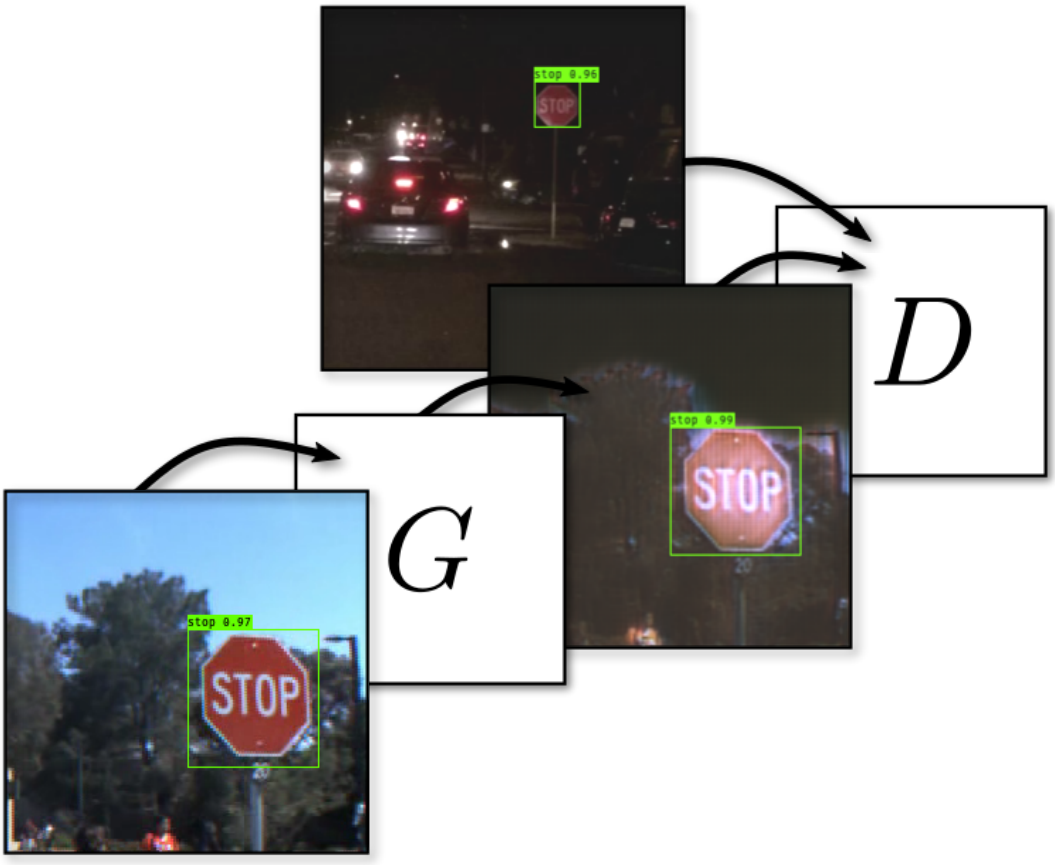


Generative Adversarial Networks for Object Detection in AD/ADAS Functions

Global Capstone Project with Chalmers University of Technology, University of California, Berkeley and Volvo Cars

Bachelor's Thesis in Department of Electrical Engineering

ROBIN HALFVORDSSON, JONATAN NORDH, ADAM SUHREN GUSTAFSSON,
JOEL WALL, MATTIAS WESTERBERG, ADAM WIREHED

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Bachelor's Thesis 2019: EENX15-19-21

Generative Adversarial Networks for Object Detection in AD/ADAS Functions

Global Capstone Project with Chalmers University of Technology, University of California, Berkeley and Volvo Cars

Robin Halfvordsson
Jonatan Nordh
Adam Suhren Gustafsson
Joel Wall
Mattias Westerberg
Adam Wirehed

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
EENX15-19-21
Chalmers University of Technology
Gothenburg, Sweden 2019

Generative Adversarial Networks for Object Detection in AD/ADAS Functions
Global Capstone Project with Chalmers, Berkeley and Volvo Cars
Robin Halfvordsson, Jonatan Nordh, Adam Suhren Gustafsson,
Joel Wall, Mattias Westerberg, Adam Wirehed

Supervisor: Jonas Sjöberg, Professor, Electrical engineering
Supervisor: Lars Tornberg, Senior Machine Learning Engineer, Volvo Car Group
Examiner: Knut Åkesson, Professor, Electrical engineering

Bachelor's Thesis 2019: EENX15-19-21
Department of Electrical Engineering
Division of Systems and Control
EENX15-19-21
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image transformed from daytime to nighttime through the use of a Generative Adversarial Network. The detections of the object detection system "YOLO" are also visible.

Typeset in LaTeX
Gothenburg, Sweden 2019

Abstract

Traffic sign identification using machine learning algorithms with the help of camera images from vehicles is key towards autonomous path planning and driving. However, changing e.g. weather and lighting conditions from what is available in the training domain can lead to deteriorating detection performance and collecting/labelling more data in the new domain is time consuming and expensive. In this work, we present an image augmentation method based on "Generative Adversarial Networks" (GANs) to augment traffic sign training data. Our method is used to map a training dataset from one domain into another. We exemplify the proposed method by augmenting daytime images to nighttime images. Daytime images from the LISA dataset, containing traffic signs, is used for training and nighttime images from the BDD-Nexar dataset is used for end to end testing. We also compare against two alternative augmentation methods which utilizes no machine learning as well as two methods based on a separate GAN and reinforcement learning respectively. Our method is able to improve the resulting detection precision/recall from 0.70/0.66 to 0.81/0.70 on night images while also slightly improving the performance on the day time images.

Keywords: Generative Adversarial Networks, AD, ADAS, Traffic sign, Machine learning, Object detection.

Acknowledgements

We in the Chalmers team could not have accomplished this project on our own. We would like to thank Volvo Cars for this opportunity - for inviting us to their office in Sunnyvale and for sponsoring our trip to the University of California, Berkeley. We would also like to thank the Berkeley supervisor Sohini Roy Chowdhury (Volvo Cars) as well as our own supervisors Jonas Sjöberg (Chalmers) and Lars Tornberg (Volvo Cars) for all the help and support during the project.

This project was done in collaboration with four students from the University of California, Berkeley. We would like to thank these students for their cooperation on the project and their hospitality during our visit.

Glossary

AutoAugment
: Machine learning method for enhancing diversity in images through reinforcement learning

BBGAN
: (Bounding Box GAN) GAN for unpaired image-to-image translation using bounding box specific losses

BDD
: (Berkeley Deep Drive) public dataset mainly developed by Path Institute, University of California, Berkeley

Blender
: Open source 3D-modelling software developed by the Blender Foundation

CNN
: Convolutional Neural Network

GAN
: Generative Adversarial Network

IoU
: Intersect of Union

LISA
: Grouping of the two traffic sign datasets, LISA-TS and LISA Extensions. Constructed by Laboratory for Intelligent & Safe Automobiles

Nexar
: Dataset of driving scenes in day and night environments, without annotations

ODS
: Object Detection System

SimpleAugment
: Python script which uses simple transformations and color adjustments to simulate nighttime environment

TensorFlow
: Open source machine learning library developed by Google

YOLO
: (You Only Look Once) object detection algorithm introduced in 2015

Contents

1. Introduction

In recent years generative machine learning models like Generative Adversarial Networks (GANs), first proposed by Ian Goodfellow in 2014 [1], have become increasingly more advanced and well-understood. These allow for generation and manipulation of realistic images, blurring the line between real and generated data. But have these algorithms become sophisticated enough to generate images for use when training an Object Detection System (ODS) which can be used in autonomous vehicles? This project will serve as an exploration for how GANs can be used to augment images containing traffic signs to improve the performance of an ODS.

1.1 Background

During the last decade rapid developments have been made in the fields of Autonomous Driving (AD) and Advanced Driver-Assistance Systems (ADAS). Features like lane centering assist and rear-view video systems e.g. are now commonly available in many commercial vehicles [2]. The step to complete AD technology comes with many obstacles. One being the processing of camera images for use in an ODS to detect and classify objects like traffic signs, pedestrians, vehicles etc. This ODS requires high accuracy as a resulting misclassification may endanger human lives.

The ODS can be implemented using Deep Learning models structured as artificial neural networks. The principle of such system is explained in Figure 1.1. With an image as input where a Pedestrian Crossing and Speed Limit 25 sign is visible, the algorithm "scans" the image looking for known patterns. If the image contains detectable patterns the network will yield an output containing bounding boxes (rectangles) of the signs, classifications (class) of the signs (e.g stop or parking) and the confidence of the prediction quantified as a number between 0 and 1.

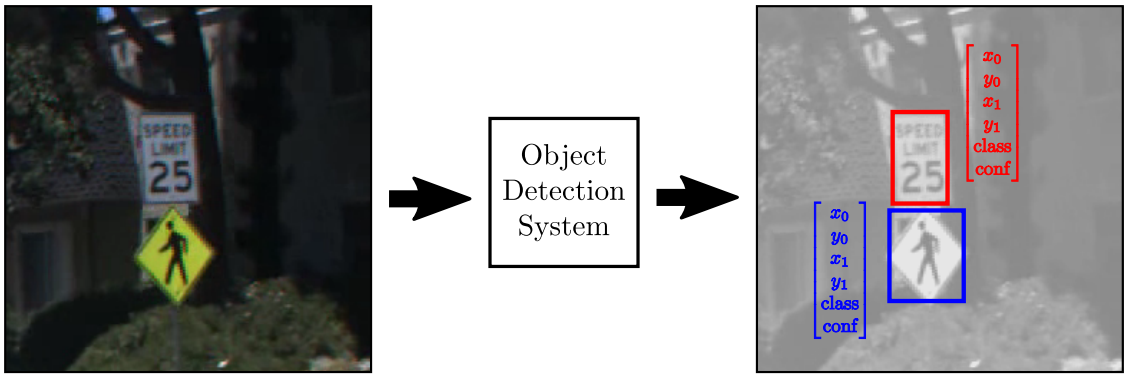


Figure 1.1: *Example of input and output of an ODS.*

To attain the required accuracy a large amount of training data is necessary to train the neural network. This data is usually collected as video and images with car-mounted cameras from public roads. These images are then annotated with the labels of the signs (e.g. stop) and the screen coordinates where the signs are located. This information is necessary for supervised training as the network will learn from the coordinates and labels to detect and classify objects. Current supplies of available training data is not always sufficient to train ODSs. Often the ODSs will be presented with non-descript real-world examples like low resolution images, partially occluded signs and/or damaged objects such as vandalized traffic signs. In addition it may encounter out-of-distribution examples like nighttime and snowy settings. These non-descript and out-of-distribution examples are usually underrepresented in the training datasets which leads to the ODS not learning and generalizing for said examples.

1.2 Contributions

The ODS used in the project is You Only Look Once (YOLO) [3]. The ODS is trained on daytime images from the LISA Traffic Sign Dataset (LISA) [4] to establish a baseline model and identify non-descript misclassifications. Most of the misclassifications are similar to the example showed in Figure 1.2 where a partially shadowed stop sign is not detected. The shadow makes the stop sign too dark for the network to detect and classify. More examples of misclassifications are shown in appendix C.2.

Figure 1.2: *Non-descript sign example found in the real world.*

For the ODS to classify non-descript signs the training dataset has to contain images of similar cases. This project's focus is to improve the object detection of the out-

of-distribution case of traffic signs in low light or nighttime conditions which is not represented in the LISA dataset.

The method used to produce nighttime examples for the ODS training dataset is through the use of image-to-image translation using a GAN. This method has shown promise in generating artificial images that looks convincingly similar to authentic ones taken by camera or painted by hand [5]. Therefore a GAN is developed with a tailored loss function for this use case. This GAN is used to convert daytime scenes with traffic signs to nighttime conditions as can be seen in Figure 1.3. This augmentation allows for the use of existing labels and sign locations in the resulting images as both the class and location is preserved. In addition, the loss function utilizes the bounding box information in the source domain to preserve the content of the signs during augmentation.

Figure 1.3: *Transformation from a real world image on the left to the corresponding GAN generated image simulating nighttime conditions on the right.*

The purpose of the new data is to improve performance of YOLO on nighttime images compared to the baseline model. YOLO is tested on a dataset containing both real daytime and nighttime images. In addition, different instances of YOLO are compared to each other. These instances are trained on unique datasets that are produced by alternative methods of data augmentation. Evaluation of the results shows that the generated data from the GAN slightly improves YOLO's object detection on nighttime images while maintaining daytime performance.

1.3 Collaborators

This work was done as part of a Global Capstone Project between Volvo Cars, Chalmers University of Technology and University of California, Berkeley. Volvo Cars is a car manufacturer with research and development centers in both Sweden and USA. They are the industrial client and partner in this cooperation. At the beginning of this cooperation the teams established a benchmark on LISA with YOLO. It was decided that the Berkeley team would focus on the data augmentation method AutoAugment [6] while the Chalmers team's main focus would be the GAN.

2. Theory

Developing and evaluating a GAN requires understanding of machine-learning and neural networks. These networks can be constructed using different mathematical models, such as convolutions, depending on the use case. These models will be described in this chapter in order to understand the different GAN models in this project. The GAN developed by the Chalmers Team is based on the theory presented here.

