. Of these three, only the localisation network has trainable parameters. The generator and sampler are normal functions that aren't affected by backpropagation. [17]

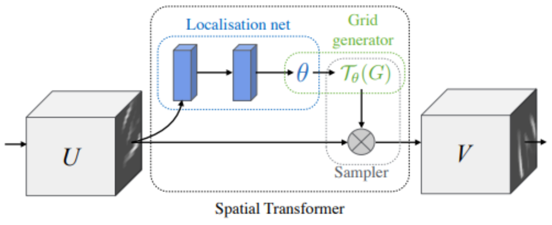


Figure 7: Architecture of a spatial transformer. Source: [17]

The PyTorch implementation of the localisation network is composed of two convolutional layers and two fully-connected layers [1]. It performs a regression on the input image to produce parameters θ that encode the type of transformation required to transform the input into the expected pose.

The grid generator applies an inverse transformation Tθ(G) on the input using the parameters obtained from the localisation network to calculate the corrected images. The outputs of the grid generator will mostly be undefined on the output image plane, so the sampler applies bilinear interpolation to find defined points for each output of the grid generator. [17]

Adding a spatial transformer to the start of CustomTS immediately improves accuracy. The results are summarized in the table in the next section.

2) Data Augmentation

For the final training run, intensive data augmentation was applied. First, the mean and standard deviation of the training dataset are calculated to normalize all training and test images. Next, the training dataset size is increased 10-fold by applying zoom, random noise, hue jitter, contrast jitter, saturation jitter, brightness jitter, random rotation, random shear, random perspective, and random translation.

Model	Test Acc	Parameters	Speed (Batch 1)	Size
CustomTS (CTS)	97.846 (12358/12630)	1,187,003	0.8ms	6.61MB
TSN + CTS	98.693 (12465/12630)	1,195,547	1.3ms	6.71MB
Aug + TSN + CTS	**99.588 (12578/12630)**	1,195,547	1.3ms	6.71MB

4.2.3 Classification Model Evaluation

The final model of CustomTS achieves near-perfect scores in all metrics: 99.58828% test accuracy, 99.58844% weighted F1 score, 99.59786% weighted Precision, and 99.58828% weighted Recall. A normalized confusion matrix for all 43 classes can be found in the Appendix: Fig.16. To visualize the rare cases in which misclassification occurs without the oversized 43-class matrix, all labels were clustered into 4 superclasses: mandatory (red rim), prohibitory (blue background), danger, and other. The GTSRB class-list in the Appendix lists the superclass of each class. The following normalized confusion matrix is for these 4 superclasses.

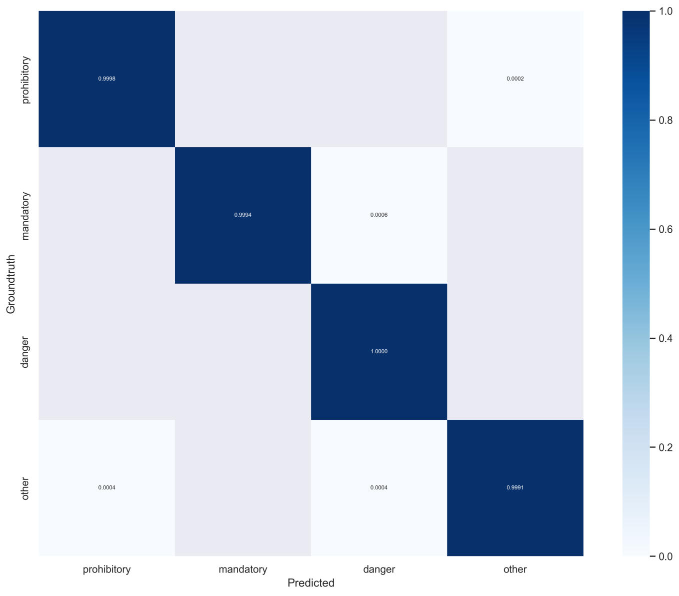


Figure 8: Normalized confusion matrix of the CustomTS with 4 superclasses.

If CustomTS was in the original GTSRB competition, it would have won second place, displacing the current 2nd place holder (Committee of CNNs by team IDSIA: 99.46% test acc) [10]. The current state-of-the-art model on GTSRB is a Deep Inception Based CNN [32] and it achieves 99.81% in test accuracy, but at the cost of having over 10 million parameters, 10 times more than CustomTS. With just 1.3ms processing speed, CustomTS is appropriate for use as the system's classifier.

4.3 Detection Module

To choose a suitable detector for the detection module, a review of available models is first necessary. Most existing Traffic Sign Detection (TSD) models use variations of popular and mainstream object detection algorithms such as RCNNs [4], SSD [4], or YOLOv3 [29]. But in the last few years, many new state-of-the-art models have pushed the boundary in both accuracy and speed, which presents new opportunities for TSD. Detection models are evaluated by their mean Average Precision (mAP). PapersWithCode maintains a leaderboard ranking of popular Object Detection models based on their mAP score on the CoCo dataset [24].

The most notable recent advancement is the SWIN Transformer [20], specifically the large SWIN-L variant. The current 17 highest ranked algorithms in the Paperwithcode leaderboard all integrate SWIN-L into their backbone for their version with highest mAP. While SWIN-L's outstanding performance is tempting, its enormous size and number of operations (197 million parameters and 1470 GFLOPS) [20] makes systems using it as a backbone impractical for autonomous driving applications. We therefore exclude all models that integrate SWIN-L into their architecture. After limiting the search to models capable of real-time performance, YOLOR [36] and YOLOv7 [35] emerge to be the two models with the highest Average Precision while still being capable of real-time inference speed on 1280x1280 images.

YOLOR and YOLOv7 each have multiple versions based on the scale they use. These scales range progressively from smallest to largest: P6 - W6 - E6 - D6. The following list illustrates the wide gap in FPS between YOLOR/v7 and other models of comparable AP. All following test results are extracted from [35], and were performed on Nvidia V100 and batchsize 1.

Model	Image Size	FPS	Parameters	AP val
YOLOR-CSP	640	106	52.9M	50.8
YOLOR P6	1280	76	37.2M	53.5
YOLOR W6	1280	66	79.8G	54.8
YOLOR E6	1280	45	115.8M	55.7
YOLOv7	640	161	36.9M	51.2
YOLOv7 W6	1280	84	70.4M	54.6
YOLOv7 E6	1280	56	97.2M	55.9
YOLOv7 D6	1280	44	154.7M	56.3
SWIN-B (C-M-RCNN)	1333	11.6	145.0M	51.9
Dual SWIN-L	1600	1.5	453M	59.1
EfficientDet-D7x	1536	6.5	77.0M	54.4
ViT-Adapter-B	-	4.4	122.0M	50.8
YOLOv5-X6	1280	38	140.7M	55

While YOLOR E6 and YOLOv7 D6 appear to be appropriate, it should be highlighted that the Nvidia V100 is an AI specialized GPU that is vastly superior to a general purpose GPU one would expect in an autonomous vehicle. Therefore it's unlikely that models with less than 60 FPS on V100 would be able to achieve adequate FPS on a normal GPU. I also exclude the smaller models YOLOR-CSP and YOLOv7, which were optimized for detection at input size 640x640. Therefore all tests were focused on YOLOR-P6, YOLOR-W6, and YOLOv7-W6 which offer a good compromise between cost and performance.

