# Machine Learning Engineer Nanodegree

## Capstone Project

Rico Meinl

May 2, 2018

## Contents

# 1 Definition

## 1.1 Project Overview

This project is classied as a Computer Vision problem and the task is to build an algorithm that can recognize traffic lights.
I worked with the LISA Trac Light Dataset [1] and for the task I will be referencing on two papers [2][3] that discuss this issue. Traffic Light Detection is a crucial element of driver assistance systems (DAS) used in autonomous vehicles to safely navigate on public roads. Fully autonomous driving is a scenario that lays in the future but building components for DAS may save plenty of lives on the way.
My personal motivation behind this is to start understanding what components are necessary to build autonomous vehicles and how we can continue to improve them. As the majority of accidents on the road stem from human error I believe it is a crucial task for humanity to invest in the development of self-driving cars. We could easily save hundreds of lives every day.
Finding a dataset that is challenging and provides variety is not an easy task. For this problem I worked with the LISA Trac Light Dataset [1] which I obtained from Kaggle.com [4]. It contains more than 44 minutes of annotated traffic light data.
The database consists of continuous test and training video sequences, totaling 43,007 frames and 113,888 annotated traffic lights. The sequences are captured by a stereo camera mounted on the roof of a vehicle driving under both night- and daytime with varying light and weather conditions.
Only the left camera view is used in this database, so the stereo feature is in the current state not used. It was collected in San Diego, California, USA.
The database provides four day-time and two night-time sequences primarily used for testing, providing 23 minutes and 25 seconds of driving in Pacic Beach and La Jolla, San Diego. The stereo image pairs are acquired using the Point Greys Bumblebee XB3 (BBX3-13S2C-60) which contains three lenses which capture images with a resolution of 1280 x 960, each with a Field of View(FoV) of 66 degrees. The left camera view is used for all the test sequences and training clips. The training clips consists of 13 daytime clips and 5 nighttime clips. The annotation classes are: go, go forward, go left, warning, warning left, stop, stop left.

## 1.2 Problem Statement

The paper [2] describes the problems and challenges that this task offers. So far basically all solutions have been based off local and small datasets which does not offer a general solution to our problem.
Finding a general solution is hard, because traffic lights are different in every part of the world. In [2] Figure 4 the author shows that even in different states in the USA the traffic lights have different positions and shapes.
We also face the issue of environmental influences which makes it hard for an algorithm to always capture the Traffic Lights shape, size and color respectively.

One problem might also be that our algorithm confuses the tail lights of the driver in front of our car with a red traffic light and abruptly stops causing an accident with vehicles behind us. For the Solution of this problem I choose to use the YOLO Algorithm, which provides fast and accurate state-of-the-art performance in Object Detection. It runs an input image through a Convolutional Neural Network and outputs an encoding where each cell contains information about 5 boxes (a box tries to capture an object).
We then filter these boxes to only keep those with the highest probability. As this algorithm is fairly non-trivial and a full explanation would go far beyond the dimensions of this proposal I attached the two main papers [5][6] for anyone who wants to delve deeper into the Math behind it.
In general I will take the pre-trained weights from the official YOLO Website [7] and fine-tune them for our Traffic Light dataset. After that our model should be able to recognize Traffic Lights on picture frames and annotate them. In contrary to some methods discussed in [2] I am choosing an end-to-end Deep Learning approach here. Other algorithms might include a pipeline such as Detection-Classification-Tracking but my approach will try to solve all of that in one go.

## 1.3 Metrics

The performance measures most commonly used on this problem are Accuracy, Precision and Recall. The definitions are shows in the following equations. TP, FP, FN, TN are abbreviations for true positives, false positives, false negatives and true negatives.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

A precision close to one indicates that all the recognized TL states are in fact correctly recognized.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