2.1 Generative Adversarial Network

GANs are a class of artificial intelligence algorithms used in unsupervised machine learning. The model consists of two neural networks contesting each other and can be used to generate artificial images for use as training data for classifiers.

The two neural networks are called the generator and the discriminator, which are represented by the boxes G respectively D in Figure 2.1. The discriminator operates as a binary classifier which labels the presented image as real or fake (generated). In a GAN the discriminator is often a convolutional neural network with an image as the input and a label (prediction), real or fake, as the output. With a trained network it will find different features in the image. The network is not necessarily looking for features that a human would look for, like shapes or colors, but self-learned features developed during training. With these features the discriminator will find similarities to images from the training and make a prediction. The generator is a network that generates artificial images of authentic objects. The input is a vector of noise, or an image, represented by $\boldsymbol{z}$ in the bottom left of Figure 2.1. The generator will perform operations on $\boldsymbol{z}$ in order to transform it into the desirable image $\boldsymbol{x}$.

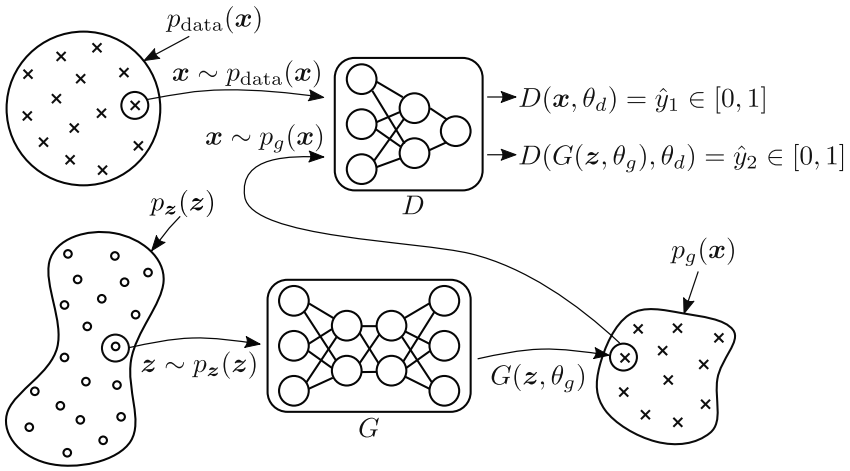


Figure 2.1: *Schematic view of a GAN*

To learn the generator's distribution p_g over data $\boldsymbol{x}$, a prior (probability distribution) on the input variables $p_z(\mathbf{z})$ is defined. The mapping to data space is represented by $G(\boldsymbol{z}; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron (neural network) with trainable parameters θ_g. A second multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ is defined that outputs a single scalar $\hat{y}$ between 0 and 1. Here $D(\boldsymbol{x})$ is the probability that $\boldsymbol{x}$ came from the real data rather than p_g. D is trained to maximize the probability of assigning the correct label to both training examples (the data $\boldsymbol{x}$ in the top left) and samples from G ($\boldsymbol{x}$ in the bottom right). G is trained to maximize $\log D(G(\boldsymbol{z}))$. If G and D have enough capacity (unlimited memory), after several training steps they will reach a point where both cannot improve hence $p_g = p_{data}$. The discriminator will then be unable to differentiate between the distribution and will give the prediction $D(\boldsymbol{x}) = \frac{1}{2}$ [1].

GANs does not only have to contain one generator and one discriminator. There are a lot of different kinds of GANs that extends the base architecture. The reason behind this is to improve performance in more specific tasks. One example is the CycleGAN that contains two discriminators and two generators [7].

Modern deep learning algorithms typically require many labeled examples to generalize well. GANs are a possible solution to these problems. The network can be trained on data that is underrepresented or not included in the datasets. The GAN will then generate artificial images of the missing data [8]. This way GANs adds the missing data necessary to train and develop accurate classification systems.

2.2 Convolutional Neural Networks

Currently there are many different variations of neural networks. Different mathematical models can be implemented as layers in these networks depending on the use case. In most scenarios the GAN is supposed to generate images represented as a 2D grid of pixels (raster). Convolutional Neural Networks (CNNs) are currently the most used network for this task because of their efficiency.

To classify or generate images, one must reduce redundant information and extract the important features. With a CNN this is done with convolutional layers in combination with activation functions and batch normalizations.

2.2.1 Convolutional Layers

All neural networks consist of layers of nodes (neurons). The number of layers and nodes differs between networks. Generally the first layer of an image-classifying CNN has the same number of nodes as pixels in the input image. In traditional neural networks all layers are fully connected. This means that every node in one layer is connected to every node in the next layer. In Figure 2.2 the nodes highlighted in gray are connected to each other, where the first system is a convolutional network and the second is a standard neural network. Each unique connection has a weight

(parameter) assigned to it. The value of the weight is a representation of how much a specific node will affect another.

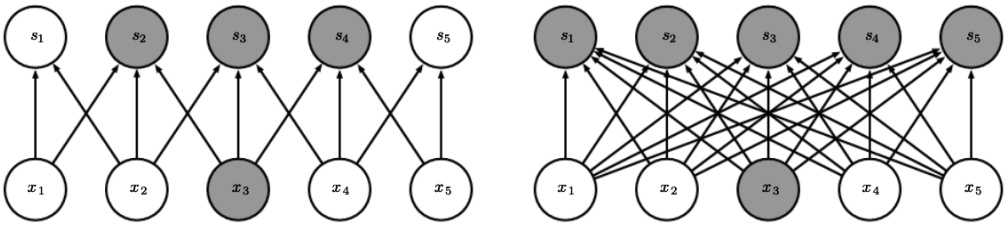


Figure 2.2: *Difference in node-connections between convolutional neural network (left) and traditional (right) [9, p. 331]*

The fully connected network uses matrix multiplication between the layers. Meaning that every output unit interacts with every input unit [9]. In many cases the output connections will not affect the input of the nodes since the value of the weights after training will be close to zero. This requires unnecessary processing power and memory for calculations that will not affect the outcome (the generated image). The convolution aspect of the method is that the regular matrix multiplication between the layers is replaced by a sparse matrix multiplication, often called a kernel operation.

The kernel is a multidimensional array of parameters. Similar to the weights in a traditional neural network, these parameters are used to change how much the nodes are affecting each other. In Figure 2.3 the parameters in the kernel *w, x, y, z* also change their value based on the training. The nodes *a, b, e, f* and the kernel are used in dot-multiplication where the output is the connected node in the next layer. This is why one node is not connected to all the nodes in the previous layer. Using a CNN for both the generator and discriminator makes GANs more effective.

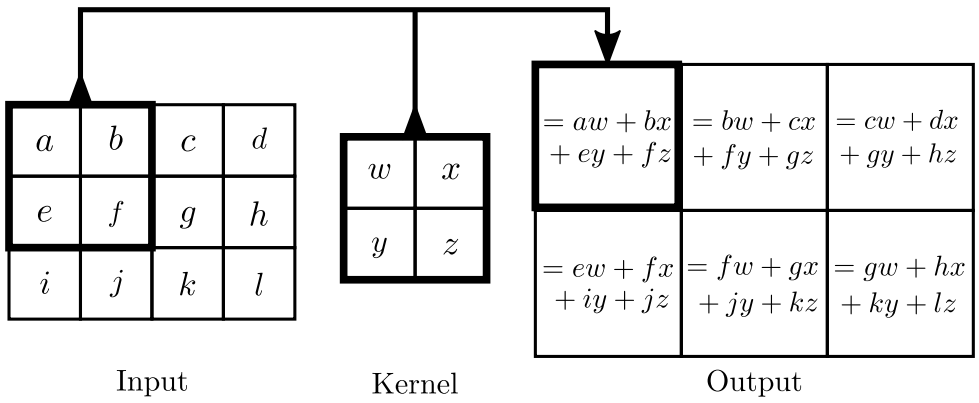


Figure 2.3: *Visual example of a convolution operation in neural networks*

2.2.2 Leaky Rectified Linear Units

The rectifier is an activation function with the purpose of improving the training of neural networks. It allows for faster and more effective training of deep neural architectures on large and complex datasets. More specifically, it helps the network account for interaction effects when one variable A affects a prediction differently depending on the value of B [10]. It also makes it easier for the network to account for non-linear effects. The leaky variant of this activation function is called Leaky Rectified Linear Units (Leaky ReLU) and defined as:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{2.1}$$

The parameter x is the value of a node in the network and α is the slope of the rectified function. The function is often used together with a convolutional layer. After the kernel operation in the convolutional layer the nodes from that layer is put through the activation function in order to suppress negative values.

2.2.3 Batch Normalization

In order to increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation [11]. This results in preventing small changes to the parameters from amplifying into larger and suboptimal changes in activations (such as the Sigmoid function) in gradients. Not only does the network become more stable, but also drastically improves the training speed and reduces overfitting.

2.3 Loss Functions

Loss functions (cost functions) maps variables into numbers representing some form of loss. This is the function that the network tries to minimize during the training. In machine learning, these functions are used to measure the difference between the predicted output and the desired output [12]. For GANs the loss function is used to measure the discriminator's confidence in it's prediction if the image is real or generated. There are different kinds of loss functions depending on the use case. One function regularly used in the discriminator is the Binary Cross Entropy (BCE):

$$L(\hat{y}, y) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]. \tag{2.2}$$

For each prediction $D(x) = \hat{y} \in [0, 1]$ by the discriminator during training, there will also be a truth $y \in \{0, 1\}$. Here 0 means fake, and 1 real. If the image is fake with $y = 0$ and the discriminator predicts so correctly with $\hat{y} \approx 0$, the value of the loss in 2.2 is low. If the image is real with $y = 1$, and the discriminator still predicts

the image as fake with $\hat{y} \approx 0$), the loss value of 2.2 will be large. The loss function will behave similarly for the reversed case.

2.4 Adversarial Training

GANs can generate training data for other neural networks. However, the GAN itself must be trained first to generate this data. The method used to train a GAN is called adversarial training. When the discriminator classifies a generated image as fake it outputs an error signal as mentioned before. This is sent to the generator to inform that the generated image was not authentic enough. The generator, based on the value from the error signal, understands that this image is not good enough, and needs to improve [13]. The adversarial part of the training is that one of the networks finds the weak points of the other network and exploits them. The weaknesses are improved upon through the error signal until the network has improved enough in that area so it can not be exploited anymore. Mathematically the value of the loss function will be low if the discriminator identifies the generated image and the same goes for the generator if it fools the discriminator. A low value of the loss function will not affect the parameters and/or change the way the networks operate.

2.4.1 Optimization

In order for the network to improve and learn from training it needs to optimize its parameters. To improve the results of the network the value output from the loss function should be as low as possible. This can be achieved by finding the global minimum of the loss function using gradient descent. Often it is done by optimization algorithms such as Adam, which is an extension to stochastic gradient descent [14]. The algorithm tweaks the parameters in the networks G and D in order to minimize their respective loss functions. In a GAN this is formulated through the adversarial objective in (2.3) where the generator wants to minimize the value, while the discriminator wants to maximize it in a zero-sum game. [1]

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}\left[\log(D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log(1 - D(G(\boldsymbol{z}))\right] \tag{2.3}$$

With two parameters, the loss function can be thought of as a 3D surface as in Figure 2.4. The surface represents the value of the loss function depending on the parameters p_1 and p_2. Gradient descent, visualized by the red dot and lines in the image, is used for finding the minimum of a function. In Figure 2.4 finding the global minimum is not too problematic. However a neural network consists of a lot more parameters than two and the loss function is not a 3D surface, but a multidimensional landscape. Having many dimensions makes the finding of the global minimum using gradient descent almost impossible. Instead the finding of a local minimum is sought after.

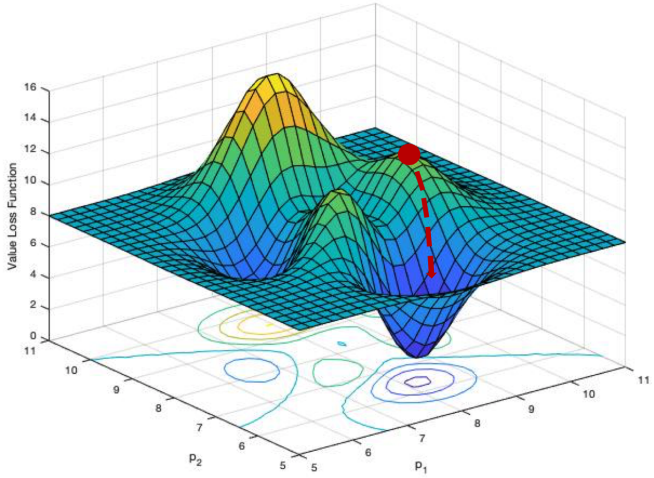


Figure 2.4: *Loss function value, with 2 parameters, displayed as a 3D surface where the red dot is the current value. Gradient decent visualized by the red lines.*

2.4.2 Backpropagation

Backpropagation is a method to calculate a gradient for training the weights of nodes. It uses generalization of the delta rule to multi-layered feedforward networks in order to iteratively compute gradients for the layers in the network [9]. The delta rule is also referred to as the least mean square root. It uses the value from the error signal to compute the needed values for the parameters to get closer to the correct result. In Figure 2.5 is an example of backpropagation where a node in the final layer should have the value 1.0 instead of 0.2. The optimization algorithm then uses backpropagation to increase the values of the positive weights, and decrease the absolute value of negative weights affecting the node in the last layer. This method propagates backwards throughout the network and recursively applies to all the layers. Therefore the algorithm calculates the change needed in the parameters starting in the output layer and going back to the input, hence the name "Backpropagation".