4.3.1 Detection Datasets

1) German Traffic Signs Detection Benchmark Dataset [10]

The German Traffic Sign Detection Benchmark (GTSDB) dataset is a very popular benchmarks for training and evaluating traffic sign detection models. It contains 600 training images and 300 test images of size 1360x800. All images are taken from traffic, and include up to 6 traffic signs. 140 images from the dataset contain no traffic signs. The images include a variety of weather conditions and times of day, but no night photos. Since GTSDB is the detection counterpart of the GTSRB classification dataset, the same 43 class labels are used.

To prepare the dataset for training, all images are first resized to 1280x800 (the input dimensions of the proposed system). Bounding box parameters are then processed to realign with the new image dimensions. Next, the bounding box parameters are transformed from the VOC format [x top-left, y top-left, x bottom-right, y bottom-right] to the yolo format [x center, y center, width, height]. Finally, all class labels are set to 0. Since the proposed system has a separate classifier, the detector should focus only on finding traffic signs.

GTSDB will be the main dataset for training and evaluating the different detection models we have. But by setting all class labels to 0 and focusing solely on finding the traffic signs, it becomes possible to mix different datasets.

2) Valeo Woodscape Dataset [33]

This dataset is only intended for augmenting GTSDB with 156 new training images. Woodscape is a fisheye-lens traffic dataset designed to promote training neural networks directly on fisheye-lens images rather than using rectification methods. It's used to familiarize the detection module with fisheye-lens images. The bounding-box version of the dataset contains 8234 images of size 1280x966 from 4 cameras and 7 classes.

The first processing step is eliminating all images that don't include a traffic sign, which leaves 1800 images. Since we're only interested in images from the front-camera, the side- and back-camera images are discarded, leaving only 650 images. These images are then resized to 1280x800, their bounding boxes parameters are re-aligned and converted to the yolo-format, and all class labels are set to 0.

There are still some issues with the remaining images. The Woodscape definition of what constitutes a "Traffic Sign" seems somewhat loose. For example, sometimes direction signs are labeled as traffic signs (see Figure: 9). Many of these signs are barely readable and don't hold any relevant information to traffic. Additionally, the back-side of traffic signs are sometimes annotated, despite holding no information. There are also several duplicate images.

Images that are inappropriate to the proposed system are therefore discarded,

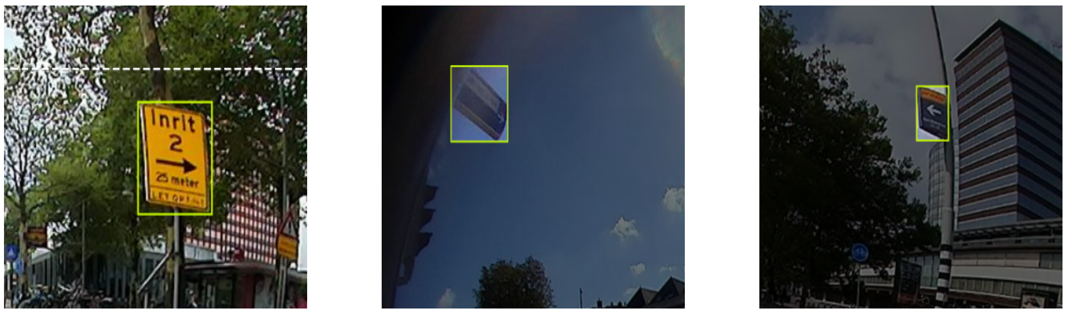


Figure 9: Some problematic samples from the Valeo Woodscape dataset.

leaving only 156 images.

3) Custom FU Berlin Evaluation Dataset

Since the prototype vehicle is equipped with a fisheye lens camera, GTSDB doesn't perfectly reflect the environment the detector is intended for. To have an objective metric of how well the detector performs on fisheye images, I extracted and labeled 100 images with traffic signs from bagfiles recorded during the test drives. The extracted images are from 17 unique locations at different timepoints. As before, the parameters are in yolo format and all traffic sign labels are set to 0.

4.3.2 Detection Models Training and Results

Training and testing was done using the official configuration files and code from the authors [25]. The configuration files of YOLOR-P6 and -W6 had to be slightly adjusted to function on single-class detection. The augmentation methods used are geometric translations, shear, flipping (left-right), saturation/hue random shifting, scaling, and mixup. Each model was trained twice for 300 epochs with batch-size 8. The following results are from the best epochs of the best runs. It should be noted that the speed column refers only to inference speed. Non-Maximum Suppression (NMS) takes between 1.2 and 1.5 ms on top of the inference time. All tests were performed on an Nvidia RTX-3080 Laptop GPU with IoU threshold 0.5 and confidence threshold 0.001.

Model	Precision	Recall	mAP@.5	Speed (batch 1)	(batch 32)	# Operations
YOLOR-P6	75.2	97.0	97.7	33.8 ms	16.3 ms	80.3 GFLOPS
YOLOR-W6	60.0	98.1	97.7	35.3 ms	19.5 ms	112.1 GFLOPS
YOLOv7-W6	97.5	96.1	98.4	19.8 ms	14.7 ms	88.7 GFLOPS

All three models achieve very high recall and inference speed. If we account for NMS by taking the observed upper-limit of 1.5 ms, the theoretical FPS is 28, 27, and 47 for YOLOR-P6, YOLOR-W6 and YOLOv7-W6 respectively. It should be noted that the FPS will decrease in the final evaluation due to the classification module, preprocessing, and miscellaneous operations. We also notice that the YOLOR models don't perform very well on precision, falling 20-35% short of YOLOv7. The significant improvement in precision and inference speed warrants the 2% drop in recall, YOLOv7-W6 is therefore picked as the main detector from this point on.

To push the performance of YOLOv7-W6 even further, three experiments are performed. The next 3 subsections will review the experiments and their results.