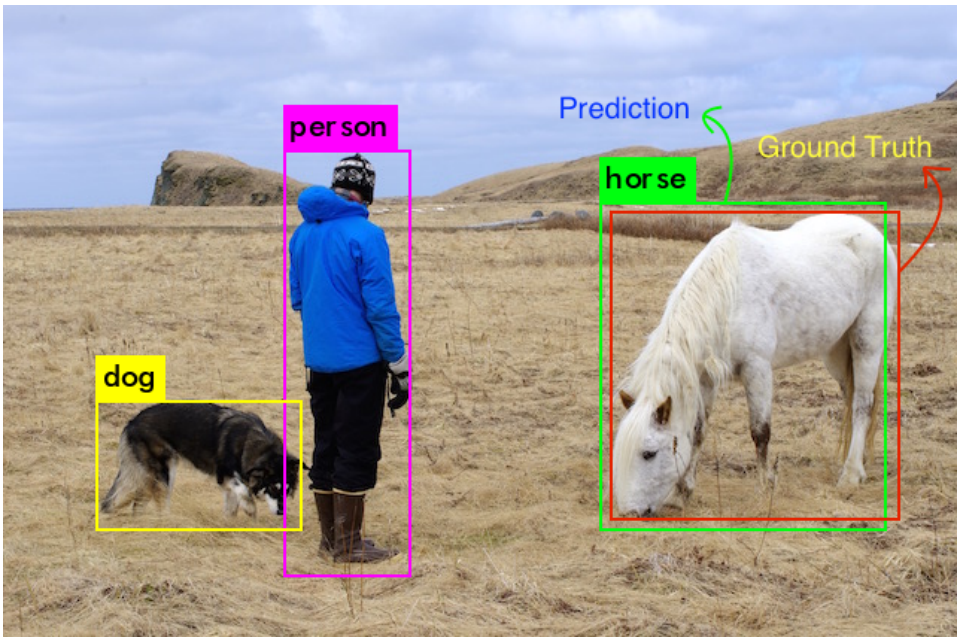
A recall close to one indicates that all the TL states, in a given video sequence, were correctly recognized by the proposed system.

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{3}$$

An accuracy close to one indicates that the system detects all TLs with no false positives.

With the YOLO algorithm I chose quite a different approach than so we have to redefine the metrics in our case. Since we are predicting not only the occurrence but also the position of the objects (Traffic Lights) we cannot apply the standard metric of precision in this case.
This is where mAP (Mean Average-Precision) comes into the picture.

We are given the actual image and the other annotations as text (bounding box coordinates (x, y, width and height) and the class. To quantify how our prediction compares to the ground truth we calculate the IoU (Intersection over Union), also known as the Jaccard Index which was first published by Paul Jaccard in the early 1900s.
To get the intersection and union values we first overlay the prediction boxes over the ground truth boxes (see image). Now for each class, the area overlapping the prediction box and ground truth box is the intersection area and the total area spanned is the union.




To decide wether a detection is correct or not we use the standard threshold of 0.5. If the IoU is $> 0.5$, it is considered a true detection, else it is considered a false detection. We now calculate the number of correct detections for each class in an image, using the IoU with our threshold. As we already have the number of actual objects of a given class in that image we can now calculate our Precision.

$$Precision_c = \frac{N(TruePositives)_c}{N(TotalObjects)_c} \tag{4}$$

Then we can take an Average Precision over all the images in the validation set.

$$AveragePrecision_c = \frac{\sum Precision_c}{N(TotalImages)_c} \tag{5}$$

To get our Mean Average Precision we must now take the average (mean) of all the class Average Precision values.