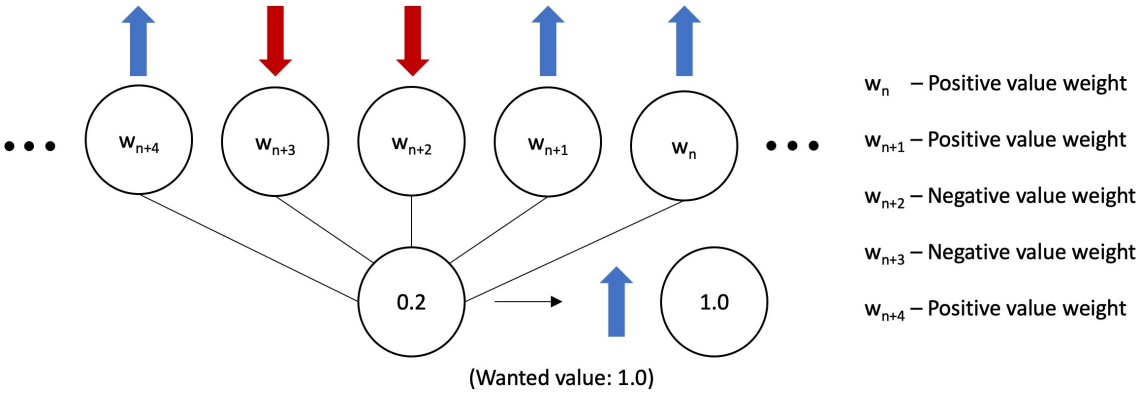


Figure 2.5: *Visual example of backpropagation of a node in the final layer*

3. Methodology

This section describes the methods used in order to improve YOLO's detection and classification rate. It includes dataset preprocessing and how the augmentation methods were built and evaluated.

3.1 Datasets

The objective was to improve the detection rate on the test dataset by training YOLO on the augmented datasets. YOLO was trained on the LISA training dataset containing 7819 images to set a performance baseline to compare the augmented datasets with. This LISA dataset contained both the LISA TS dataset and the LISA Extension dataset [4]. The trained YOLO network was tested on annotated nighttime images in Berkeley DeepDrive [15] and Nexar [16] to test the nighttime detection rate. It was also tested on daytime LISA images to evaluate if the daytime performance was affected by the additional trianing data. The number of images and traffic signs in each dataset is presented in Table 3.1. The data split aimed to have 80% of training images and 20% of testing images. The ideal distribution would be to have the same proportion of training and testing images for each kind of sign, preferably 80%/20%. However the split was done by track and not by a random split of the whole dataset. One track is a scene of multiple frames within a short time period. This was made to ensure that no images from the same track would be represented in both the training and testing set, to avoid a biased testing set. This resulted in a skewed representation of some signs. The distribution of classes in the datasets is represented in appendix A.

Table 3.1: *Description of the datasets used for training and testing in the project.*

Dataset	Description	Type	Use	Signs	Images
LISA	LISA TS + LISA Extension (256x256)	Day	-	10503	9924
LISA test	LISA test split (20%)	Day	Test	2161	2105
LISA training	LISA training split (80%)	Day	Train	8342	7819
BDDNex	BDD + Nexar (256x256)	Night	Test	2248	1992

3.1.1 Preprocessing of Datasets

Training the GAN required significant computing power to process large images. The size of the LISA dataset varied from 640x480 to 1280x960 pixels. In order to process the images faster they were cropped to 256x256 pixels as seen in Figure 3.1 while retaining the traffic signs. The cropping was made in the area of the image that contained the most signs (bounding boxes). In addition, random movement

was applied to the x- and y-coordinates of the crop to prevent signs from being centered in the middle of the crop. This was done to ensure that the training data was unbiased in regards to the locality of the signs in the image. This reduced the problem size while images still were subject to localization and classification. Every method of data augmentation utilized this cropped version of LISA.

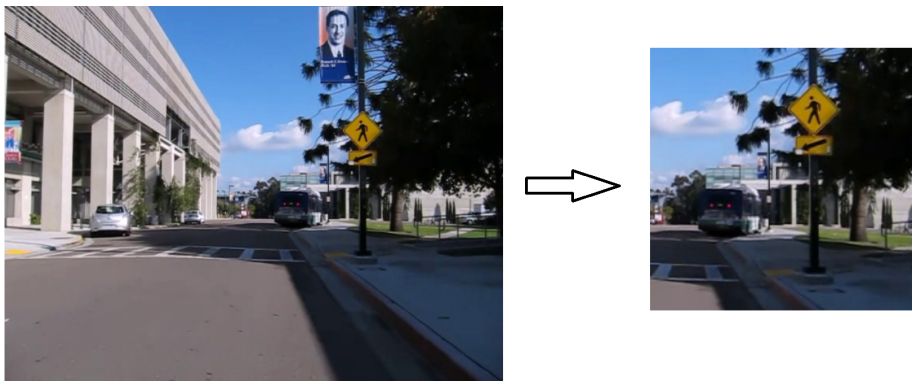


Figure 3.1: *A full size image of the LISA dataset cropped to 256x256 pixels.*

3.1.2 YOLO Training Data

How the augmented datasets created with SimpleAugment, Blender and GAN were created is illustrated in Figure 3.2. The training sets consists of twice the amount of images as the LISA training dataset with the addition of the augmented images. YOLO was retrained on the datasets augmented with Blender, SimpleAugment, and GAN separately and tested on the same BDD, Nexar and LISA dataset in order to compare the performance.

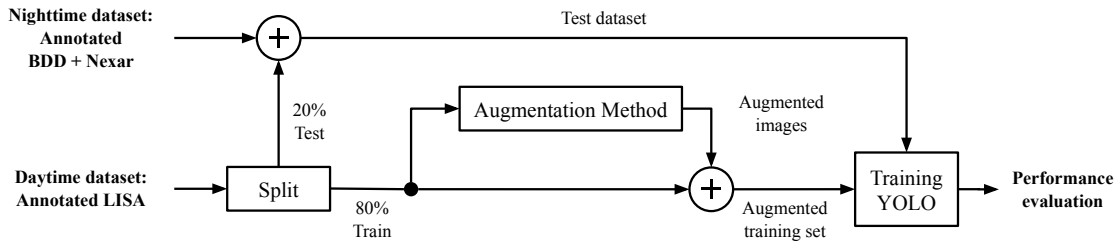


Figure 3.2: *Schematic illustration of iterative process in improving the training data to improve accuracy.*

3.1.3 GAN Training Data

A nighttime dataset was acquired for the training of the GAN to generate nighttime images. For this dataset, both Nexar and the BDD dataset were used. These datasets contained both night and daytime images in car traffic scenes. To separate nighttime images from these sets, a Python script was created to resolve this automatically. The resulting dataset consisted of 9652 nighttime images.

3.1.4 Testing data

The LISA dataset do not contain any nighttime images. To ensure that there was not any biased results, an out of distribution night dataset was acquired. The night dataset used for testing were handpicked BDD + Nexar images containing the same traffic signs as in the test LISA dataset. These images were manually annotated. This resulting dataset of 1992 images was used to test YOLO's performance on nighttime images. Each augmentation method was also tested on 2105 LISA images.

3.2 Bounding Box GAN

A customized GAN was developed to make a style transfer from daytime to nighttime while preserving the content in the bounding box of the images. The loss function consists of two parts, a BCE on the style transfer and a mean square error (MSE) on the content inside the bounding box. The discriminator was a CNN and the generator was using a Feed Forward architecture. Due to the nature of the loss function, the network given the name Bounding Box GAN (BBGAN). Images from the LISA dataset were mapped from daytime to nighttime using the BBGAN, of which two examples are shown in Figure 3.3.

Figure 3.3: *Cropped LISA image transformed to nighttime using BBGAN.*

3.2.1 Loss Function

The loss function consists of one loss for the content and another one for the style. Constructing a loss function that only considers what is inside the bounding box will minimize the loss of content in this area. This loss is defined as the mean square error between the real and generated image:

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n}(z_i - G(z_i))^2 \tag{3.1}$$

where z_i is the pixel values of the real image, $G(z_i)$ is the pixel values of the generated image and n is the number of pixels. The operations acting inside the bounding box area needs to preserve this content in order to minimize the loss. For each layer the

same kernel was used all over the image and its feature maps. Therefore the loss function inside the bounding box also improves the preservation of content in the whole image. In order to generate images in a night style one more loss was needed, the style loss. This loss is a normal binary cross entropy. The total loss of these two losses, with respective weights, is presented in Figure 3.4.

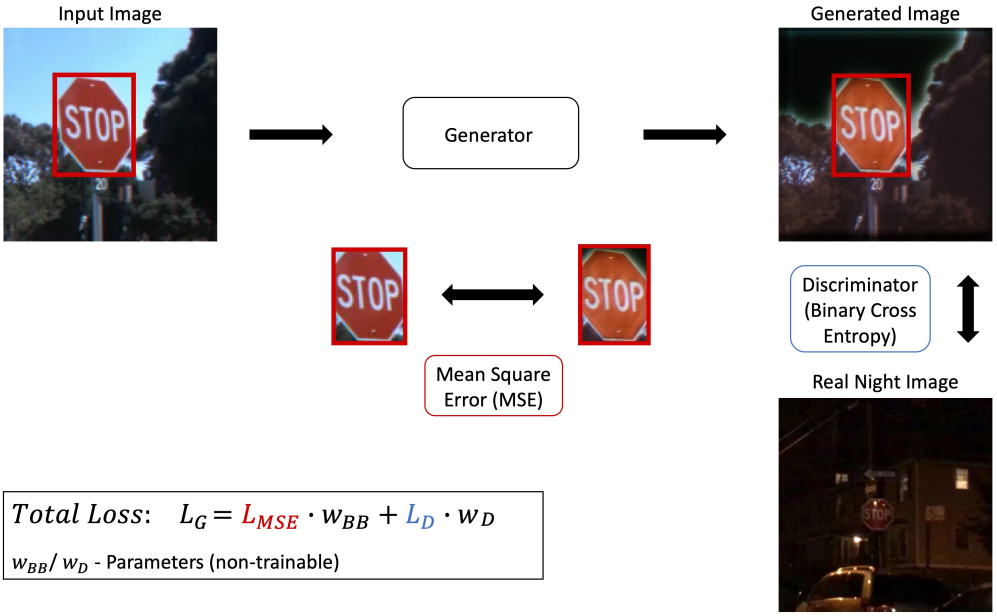


Figure 3.4: *Visual representation of the custom loss function*

3.2.2 Feed Forward Generator Model

The chosen generator model implemented is called Feed Forward which is inspired by U-net architectures [17]. This method improves the preservation of content in the image by adding feature maps from previous layers to newer ones. The networks capability of preserving content in road signs during the transformation from day to night was crucial to produce realistic nighttime images.

On the left side of Figure 3.5 the image is going through blocks containing convolutional layers (green). These blocks are decreasing the image resolution and increasing the feature maps. The feature maps are saved to be reused later. On the right side, the image is going through blocks containing deconvolutional layers (red). These blocks are increasing the image resolution and decreasing the feature maps. It is in these blocks the saved feature maps are added, as visualized by the purple arrows.

The reason behind adding old feature maps to the later ones is mitigate information loss. When the image flows through the network the feature maps are tweaked by different operations in order to change the day image to a night image. These operations can lead to content loss. The saved feature maps have fewer operations

affecting compared to later ones, thus still have a lot of information from the original image. Merging the old feature maps with the new ones are done by concatenation along the feature axis. How much these feature maps should affect the current image is determined by two weight parameters, α and β, which are multiplied with the feature maps. These two parameters are trainable within TensorFlow's computation graph and therefore were able to change during the training.