1) Adjusting the Loss Function

For the first experiment, some changes were made to the loss function. The final loss values in YOLO are the weighted sum of the box, objectness, and class losses. Objectness can be understood as the model's confidence that an object's center falls within a certain grid cell. Binary Cross Entropy is used to calculate both the class and objectness losses, while the mean of the Complete-IoU is used to calculate box loss. After some inspection, I noticed that the weight for the box loss (1.0) is higher than the weight for objectness loss (0.7). This is likely because the CoCo mAP metric is evaluated over several IoU thresholds, and needs to achieve 95% overlap for predictions to even be considered a True Positives at the final threshold. This could lead to accumulated small bounding box loss gradients overpowering objectness loss gradients, even when the boxes fit the objects well enough.

Since the proposed system benefits more by increasing unique object detections than by having perfect bounding boxes, the weight of the objectness loss was increased to (1.0). The weight for class loss is set to (0.3), but this will have negligible effect since only a single class is used. Training YOLOv7-W6 for 300 epochs with the adjusted loss function lead to a 0.9% increase in recall and 0.6% increase in mAP.

2) Finetuning Pretrained Weights

For the next experiment, YOLOv7-W6 was loaded with weights pretrained on the CoCo dataset (download source: [25]) and finetuned for 100 epoch with the upgraded loss function (1.0 weight for objectness loss). This approach yielded the best results so far. The resulting model will be referred to as "V7Best":

Model	Precision	Recall	mAP@.5
Baseline	97.5	96.1	98.4
(1.0) ObjLoss	98.3 (+0.8)	97.0 (+0.9)	99.0 (+0.6)
V7Best	**98.9 (+1.4)**	**99.2 (+3.1)**	**99.5 (+1.1)**

3) Finetuning on Fisheye Images

Detecting Traffic Signs from fisheye-lens images is more difficult due to the distortion that occurs on the sides. In order to familiarize the model with that type of distortion, the pretrained weights from the previous section were finetuned with the same settings for 100 epochs on a hybrid training dataset containing all the training images from GTSDB and 156 handpicked images from the Valeo Woodscape dataset.

The resulting model performs slightly worse on the GTSDB test dataset, but yields significantly better results on the custom FU-Berlin testing dataset. The resulting model will be referred to as "V7Fisheye". The comparison table reveals that familiarizing the model with fisheye images resulted in a 10.3% mAP increase on the custom test dataset. For the final table results, the confidence threshold was raised to 0.15 for enhanced precision.

Class	Test Dataset	Precision	Recall	mAP@.5
V7Best	**GTSDB Test**	98.9%	99.2%	99.6%
V7Fisheye	**GTSDB Test**	98.3%	98.1%	99.5
V7Best	**FU-Berlin Test**	84.3%	84.3%	82.2%
V7Fisheye	**FU-Berlin Test**	**90.2%**	**93.1%**	**92.5%**

4.4 Composing the Full System

With the trained detector and classifier ready for deployment, they're combined into a single system. Two versions of the full system will be evaluated, "V7BestFS" and "V7FisheyeFS". The detector finds bounding boxes of traffic signs in an input image, crops the boxes, then resizes the cropped traffic signs to size 32x32. The resized crops are then normalized and inputted into the classifier. The classifier returns the traffic sign label, and the final result is a list of shape [x1,y1,x2,y2,class].

5 Full System Evaluation

5.1 Performance Evaluation

Until now, the detection models were evaluated with all class labels set to 0. But to evaluate the full system, the fully-labeled GTSDB test dataset is now used. The custom FU-Berlin dataset can't be used for full system evaluation because most images include the "No Stopping" Traffic Sign, which isn't included in the classification dataset. Instead, detection result images will be presented at the end of this section. the F1 graphs recommend 0.75-0.8 confidence for best precision-recall balance. All following test results were obtained with confidence threshold 0.75. The table below illustrates the final results. See Appendix Fig: **??** and Fig: 18 for the Precision, Recall, F1, and PR graphs of each system.

Model	Precision	Recall	mAP@.5	mAP@.5:.95
V7BestFS	0.99	0.99	0.99	0.828
V7FisheyeFS	0.971	0.938	0.935	0.791

Table 4: Full System performance on all 43 classes of GTSDB.

The custom classifier demonstrates excellent performance. The mAP barely drops by changing from single-class detection to 43-class detection. The complete confusion matrices of both runs Fig: 10 and Fig: 11 reveal that V7BestFS made only a single classification error, while V7FisheyeFS made two. The confusion matrices also reveals a major problem in the system, the classifier will always confidently predict a traffic sign even when presented with a false positive by the detector. Even though the precision and recall are quite high, a false positive being interpreted as a traffic sign could lead to critical situation in autonomous driving. V7BestFS had 4 false positives and 4 false negatives, while V7FisheyeFS had 5 false positives and 8 false negatives.

This problem of confident classification of false negatives is likely to be even worse in real-world tests, because the detector will detect many types of traffic signs that the classifier isn't familiar with. But the classifier will still confidently classify a traffic sign it never trained on. This issue is explorer further in the final subsection.

5.2 Efficiency Evaluation

The system's speed is dependent on the application, since I/O can be the largest bottleneck to performance. The measured total run time for the system was 25.7 ms per 1280x1280 image. This is broken down to 20.2ms for detection, 2.7 ms for classification (including cropping, resizing, and normalization), 1.2 for non-maximum suppression, and 1.6 ms for miscellaneous operations. With 25.7 ms processing time the system should theoretically be easily capable of 30 FPS processing, but due to image I/O bottlenecks the highest observed speed was 27 FPS.

5.3 Comparison to Existing Models

The full system is very competitive when compared to the existing models. To the best of my knowledge, the best opensource detection model for GTSDB is Faster R-CNN Inception Resnet V2 by Alvaro Arcos-Garcia et al. [5] which achieved 95.77% mAP at 2.26 FPS. As for the state-of-the-art traffic sign detection model, Hamed Aghdam et al. [14] reported the highest known mAP of 99.8% on GTSDB with over 30 FPS. Another interesting comparison is a YOLOv3 implementation heavily customized for GTSDB by Yawar Rehman et al. [29] which achieved 93.09%.

It should be noted, all the aforementioned results were calculated only on the 4 superclasses (mandatory, prohibotory, danger, and other) while our system detects and classifies all 43. Despite this handicap, V7BestFS outperforms most known the best opensource implementation by scoring 99.0% on Recall, Precision, and mAP. To the best of my knowledge, the state-of-the-art implementation by Adgham et al. is the only real-time neural network that outperforms V7BestFS on GTSDB.

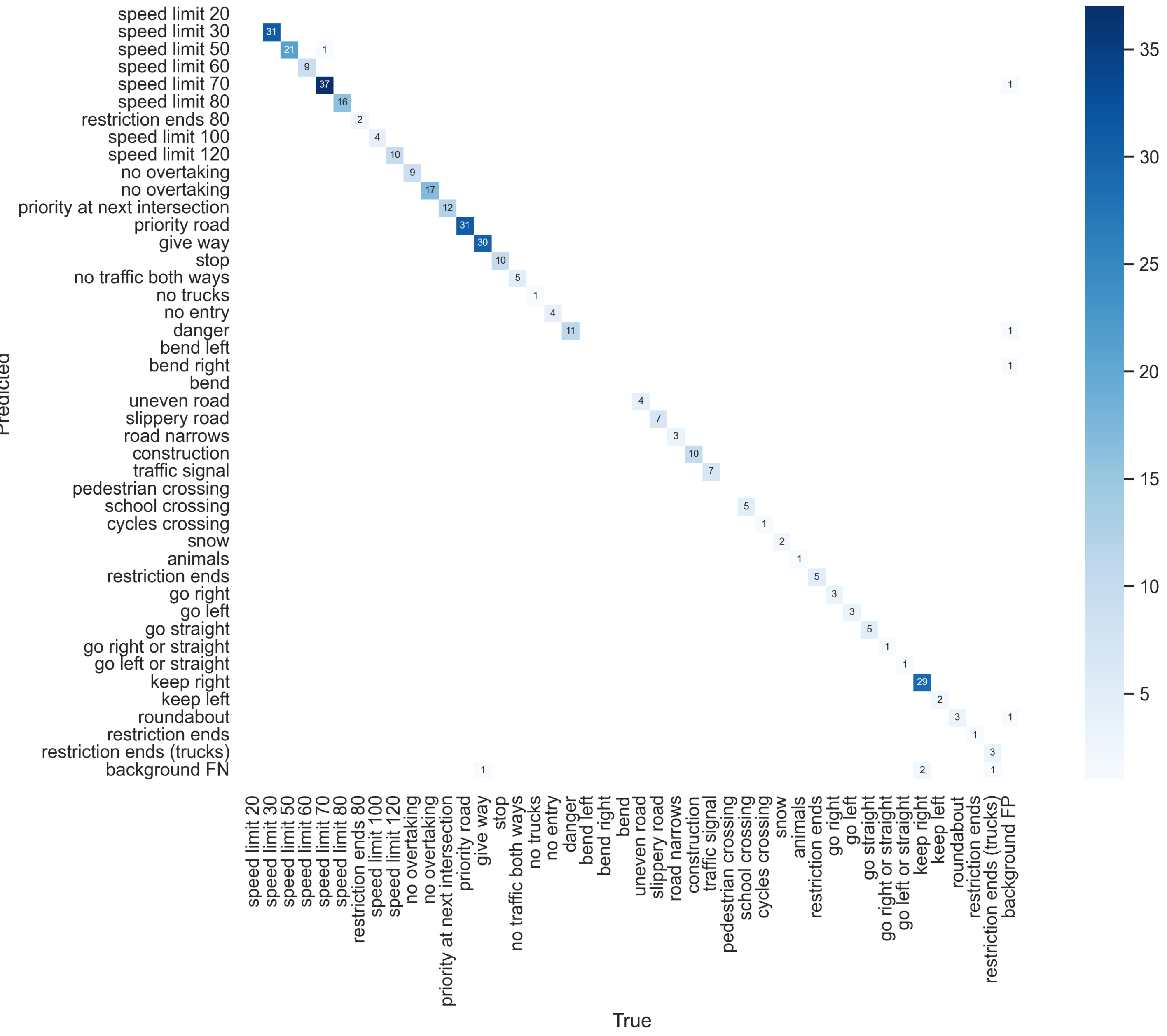


Figure 10: V7BestFS confusion matrix on all 43 classes.

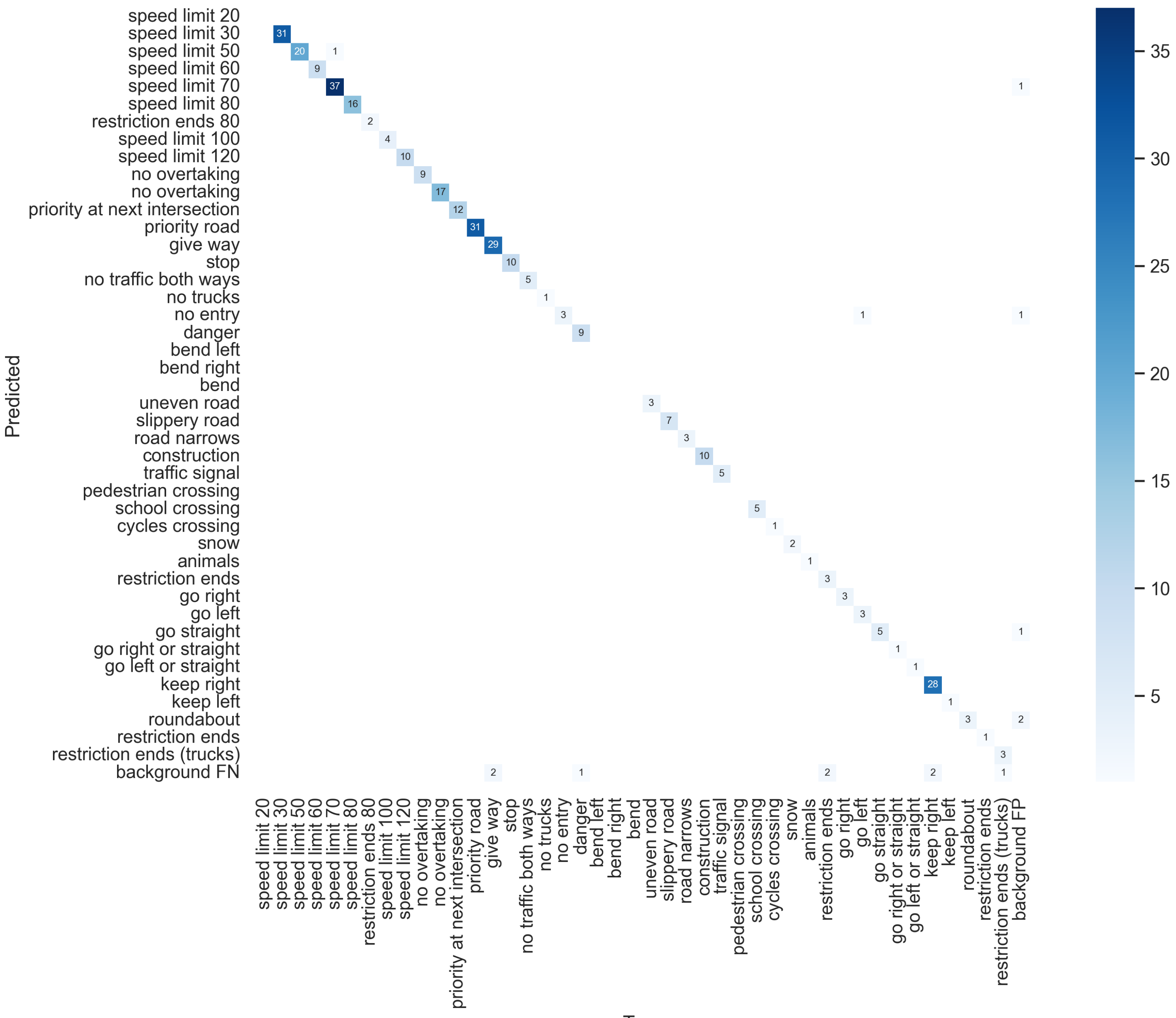


Figure 11: V7FisheyeFS confusion matrix on all 43 classes.