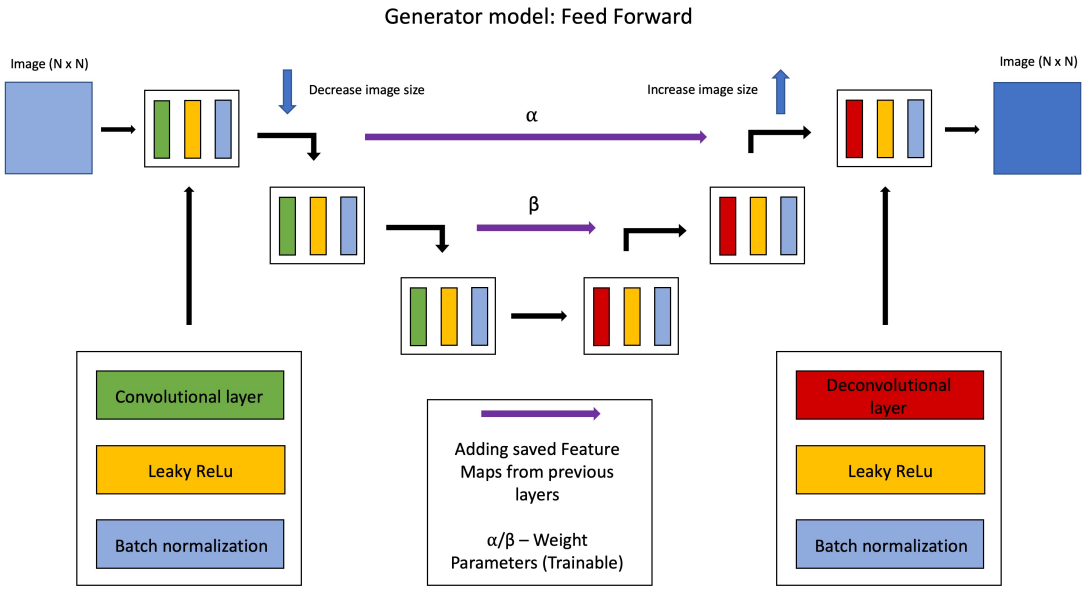


Figure 3.5: *Network structure for the Feed Forward generator model.*

3.2.3 Discriminator Model

The discriminator model chosen was a custom made CNN with an architecture as seen in Figure 3.6. An input image is first fed into a hidden layer without any batch normalization. All following layers consist of a convolutional 2D layer, a Leaky ReLU with $\alpha = 0.2$ and a batch normalization. Only the first hidden layer keeps the dimension and the other seven hidden layers bisect the image size. The input size of $256 \times 256 \times 3$ is downscaled throughout the network to the final size $2 \times 2 \times 1024$. Thereafter, three fully connected layers downsizes the 4096 nodes to 100 nodes before going to the final single node. It is the value of this final node that predicts the probability for fake or real. It is also the input of the loss function and the subject of the optimizer.

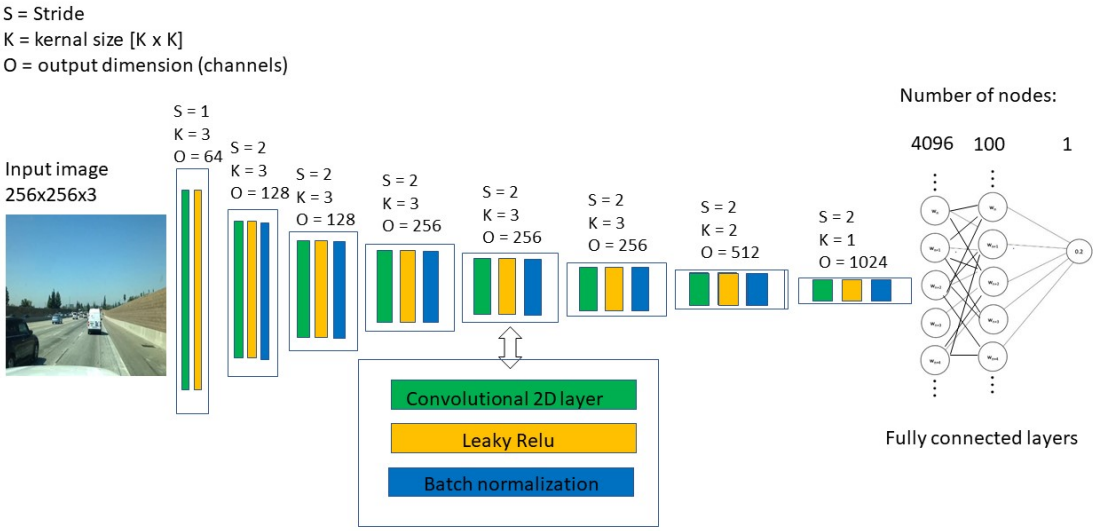


Figure 3.6: *Network structure for the customized GAN's discriminator*

3.3 Alternative Augmentation Methods

To evaluate the performance of the BBGAN, several other augmentation methods were implemented. Two machine-learning based methods, CycleGAN and AutoAugment, as well as two simpler methods not utilizing machine-learning, SimpleAugment and Blender. These methods were used to compare against the BBGAN implementation.

3.3.1 Blender

The use of 3D-modelling software like Blender [18] has previously been used to successfully implement automated pipelines for generation of annotated training data for classifiers [19]. Inspired by this, 3D-models of the geometries of traffic signs were first created using the software. Images of traffic signs and cameras panoramas of nighttime scenes were then collected. A script could then be used to randomly render traffic signs as in Figure 3.7 from various angles and backgrounds. The world space coordinates of the sign model were automatically transformed to screen space and used as annotations for each rendered image.

Figure 3.7: *Example of images generated by Blender.*

3.3.2 SimpleAugment

The Python script SimpleAugment augments images from LISA using simple operations. The images were augmented in three sections. The first increased the blue color values in each pixel as well as lowered the RGB values in the image depending on the initial RGB value. The pixels with higher values got decreased exponentially more than the pixels with initially lower values in order to create a darker version of the input image. The second was a further darkening of the pixels in the top half of the image to make the sky darker. The last alteration was attaching the traffic sign from the input image on the exact same location using the coordinates from the original bounding box but in a lighter tone than the rest of the image, in order to highlight the traffic sign as illustrated in Figure 3.8.

Figure 3.8: *Example of images generated by SimpleAugment.*

3.3.3 CycleGAN

One approach at augmenting images with GANs was to use an already existing GAN implementation. CycleGAN is a type of GAN that takes images as inputs and outputs the same images with a different style. The key difference from traditional GANs is that it preserves the content of the data it is fed instead of creating new content from noise. The goal was to optimize its weights and biases such that mapping from the daytime domain to the nighttime domain and back again yields the same image. This should also be true for the reverse mapping [7].

CycleGAN comes with the option of choosing different generative models, either a

Residual network or a U-net. U-net generates different sections of the image at a time and therefore does not get full context of the image, potentially preventing it from learning certain features. Instead, a Residual network was used which generates a whole image at a time with the cost of higher VRAM usage. A day to night transformation from CycleGAN trained with the GAN training dataset and the cropped LISA dataset is exemplified in Figure 3.9.

Figure 3.9: *Cropped LISA image transformed to nighttime using CycleGAN.*

CycleGAN was also implemented with insertions of traffic signs from the daytime dataset using the coordinates from the original bounding box. Creating a similar dataset but with brighter signs as can be seen in Figure 3.10.

Figure 3.10: *Cropped LISA image transformed to nighttime using CycleGAN with insertions.*

3.3.4 AutoAugment

AutoAugment is a machine-learning method with 20 different policies for augmentation. Each policy applies two transformations to the bounding box area of the image. In Figure 3.11 examples of augmentations created by using AutoAugment are illustrated where the policies are shearing, HSV-transformation and adding gray color as occlusion.

Figure 3.11: *Examples of possible transformations by AutoAugment*

AutoAugment and BBGAN were two standalone augmentation methods, but were also combined in a pipeline as a third augmentation method. In this case, the BBGAN dataset was compiled and then further augmented by AutoAugment. This method is exemplified in Figure 3.12.

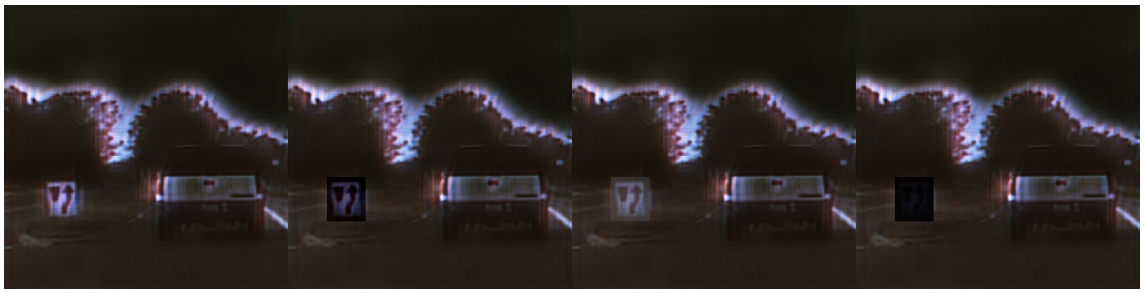


Figure 3.12: *Examples of augmenting the BBGAN dataset with AutoAugment.*

3.4 Training Networks

The training of the GANs and YOLO was done using a computation instance on the Google Cloud Platform [20]. The instances used either an Nvidia Tesla P100 GPU or a K80 as well as a quad core CPU for this task.

BBGAN was trained for approximately two hours during which it produced better and better nighttime images. BBGAN has a breakpoint at which it starts to generate less accurate night images. The training was therefore terminated before that point.

When training YOLO, 90% of the training dataset was used for training, and 10% was used for validation. After each epoch (all training examples) the model was tested on the validation set and the average loss was logged. These losses are visualized in Figure 3.13 to determine if the model was still learning, or if it had plateaued. At this point the model would not improve it's performance. When training YOLO, all layers but two were frozen until epoch 50. After this the rest of the layers were unlocked and their parameters could be changed during the training. It was evident that using augmented data allowed for the model to reach lower losses.

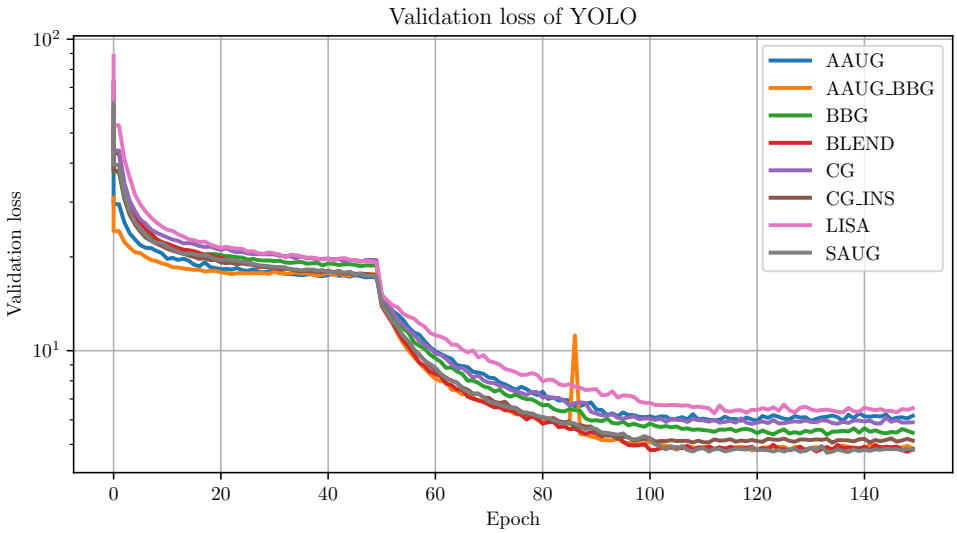


Figure 3.13: *Training losses of YOLO. Some point are outliers which are not representative of the average loss trend. This is the case around epoch 85.*

3.5 Evaluation

After training YOLO with augmented datasets, it was tested on the test set where both precision and recall were measured. Precision is the ratio of true positives (TP) and the total number of detected objects, i.e. true positives + false positives (FP). TP is registered when YOLO detects a sign correctly. FP is when YOLO makes an incorrect detection. Precision is therefore defined as:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{3.2}$$

False negative (FN) is when YOLO completely misses a sign. Recall is the ratio of TP and total number of detectable objects, i.e. TP + FN defined as:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{3.3}$$

The results gathered were the performance from one single YOLO instance of each augmentation method. Re-training YOLO for each method was not an option due to limited time and computation power. One YOLO instance took around 48 hours to train. To acquire a robust measure of performance, the test set was re-sampled 1000 times with the Bootstrap method. The method gives a mean-average performance metric of the augmentation methods without retraining YOLO.

In order to determine why the object detection made mistakes when detecting a traffic sign, the Intersection of Union (IoU) was evaluated. The IoU quantifies how well the detected area overlaps with the annotated area in the bounding box.

4. Results

The results of each augmentation methods dataset is presented using abbreviated codes found in Table 4.1.

Table 4.1: *Codes used for each augmentation method used when training YOLO.*

Code	Meaning
LISA	Only the daytime LISA images
BLEND	Daytime LISA + nighttime images rendered in Blender
SAUG	Daytime LISA + nighttime images generated with SimpleAugment
CG	Daytime LISA + nighttime images generated with CycleGAN
CG_INS	Daytime LISA + CG with daytime cropped signs inserted
BBG	Daytime LISA + nighttime images generated BoundingBoxGAN
AAUG	Daytime LISA + daytime images generated using AutoAugment
AAUG_BBG	Daytime LISA + nighttime images generated using AAUG applied on BBG

The impact of the augmented training data from the various methods on the object detection performance of YOLO is evaluated using both the mean precision μ_p and recall μ_r. Bootstrapping is used to visualize the standard deviation σ_p and σ_r with dataset size N, sample size n and number of resamples B. These results are presented in Table 4.2.