5.4 Analysis on Test Images

The final evaluation will be reviewing detection results of the system on fisheye-lens images from the custom FU-Berlin test dataset and night photos from the Italian Traffic Signs Dataset (DITS) [16]. Failure cases will be displayed and reviewed. The following detections were produced by V7FisheyeFS with confidence threshold 0.15.

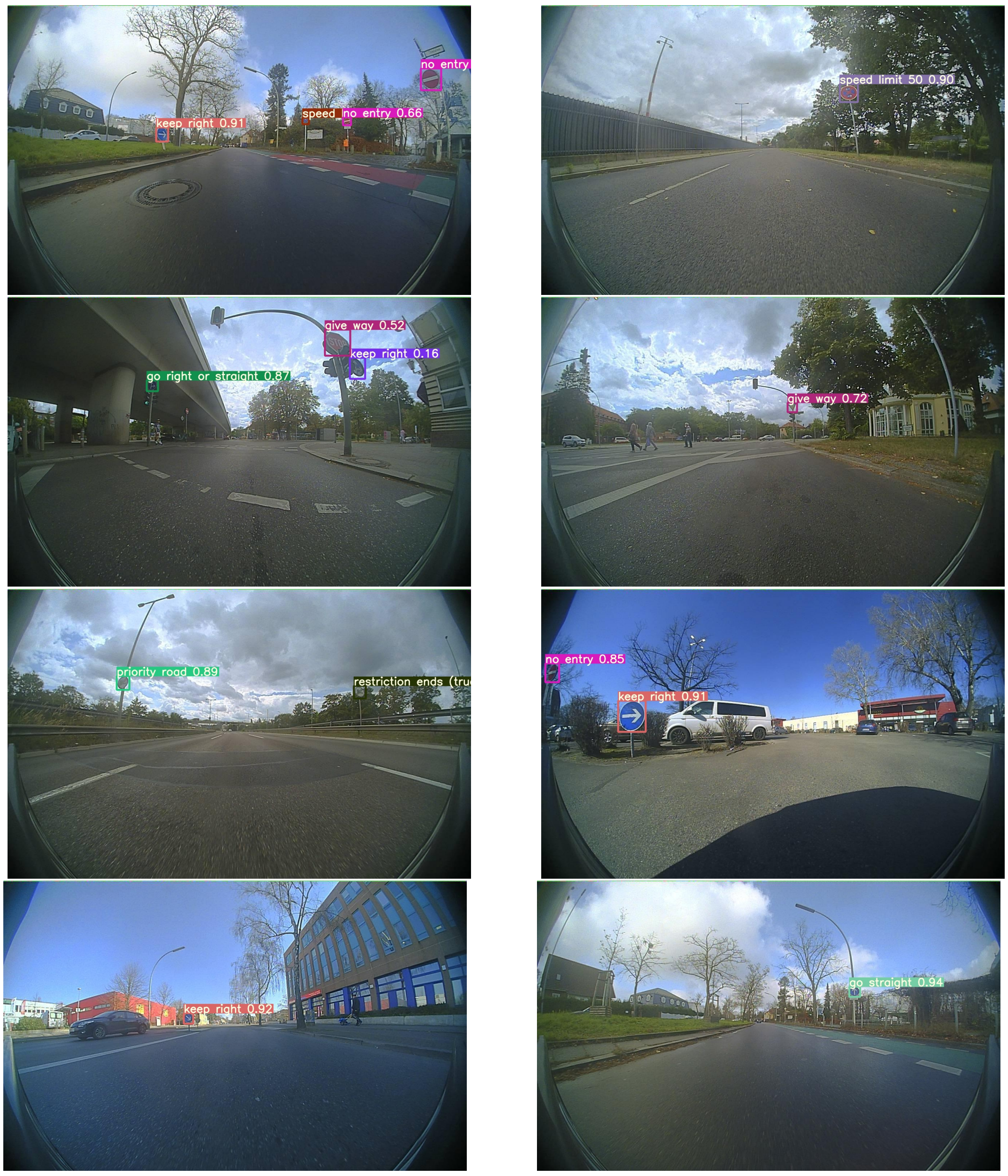


Figure 12: Sample images from the custom FU Berlin dataset.

Training V7FisheyeFS on fisheye-lens images paysoff, and the system is capable of detecting even small traffic signs on the extreme ends of the image (see 6th image). But every time the system in confronted with a "No Stopping" traffic sign, a random class is assigned. Additionally, a STOP sign is classified as give way in the 3rd image.