Table 4.2: *The precision and recall metrics of the various augmentation models.*

Model	Test	μ_p	σ_p	μ_r	σ_r	N	n	B
LISA	Day	0.897	0.007	0.883	0.007	2105	2105	1000
LISA	Night	0.700	0.010	0.662	0.010	1992	1992	1000
LISA	All	0.799	0.006	0.770	0.007	4097	4097	1000
BLEND	Day	0.898	0.007	0.891	0.007	2105	2105	1000
BLEND	Night	0.788	0.009	0.768	0.009	1992	1992	1000
BLEND	All	0.842	0.006	0.828	0.006	4097	4097	1000
SAUG	Day	0.903	0.007	0.887	0.007	2105	2105	1000
SAUG	Night	0.756	0.009	0.708	0.010	1992	1992	1000
SAUG	All	0.830	0.006	0.795	0.006	4097	4097	1000
CG	Day	0.890	0.007	0.881	0.007	2105	2105	1000
CG	Night	0.717	0.010	0.664	0.010	1992	1992	1000
CG	All	0.805	0.006	0.770	0.007	4097	4097	1000
CG_INS	Day	0.910	0.006	0.893	0.007	2105	2105	1000
CG_INS	Night	0.716	0.009	0.712	0.010	1992	1992	1000
CG_INS	All	0.811	0.006	0.800	0.007	4097	4097	1000
BBG	Day	0.907	0.007	0.895	0.007	2105	2105	1000
BBG	Night	0.761	0.010	0.677	0.010	1992	1992	1000
BBG	All	0.836	0.006	0.783	0.007	4097	4097	1000
AAUG	Day	0.906	0.006	0.903	0.006	2105	2105	1000
AAUG	Night	0.786	0.009	0.692	0.010	1992	1992	1000
AAUG	All	0.848	0.006	0.795	0.006	4097	4097	1000
AAUG_BBG	Day	0.916	0.006	0.913	0.006	2105	2105	1000
AAUG_BBG	Night	0.832	0.008	0.707	0.010	1992	1992	1000
AAUG_BBG	All	0.877	0.005	0.808	0.006	4097	4097	1000

The metrics obtained from the bootstrap are visualized in the box plots as seen in Figures 4.1 and 4.2. From these diagrams and the previous table it is evident that all methods seem to yield the same, or better precision and recall metrics compared to the baseline LISA.

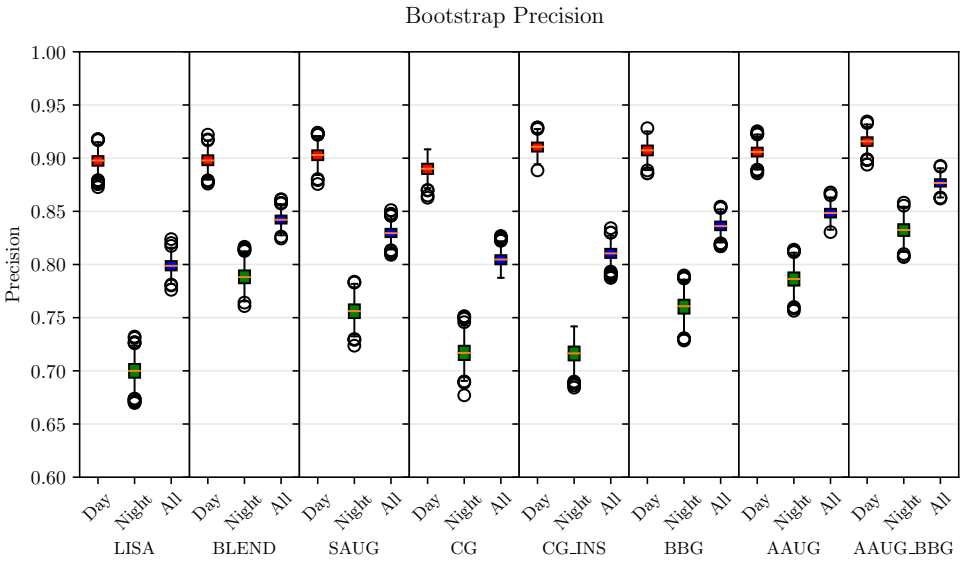


Figure 4.1: *YOLO precision after training using the various models.*

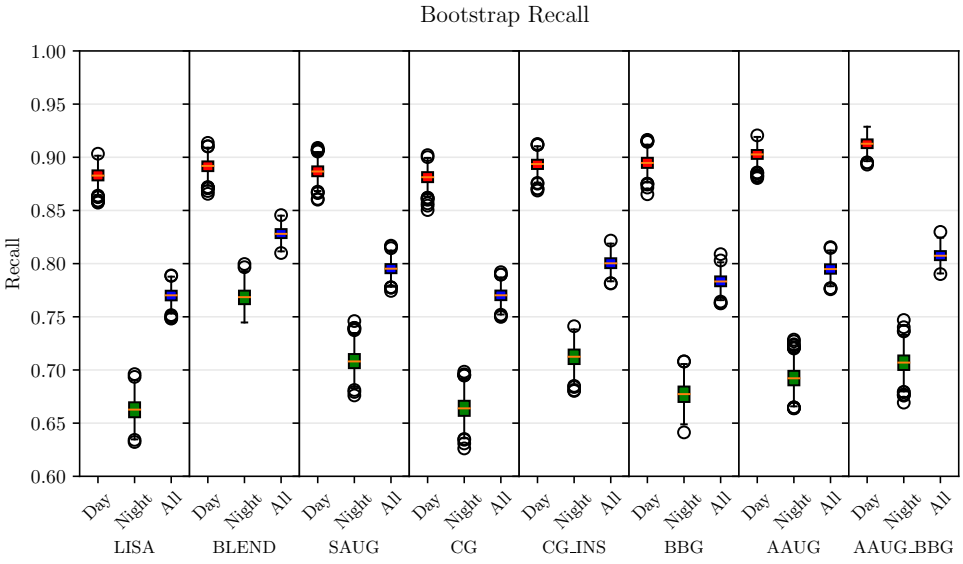


Figure 4.2: *YOLO recall after training using the various models.*

In addition, the object detection performance of the various models is visualized in Figure 4.3. Here the detection rate (number of detected signs) of each instance of the different classes found in the test dataset is presented for each augmentation method (markers). The different sign classes on the y-axis are sorted by total number of occurrences (number in the parentheses) of each class in the test dataset. The

classes which seem to cause most spread in the detection rate are the *noLeftTurn* and *speedLimit25* signs. The detection ratio (detection compared to the total in the class) of each class is shown in Figure 4.4.

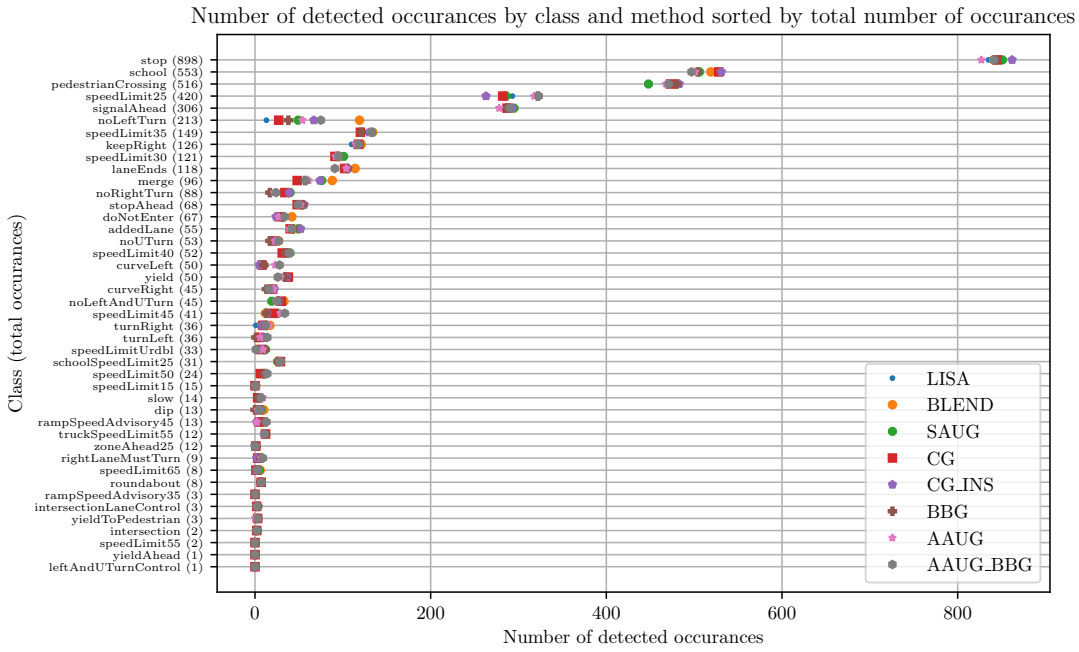


Figure 4.3: *Detection rate of the various models by each class found in the test dataset sorted by total number of occurrences of each class in ascending order.*

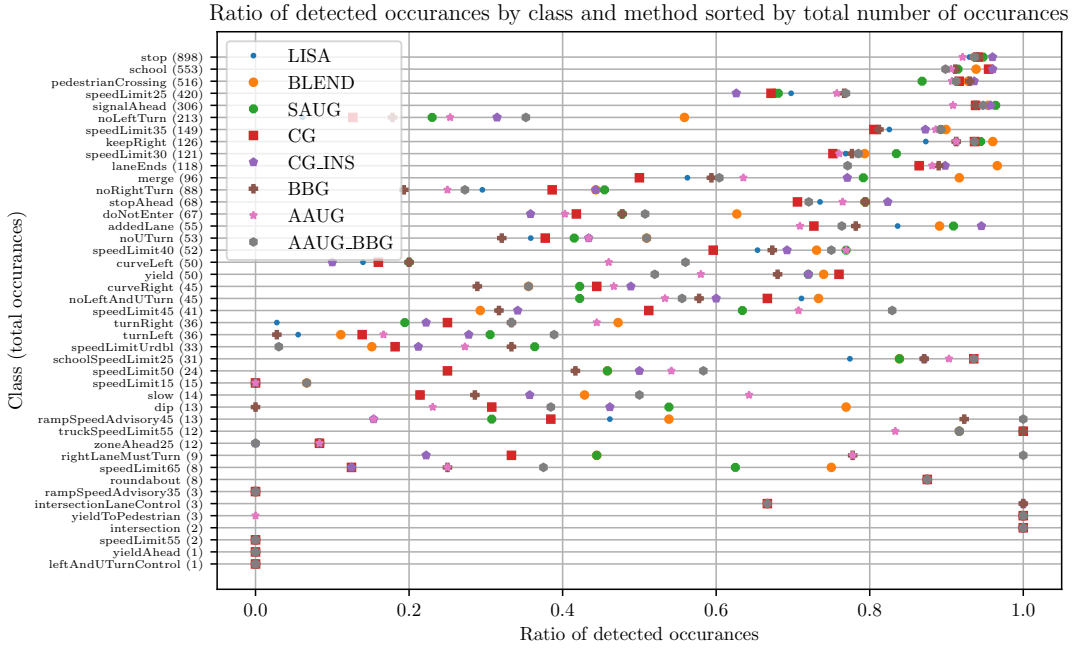


Figure 4.4: *Detection ratio of the various models by each class found in the test dataset sorted by total number of occurrences of each class in ascending order.*

The mistakes made by YOLO on each dataset is also presented in Figure 4.5. The different kind of mistakes are presented on the y-axis and the ratio of these mistakes are the x-axis. No IoU is when the intersection of union is zero. Low IoU is when its lower than 0.5. Wrong class is when YOLO lables a sign incorrectly. Low confidence is when YOLOs classification of a label is correct but the confidence is lower than 0.4. The ratio of the types of mistakes by the various methods is evidently not the same during daytime compared to nighttime. The mistakes mainly consists of having no IoU and classifying the wrong class. A method that stand out is AAUG with a high ratio of no IoU (52%/63%) and a low ratio of wrong class (28%/25%). Another one is BLEND with No IoU (41%/52%) and Wrong class (44%/34%) for day/nighttime ratio.

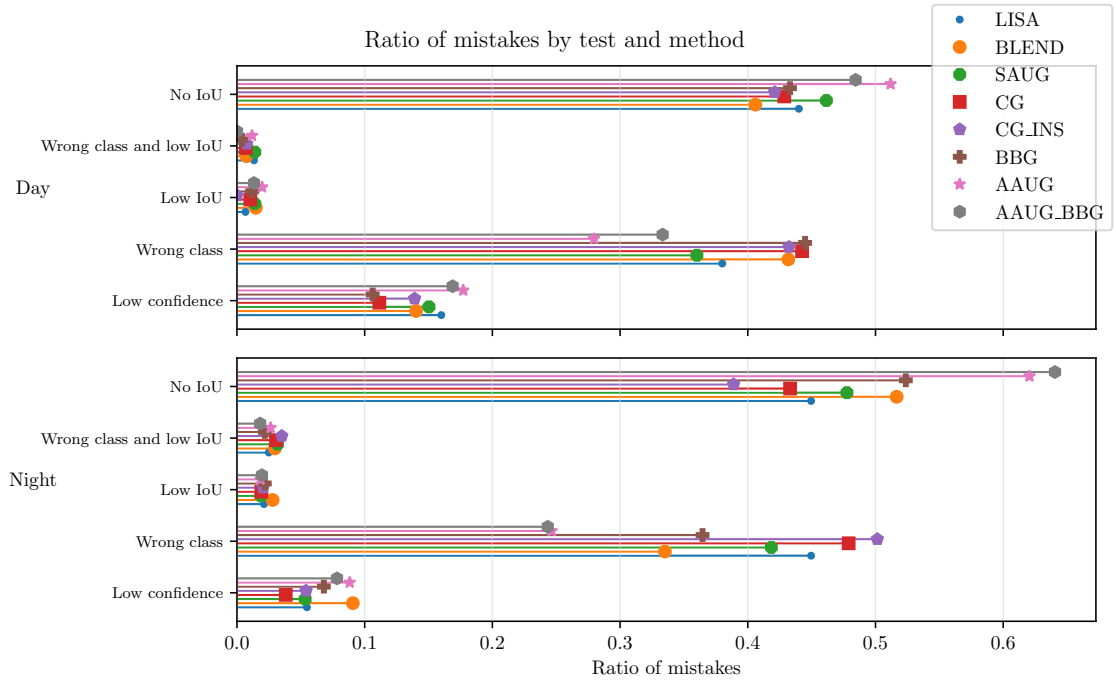


Figure 4.5: *Ratio of mistakes of YOLO by each augmentation method and test dataset.*

5. Discussion

From the results it can be seen that the baseline model trained using only LISA produced an average recall rate of 0.883 for day and 0.662 for night. This discrepancy is expected and supposed to be solved using augmentation of the training data.

All methods seem to retain daytime recall rate with the exception of AAUG and AAUG_BBG where it has improved to around 0.905. Common for these methods is the use of AutoAugment to apply policies to the sign's bounding box. While this does not necessarily provide examples of dark signs, it allows YOLO to further generalize its identification of well lit signs. In Figure 4.1 it can be seen that nighttime precision of AAUG and AAUG_BBG is high, around 0.82, in relation to the recall rate. This can be explained by looking at the mistakes of each method in Figure 4.5. Here it is evident that YOLO fails to detect many signs during nighttime. With both AAUG and AAUG_BBG the *No IoU* mistakes increases and *Wrong class* decreases compared to LISA (baseline). This leads to fewer false positives, more false negatives and therefore "improved" precision over the baseline.

In regards to the recall rate, the method which stands out the most with a nighttime recall of 0.768 is BLEND. This method uses high-quality image textures for each sign as opposed to low-quality real-world examples used in the other methods. This allows YOLO to learn intricate details of each sign and distinguish similar signs like *noLeftTurn* and *noLeftAndUTurn*. This specific example can be seen in Figure 4.3 where BLEND manages to identify over 100 more examples of the sign compared to LISA. While the results are promising, it is important to remember that this method requires the most manual labor to set up and will not necessarily generalize well for other types of augmentation.

CG does not seem to improve the recall metric. This is due to the fact that the content of the traffic signs in the CG dataset are too dark to read in many cases. CG produces low quality images because it relies on the ability to map back and forth between the daytime and nighttime domains. The problem is that there is poor visibility in nighttime images which in turn makes a night to day transformation difficult.

SAUG and CGAN_INS both increased the performance on nighttime environments even though the bounding box area of was identical to the LISA images. Therefore it seems the surrounding environment around the bounding box make YOLO generalize better in regards to performance. It also indicates that the ODS has learned features other than those of the signs in the image.

BBG provides a modest nighttime recall rate of 0.677 which is barely an improvement over LISA. While this is not the result that was desired with the GAN there are

some merits with this approach. It is easy to train the GAN on any kind of style like weather or time of day given image examples of these settings, compared to the manual labor required by BLEND and SAUG. Given more time and computation power, the performance of the GAN could be further improved with changes to the generator and discriminator architecture. The computation power limits the number of layers and complexity of the GAN. In addition, training the BBGAN takes 1-2 hours on a Google Cloud computation instance with an Nvidia Tesla P100 GPU before it produces images that start looking like nighttime. After this, the signs in the images starts to get non-recognizable. This could potentially be improved upon by changing the architecture and using brighter nighttime images as training data.

5.1 Quality of data

The LISA dataset contains both colored and grayscale images. One ODS could learn to identify a red stop sign mostly from the characteristics of the color in the sign. A stop sign without any color will then cause the ODS to learn that the shape of the sign is more important than the color. This confusing data could affect the network in a way that makes it less robust to real world usage. It is however important for an ODS to learn to detect objects in different ways and not rely too much on color since many traffic signs share the same color.

It is important to take into account the distribution of classes in both the training and testing datasets. The absence of better datasets resulted in the test set not being a good representation of the training set in terms of the distribution of classes. Some traffic signs were over-represented in the training or test data as can be seen in appendix A. While effort has been made to match the distribution of signs in both datasets, there are examples of classes where there are imbalances in the distribution. This introduces an uncertainty factor that makes it more difficult to make conclusions from the presented results.

The data used to train the GAN is a direct representation of the target domain for the generated images. If the BDD/Nexar training dataset had better quality in terms of visibility and clarity the results could potentially be improved. Other projects prove that it is possible to generate lifelike images if enough resources are put into development, for example StyleGAN [21].

5.2 Multiple Domains

This project focused on a single domain transformation, daytime to nighttime. However, to develop a complete solution for use in autonomous driving, the GAN must be able to transform into other domains as well as combinations of these domains. A real-world example could be mapping daytime images to snowy conditions during nighttime. Several multi-domain image transformation GANs already exists, e.g. StarGAN [22]. If these GAN solutions were combined with the loss function introduced in this report, a sufficient augmentation pipeline could be developed.

6. Conclusion

Conventional methods of data augmentation, like color manipulation in SimpleAugment and 3D-renders in Blender, offers measurable increase in performance when learning out of distribution features. Given enough manual labor in the form of analysis in the target domain of features, these methods can be implemented. However, due to the required manual labor these methods will have problems scaling up and generalizing as the target domain of augmentation grows. In order to create an end-to-end augmenter capable of handling various lighting conditions, weather, time of year, different locales and more, there is a need for a more robust solution.

To train an ODS like YOLO it is important to preserve identifiable features of the detectable object when performing augmentations. In the case of day-to-night transformations it is important not to lose any content by making the traffic sign too dark. It can be achieved using already annotated data, making it possible to perform transformations outside the objects bounding box in order to preserve its features. At the same time it needs to approximate the target domain of nighttime scenes.

One scalable solution is a variant of an image-to-image GAN with a novel feature-specific loss utilizing the MSE of the bounding box contents in annotated images. While this method provides only marginal improvements to nighttime object detection recall of the baseline, it shows some promise. Specifically when combined with other augmentation methods in an "end-to-end" augmentation pipeline like AutoAugment.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Departement d'informatique et de recherche opérationnelle Université de Montréal*, 2014. [Online]. Available: https://arxiv.org/abs/1406.2661.

[2] N. H. T. S. Administration. (2017). Automated vehicles for safety, [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[3] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *University of Washington*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.02767.

[4] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund, "Vision based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Transactions on Intelligent Transportation Systems*, 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6335478.

[5] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv e-prints*, arXiv:1511.06434, arXiv:1511.06434, Nov. 2015. arXiv: 1511.06434 [cs.LG].

[6] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation policies from data," *Google Brain*, Oct. 2018. [Online]. Available: https://arxiv.org/abs/1805.09501.

[7] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *CoRR*, vol. abs/1703.10593, 2017. arXiv: 1703.10593. [Online]. Available: http://arxiv.org/abs/1703.10593.

[8] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017. arXiv: 1701.00160. [Online]. Available: http://arxiv.org/abs/1701.00160.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[10] DanB. (2018). Rectified linear units (relu) in deep learning, [Online]. Available: https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning/log.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," Mar. 2015. [Online]. Available: https://arxiv.org/pdf/1502.03167v3.pdf.

[12] A. Agrawal. (2017). Loss functions and optimization algorithms. demystified., [Online]. Available: https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c.

[13] Skymind, *A beginner's guide to generative adversarial networks (gans)*. [Online]. Available: https://skymind.ai/wiki/generative-adversarial-network-gan (visited on 02/05/2019).

[14] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," Jul. 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[15] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "BDD100K: A diverse driving video database with scalable annotation tooling," *CoRR*, vol. abs/1805.04687, 2018. arXiv: 1805.04687. [Online]. Available: http://arxiv.org/abs/1805.04687.

[16] Nexar. (2017). Nexar challenge ii, [Online]. Available: https://www.getnexar.com/challenge-2/.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. arXiv: 1505.04597. [Online]. Available: http://arxiv.org/abs/1505.04597.

[18] blender.org, *Home of the blender project - free and open 3d creation software*, [Online; accessed 14-February-2019]. [Online]. Available: https://www.blender.org/.

[19] M. Goyal, P. Rajpura, H. Bojinov, and R. Hegde, "Dataset augmentation with synthetic images improves semantic segmentation," *arXiv e-prints*, Jun. 2018. [Online]. Available: https://arxiv.org/pdf/1709.00849.pdf.

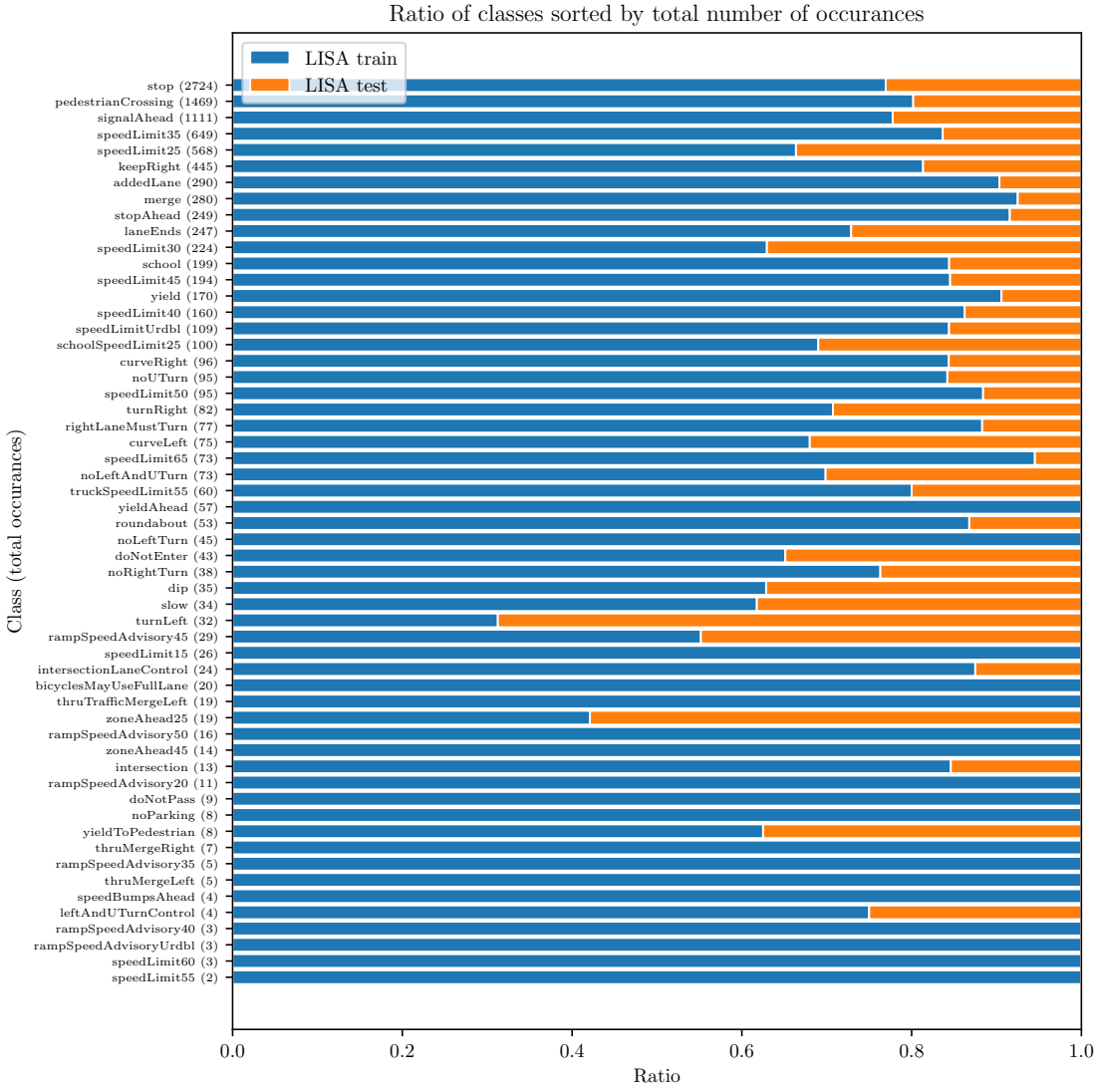
[20] *Google cloud platform*. [Online]. Available: https://cloud.google.com.

[21] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *CoRR*, vol. abs/1812.04948, 2018. arXiv: 1812.04948. [Online]. Available: http://arxiv.org/abs/1812.04948.