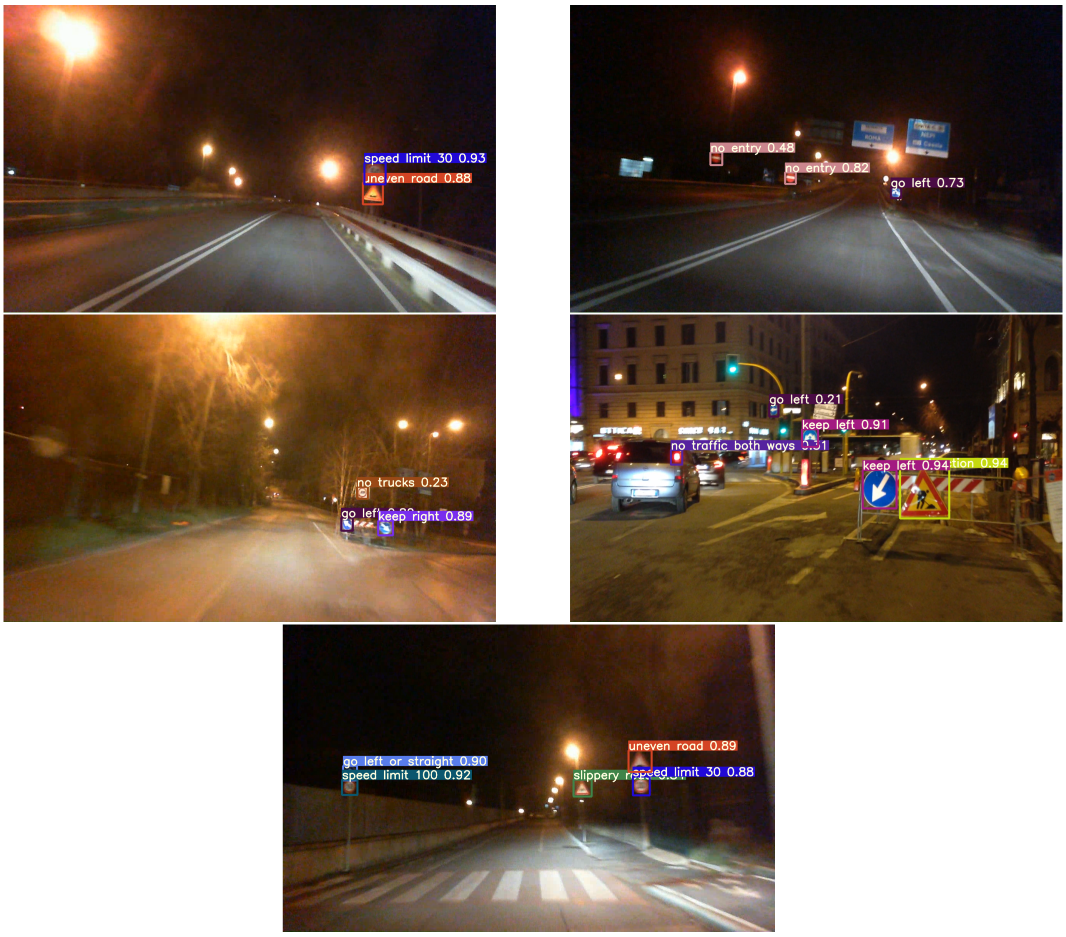


Figure 13: Sample images from DITS.

The system manages decent results on night photos considering it was never trained on them. It doesn't fail to detect any traffic signs, but the unique light intensities in dark environment lead to some false positives. One interesting failure of the system is the detection of a car's tail-light as a traffic sign in the 4th image. The limited number of traffic signs the classifier is capable of labeling is also on display in the 4th image, where the go-left-or-right sign is labeled go left.

6 Conclusion and Future Work

The final full system is promising and manages to achieve the three criteria established in the introduction. It is capable of real time processing (30 FPS) excluding I/O, is moderate in size, and has high recall and accuracy. It achieves highly competitive scores on GTSDB, and holds its ground when confronted with images in environments it wasn't trained on. The results confirm YOLOv7-W6's potential at traffic sign detection. But the limited number of traffic signs the classifier can recognize represents a critical flaw for the system if it were to be deployed for real-world use. Moreover, the classifier will confidently classify false positives by the detector as some traffic sign.

A future point of interest should be training the classifier to recognize more traffic sign classes (especially the "No Stopping" traffic sign). A new traffic sign classification dataset will need to be produced to this end, as none of the current public traffic sign classification datasets have an adequate number of traffic sign classes. Training the classifier on random background crops from traffic image would also help the classifier recognize the cases in which the detector makes a mistake.

Another approach that could improve performance is integrating past predictions into the system's confidence of the current prediction. This can be done using a Kalman Filter. By calculating the expected position of the traffic sign as an offset by velocity, the consistency of the system's predicitons can be improved. It would also allow the system to determine whether a detected traffic sign is new or was available in previous frames.

References

[1] URL: https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html.

[2] Chien-Yao Wang et al. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2020. URL: https://openaccess.thecvf.com/content_CVPRW_2020/papers/w28/Wang_CSPNet_A_New_Backbone_That_Can_Enhance_Learning_Capability_of_CVPRW_2020_paper.pdf.

[3] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. URL: https://arxiv.org/pdf/1311.2524.pdf.

[4] Álvaro Arcos-García, Juan Alvarez-Garcia, and Luis Soria Morillo. "Evaluation of Deep Neural Networks for traffic sign detection systems". In: *Neurocomputing* 316 (Aug. 2018). DOI: 10.1016/j.neucom.2018.08.009.

[5] Álvaro Arcos-García, Juan Alvarez-Garcia, and Luis Soria Morillo. "Evaluation of Deep Neural Networks for traffic sign detection systems". In: *Neurocomputing* 316 (Aug. 2018). DOI: 10.1016/j.neucom.2018.08.009.

[6] *Belgian Traffic Signs Classification Dataset*. URL: https://btsd.ethz.ch/shareddata/.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[8] Gaudenz Boesch. *Object Detection in 2022: The Definitive Guide*. 2022. URL: https://viso.ai/deep-learning/object-detection/.

[9] *Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods*. URL: https://www.sciencedirect.com/science/article/abs/pii/S0893608018300054?via%5C%3Dihub.

[10] *German Traffic Signs Benchmark by INI*. URL: https://benchmark.ini.rub.de/index.html.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[12] Andrii Gozhulovskyi. *Classification of Traffic Signs with LeNet-5 CNN*. 2022. URL: https://towardsdatascience.com/classification-of-traffic-signs-with-lenet-5-cnn-cb861289bd62.

[13] Jiuxiang Gu et al. *Recent Advances in Convolutional Neural Networks*. 2015. arXiv: 1512.07108. URL: http://arxiv.org/abs/1512.07108.

[14] Hamed Habibi Aghdam, Elnaz Jahani Heravi, and Domenec Puig. "A practical approach for detection and classification of traffic signs using Convolutional Neural Networks". In: *Robotics and Autonomous Systems* 84 (2016), pp. 97–112. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2016.07.003. URL: https://www.sciencedirect.com/science/article/pii/S092188901530316X.

[15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] *Italian Traffic Signs Dataset*. URL: http://www.diag.uniroma1.it//~bloisi/ds/dits.html.

[17] Max Jaderberg et al. *Spatial Transformer Networks*. 2015. arXiv: 1506.02025. URL: http://arxiv.org/abs/1506.02025.

[18] Sambasivarao. K. *Non-maximum Suppression (NMS)*. URL: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

[19] et al. Kaiming He Xiangyu Zhang. *Deep Residual Learning for Image Recognition*. 2015. URL: https://arxiv.org/pdf/1512.03385.pdf.

[20] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030. URL: https://arxiv.org/abs/2103.14030.

[21] *Mish: A Self Regularized Non-Monotonic Activation Function*. 2020. URL: https://www.bmvc2020-conference.com/assets/papers/0928.pdf.

[22] Michael Nielsen. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/index.html. 2019.