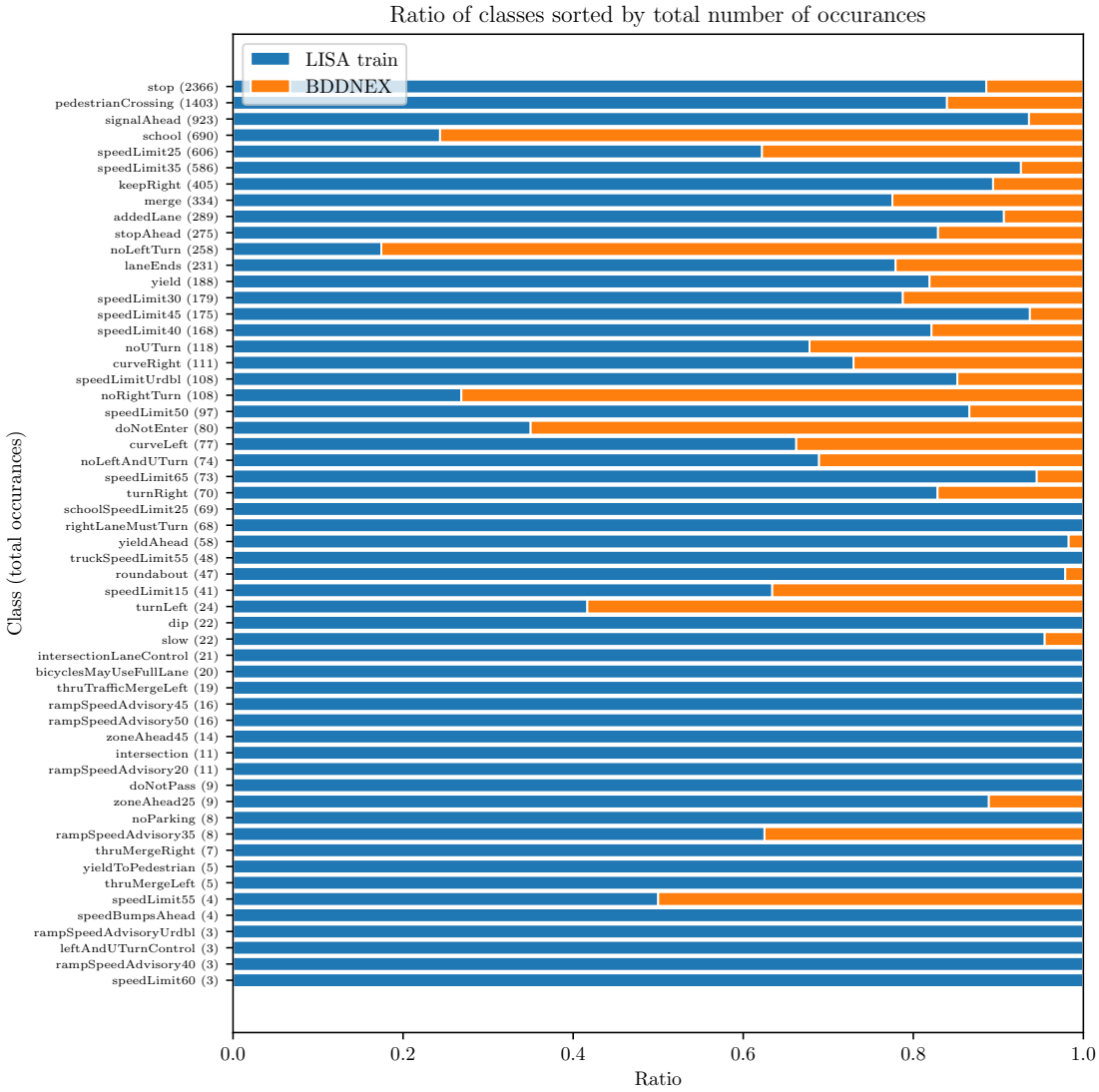
[22] Y. Choi, M. Choi, M. Kim, J. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," *CoRR*, vol. abs/1711.09020, 2017. arXiv: 1711.09020. [Online]. Available: http://arxiv.org/abs/1711.09020.