[23] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *CoRR* abs/1511.08458 (2015). URL: http://arxiv.org/abs/1511.08458.

[24] *Object Detection on COCO test-dev*. 2022. URL: https://paperswithcode.com/sota/object-detection-on-coco.

[25] *Official YOLOv7 Training Code and CoCo Weights*. URL: https://github.com/WongKinYiu/yolov7.

[26] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[27] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[28] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[29] Yawar Rehman, Hafsa Amanullah, and et al. Saqib Bhatti. "Detection of Small Size Traffic Signs Using Regressive Anchor Box Selection and DBL Layer Tweaking in YOLOv3". In: *Applied Sciences* 11.23 (2021). ISSN: 2076-3417. DOI: 10.3390/app112311555. URL: https://www.mdpi.com/2076-3417/11/23/11555.

[30] et al. Safat B. Wali Majid Abdullah. *Vision-Based Traffic Sign Detection and Recognition Systems: Current Trends and Challenges*. 2019. URL: https://www.mdpi.com/1424-8220/19/9/2093/htm#B17-sensors-19-02093.

[31] *Spatial Transformer Networks A Self-Contained Introduction*. URL: https://towardsdatascience.com/spatial-transformer-networks-b743c0d112be.

[32] *Traffic Sign Classification Using Deep Inception Based Convolutional Networks*. 2016. URL: https://arxiv.org/abs/1511.02992.

[33] *Valeo Woodscape*. URL: https://woodscape.valeo.com/download.

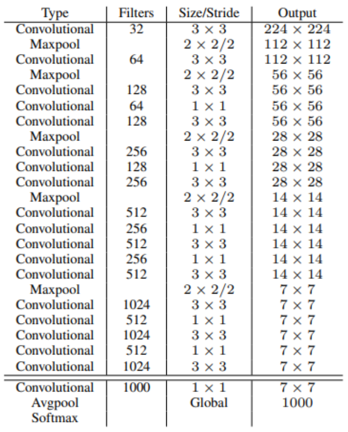
[34] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2020. arXiv: 2011.08036. URL: https://arxiv.org/abs/2011.08036.

[35] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. DOI: 10.48550/ARXIV.2207.02696. URL: https://arxiv.org/abs/2207.02696.

[36] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. *You Only Learn One Representation: Unified Network for Multiple Tasks*. 2021. arXiv: 2105.04206. URL: https://arxiv.org/abs/2105.04206.

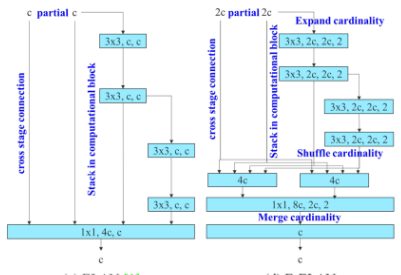
A Appendix

A.1 YOLO Figures



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 14: Architecture of Darknet-19, YOLOv2's backbone. Image Source [26]



(c) ELAN [1]   (d) E-ELAN

Figure 15: ELAN and Extended ELAN blocks architecture. E-ELAN is only used in YOLOv7-E6E. Image Source [35]

A.2 GTSDB and GTSRB List of classes

GTSRB and GTSDB full list of classes:

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)

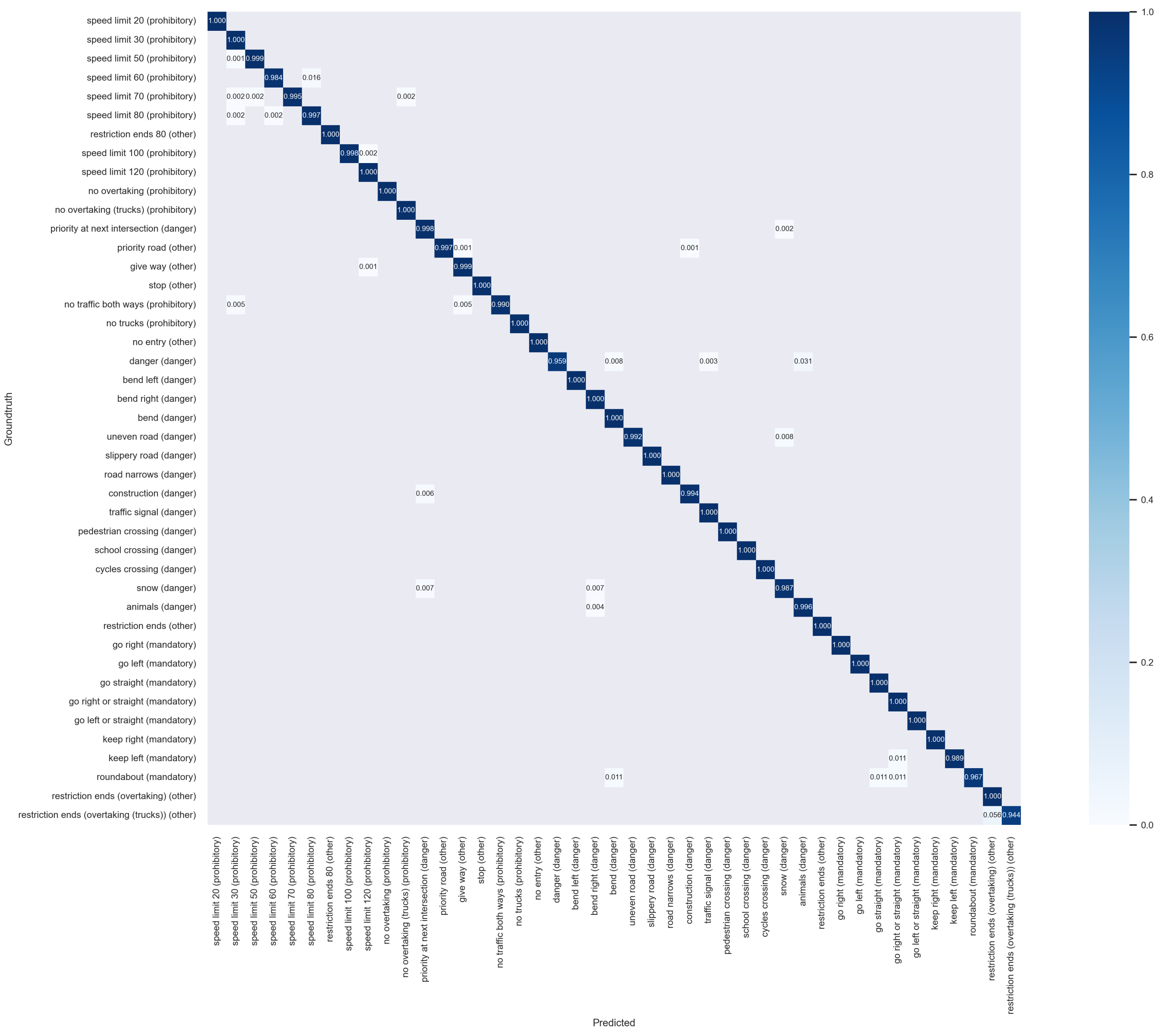


Figure 16: Confusion matrix of the CustomTS classifier with all 43 classes.