A. Dataset distributions

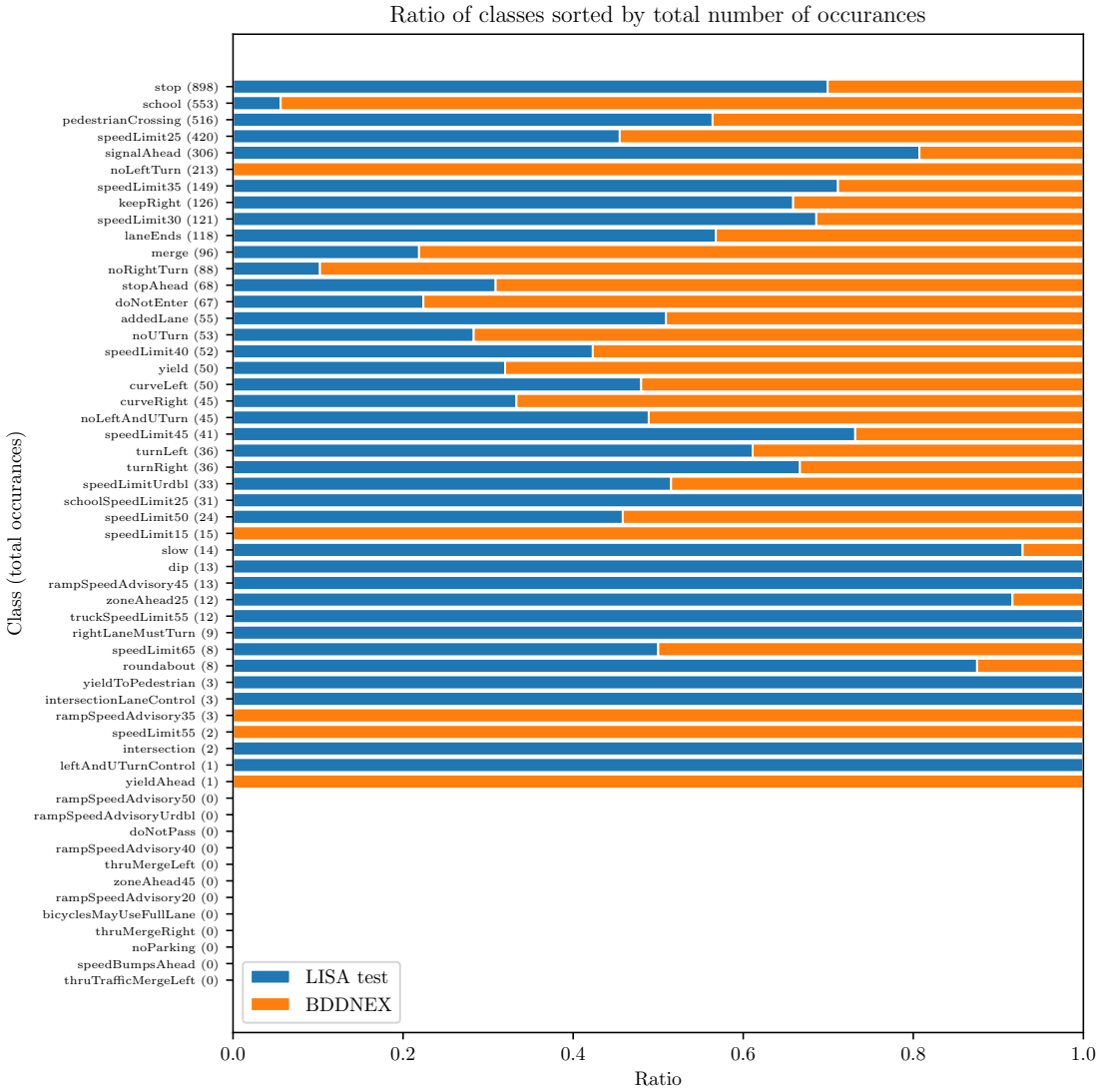
A.1 LISA training and test splits



A.2 LISA training and BDDNEX

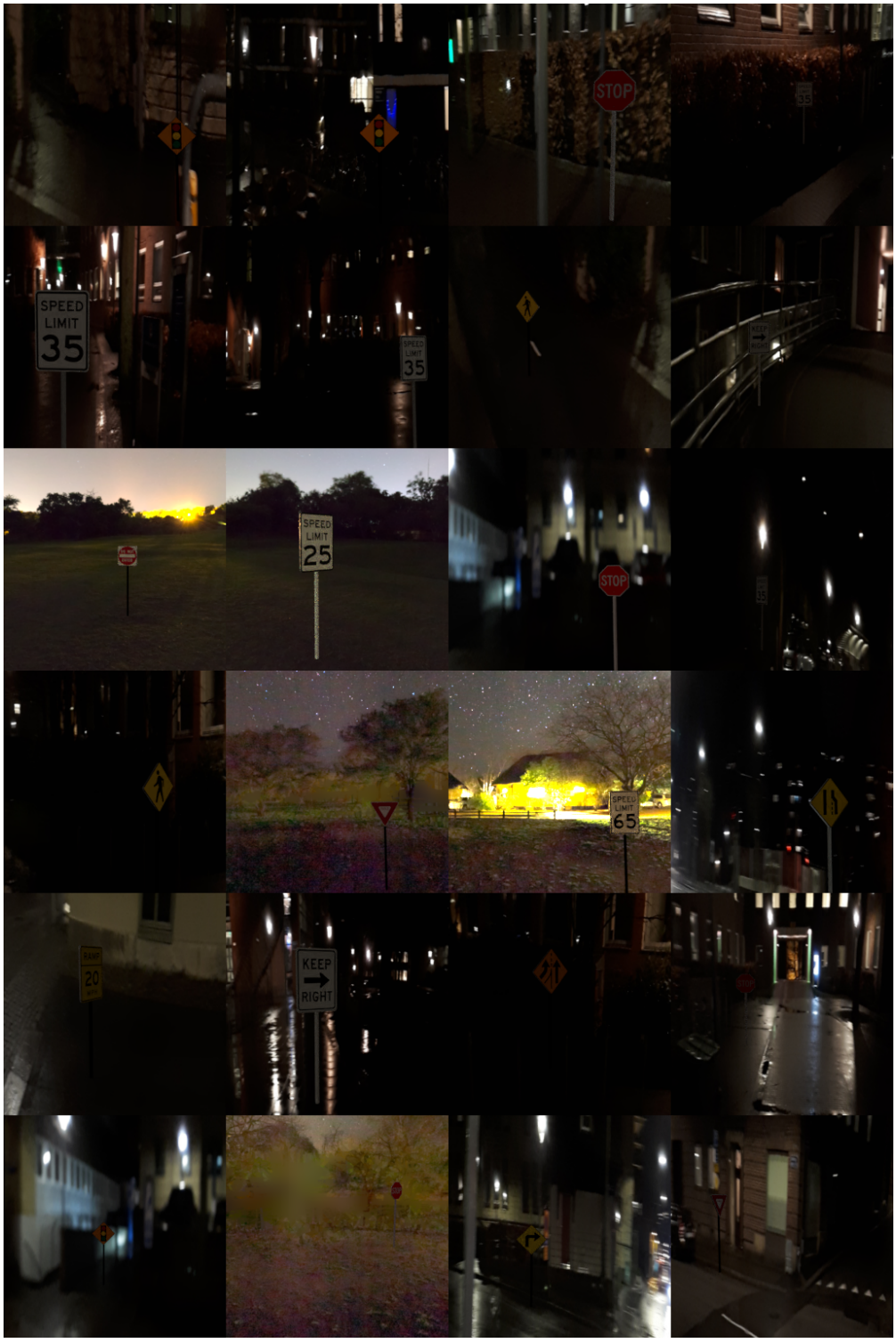


A.3 LISA test and BDDNEX

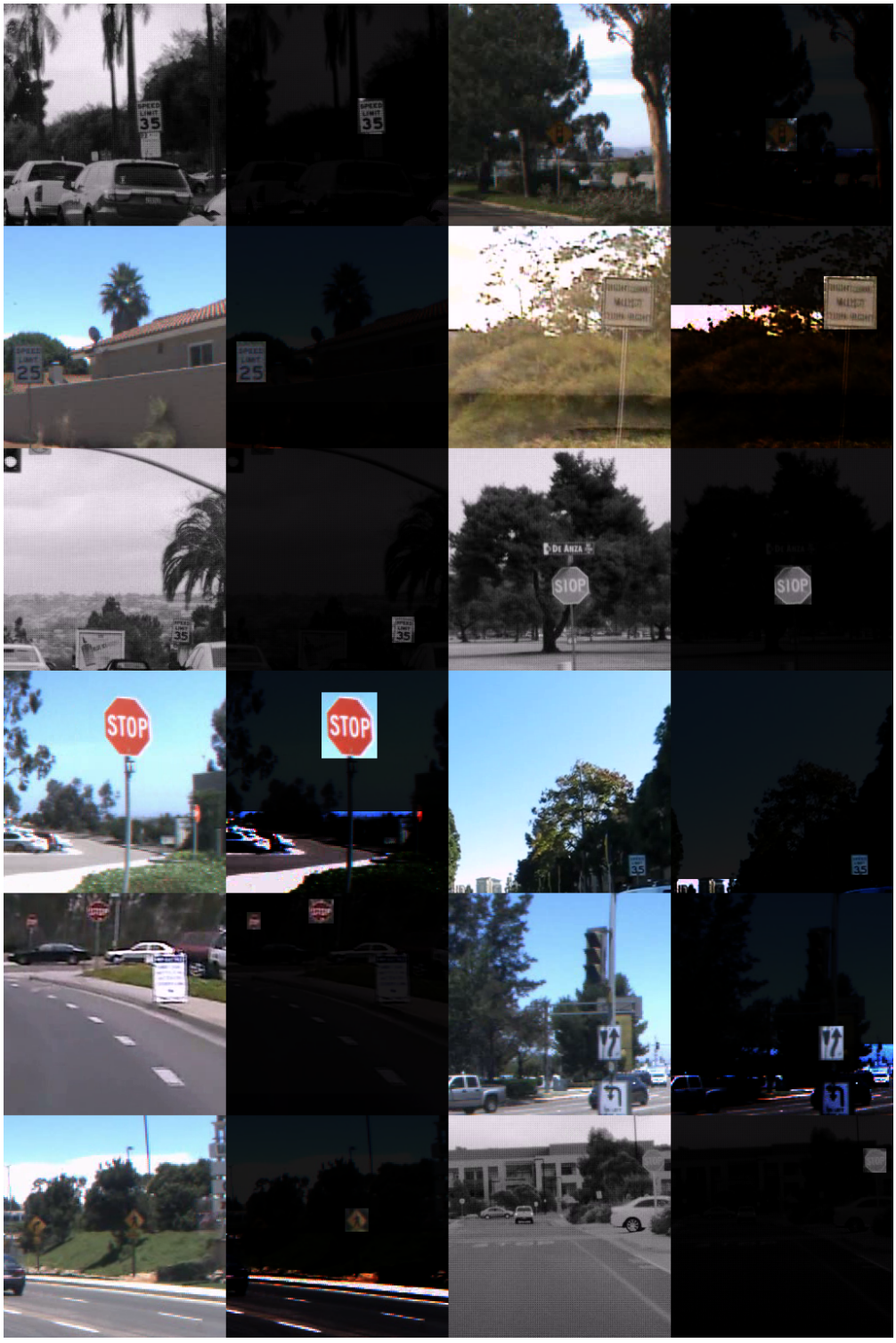


B. Generated image examples

B.1 Blender

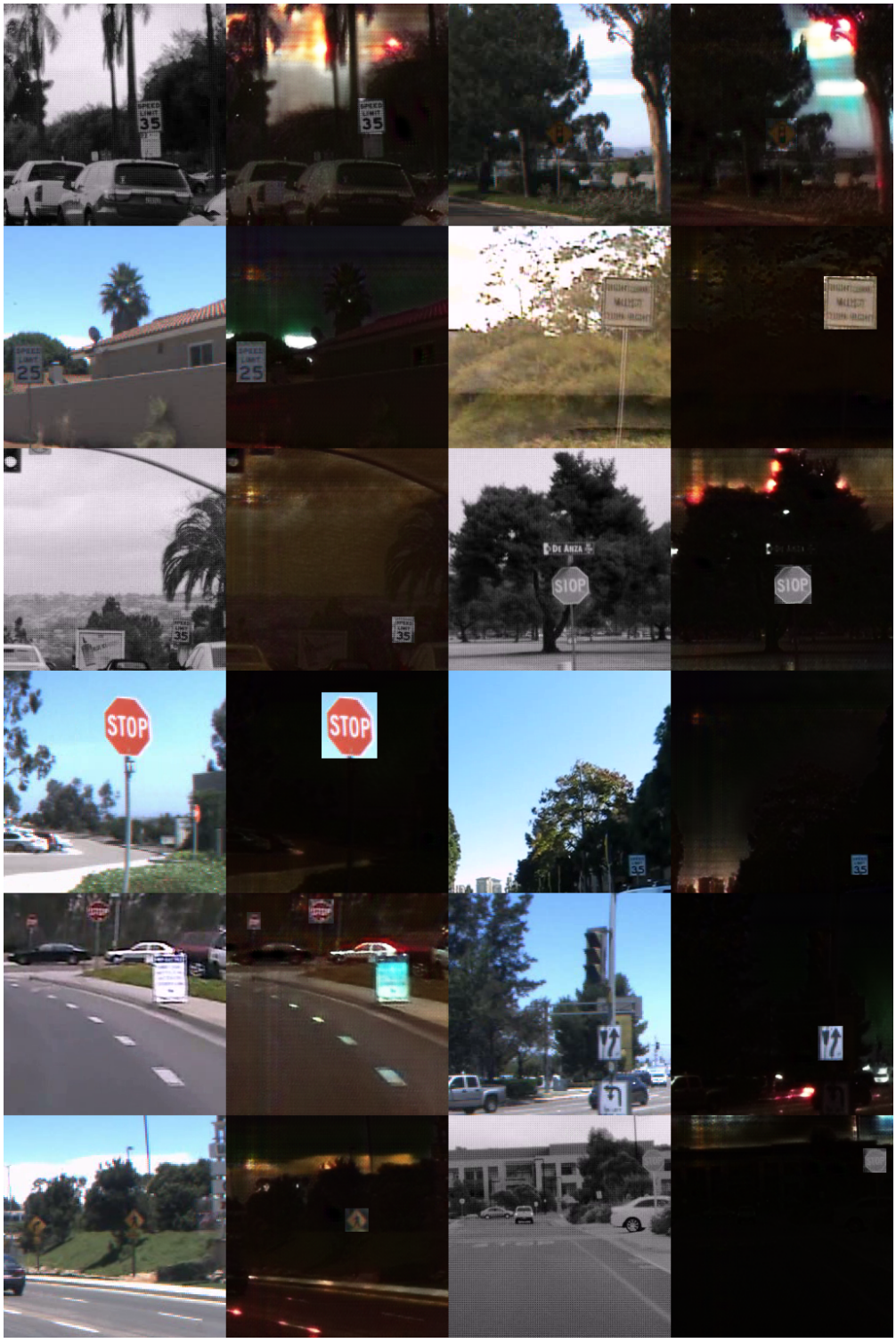


B.2 SimpleAugment



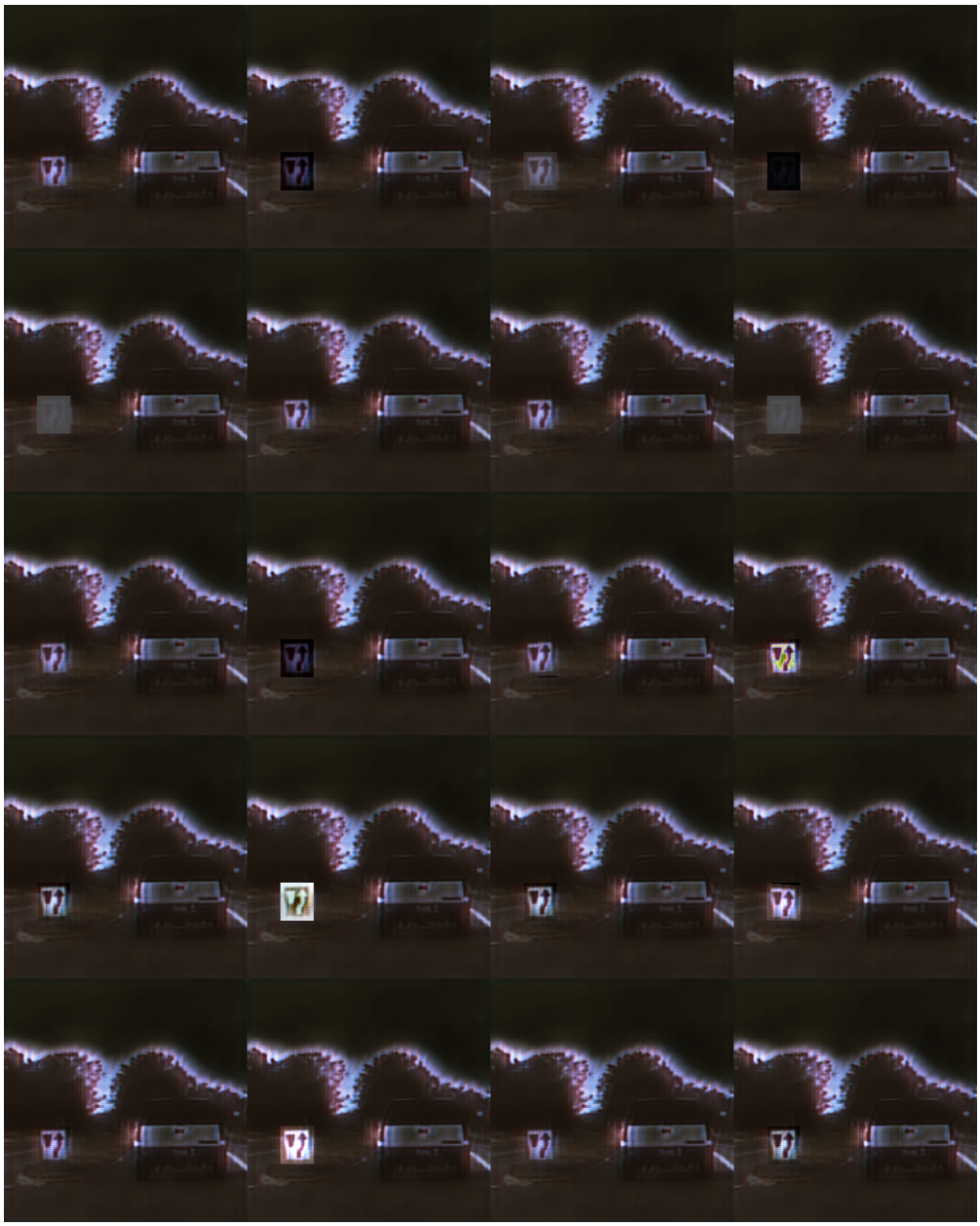
B.3 CycleGAN

B.4 CycleGAN with insertions



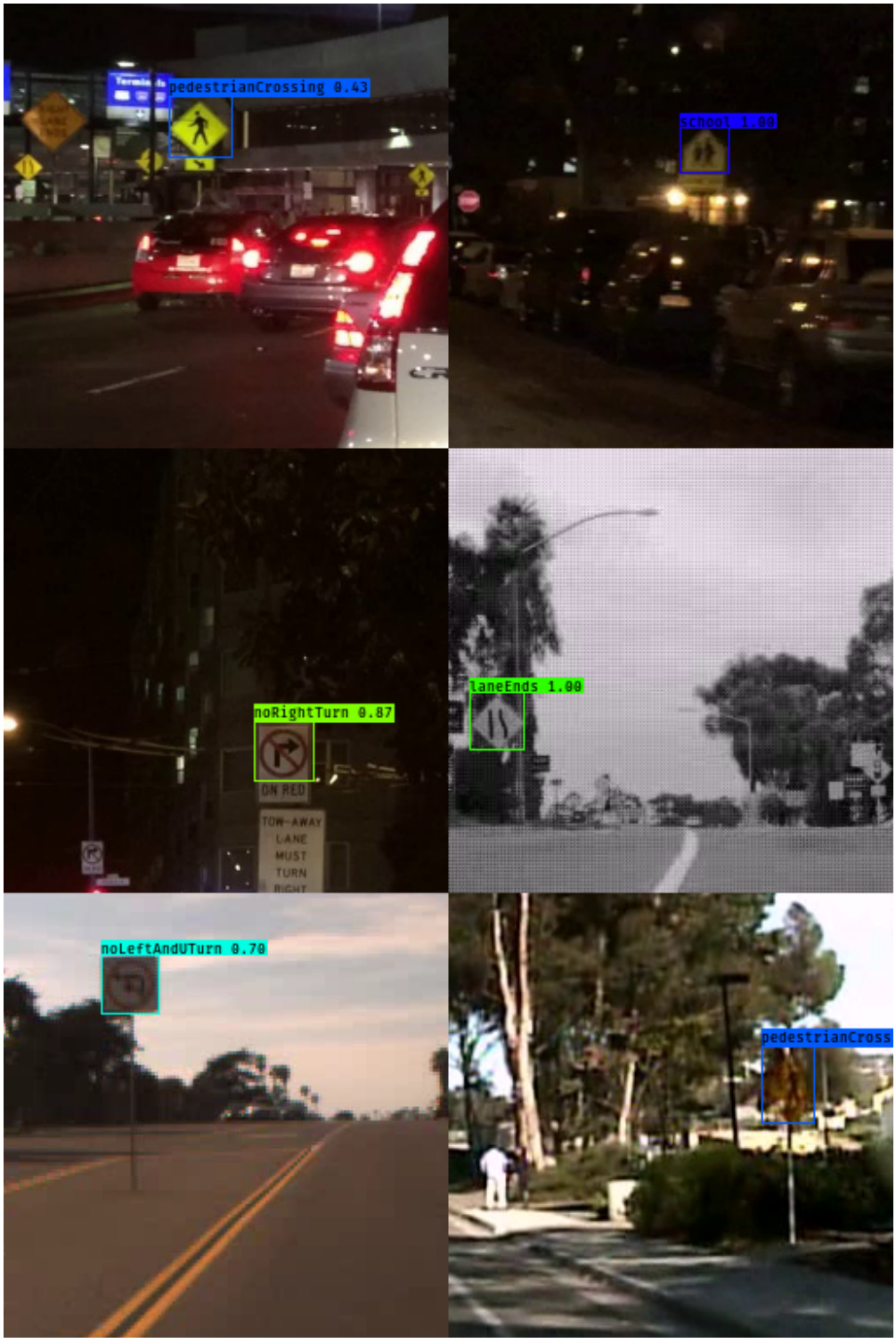
B.5 BBGAN

B.6 AutoAugment with BBGAN



C. YOLO detection examples

C.1 Correct detections



C.2 Failed detections