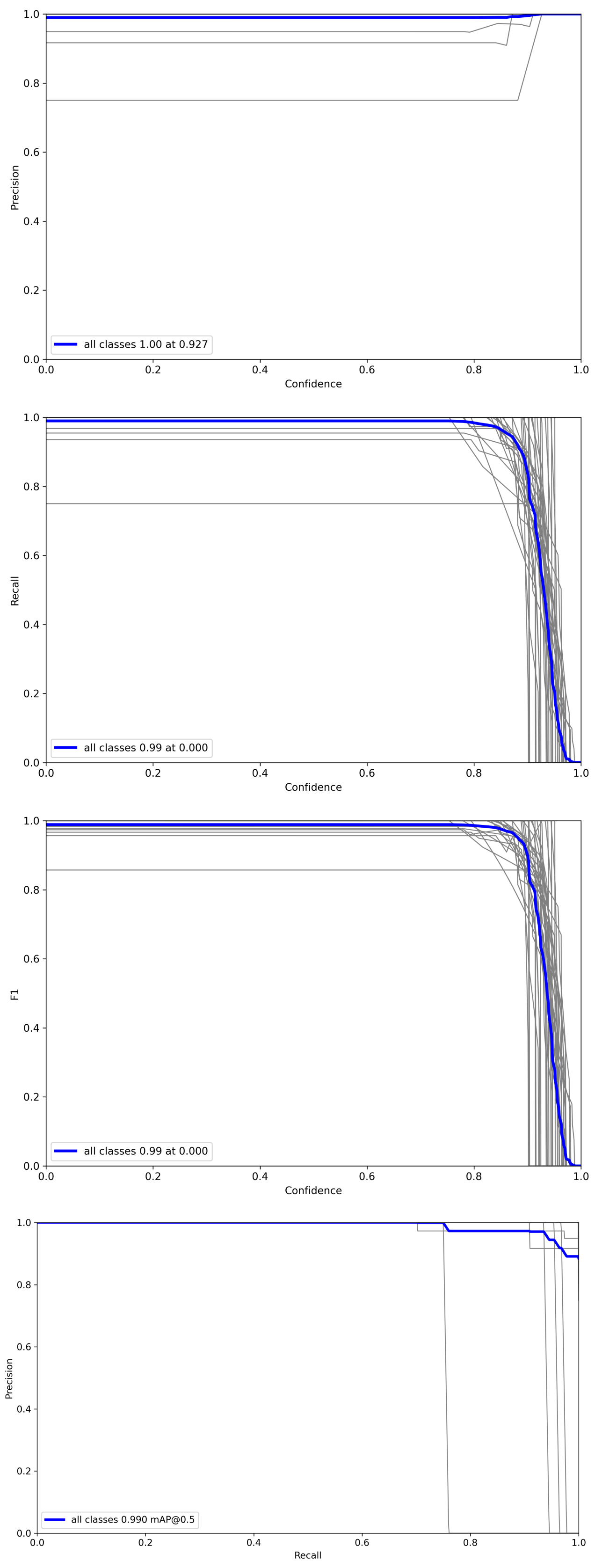


Figure 17: Precision, Recall, F1, and PR graphs of V7BestFS.

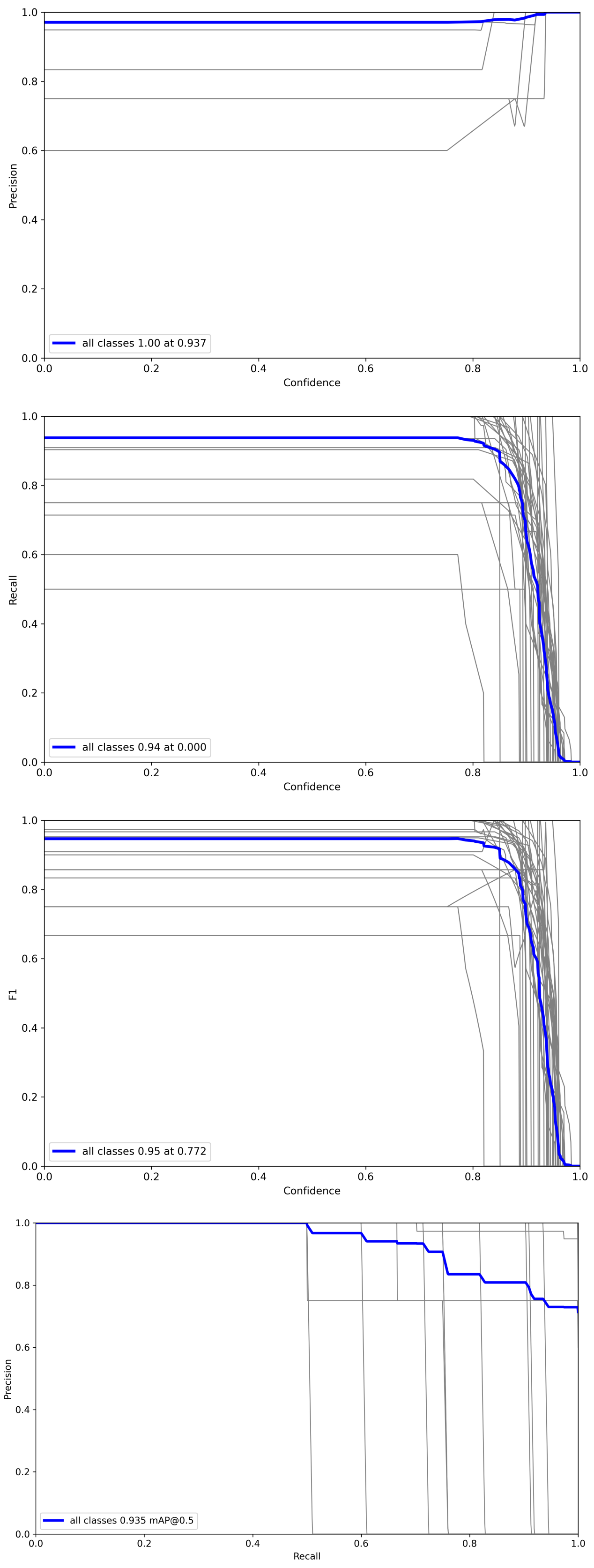


Figure 18: Precision, Recall, F1, and PR graphs of V7FisheyeFS.