# 2 Analysis

## 2.1 Data Exploration

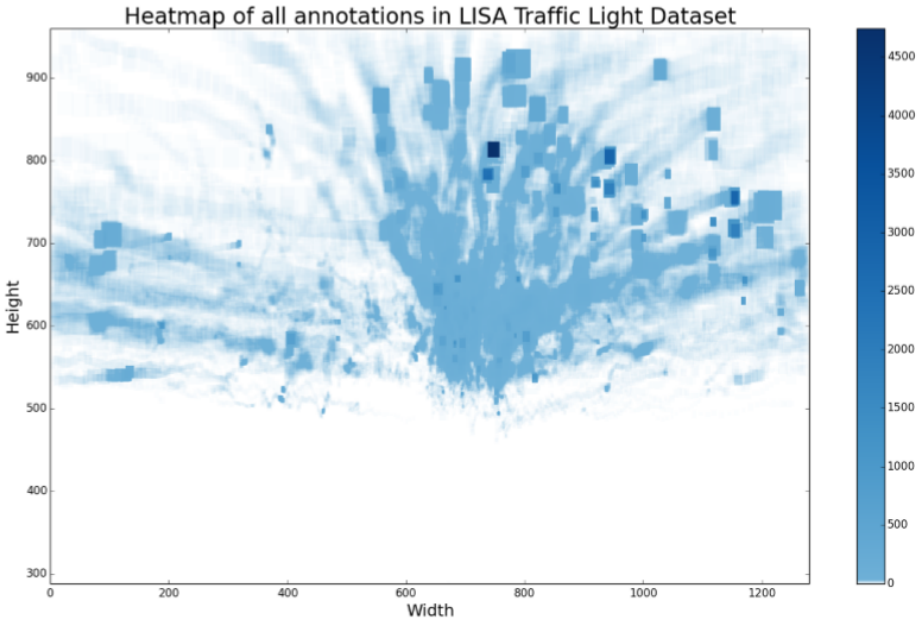
| LISA, UCSD | |
|---|---|
| #classes | 7 (go, go forward, go left, warning, warning left, stop, stop left) |
| #frames/ #GT | 43,007 / 119,231 |
| Frame spec. | Stereo, 1280 x 960, 8-bit, RGB |
| Video | Yes, 44min 41s @ 16 FPS |
| Description | 4 test seq. $\geq$ 4min and 18 training clips $\leq$ 2min 49s, urban, morning, evening, night, San Diego, USA |

A more detailed description of the data can be found in this table from [2]:

| Sequence name | Description | # Frames | # Annotations | # TLs | Length | Classes |
|---|---|---|---|---|---|---|
| Day seq. 1 | morning, urban, backlight | 4,060 | 10,308 | 25 | 4min 14s | Go, warning, warning left, stop, stop left |
| Day seq. 2 | evening, urban | 6,894 | 11,144 | 35 | 7min 11s | Go, go forward, go left, warning, stop, stop left |
| Night seq. 1 | night, urban | 4,993 | 18,984 | 25 | 5min 12s | Go, go left, warning, stop, stop left |
| Night seq. 2 | night, urban | 6,534 | 23,734 | 62 | 6min 48s | Go, go left, warning, stop, stop left |
| Day clip 1 | evening, urban, lens are | 2,161 | 10,372 | 10 | 2min 15s | Go, warning, stop, stop left |
| Day clip 2 | evening, urban | 1,031 | 2,230 | 6 | 1min 4s | Go, go left, warning left, stop, stop left |
| Day clip 3 | evening, urban | 643 | 1,327 | 3 | 40s | Go, warning, stop |
| Day clip 4 | evening, urban | 398 | 859 | 8 | 24s | Go |
| Day clip 5 | morning, urban | 2,667 | 9,717 | 8 | 2min 46s | Go, go left, warning, warning left, stop, stop left |
| Day clip 6 | morning, urban | 468 | 1,215 | 4 | 29s | Go, stop, stop left |
| Day clip 7 | morning, urban | 2,719 | 8,189 | 10 | 2min 49s | Go, go left, warning, warning left, stop, stop left |
| Day clip 8 | morning, urban | 1,040 | 2,025 | 8 | 1min 5s | Go, go left, stop, stop left |
| Day clip 9 | morning, urban | 960 | 1,940 | 4 | 1min | Go, go left, warning left, stop, stop left |
| Day clip 10 | morning, urban | 40 | 137 | 4 | 3s | Go, stop left |
| Day clip 11 | morning, urban | 1,053 | 1,268 | 6 | 1min 5s | Go, stop |
| Day clip 12 | morning, urban | 152 | 229 | 3 | 9s | Go |
| Day clip 13 | evening, urban | 693 | 1,256 | 8 | 43s | Go, warning, stop |
| Night clip 1 | night, urban | 591 | 1,885 | 8 | 36s | Go |
| Night clip 2 | night, urban | 2,300 | 4,607 | 25 | 2min 23s | Go, go left, warning, warning left, stop, stop left |
| Night clip 3 | night, urban | 1,051 | 2,027 | 14 | 1min 5s | Go, go left, warning left, stop, stop left |
| Night clip 4 | night, urban | 1,105 | 2,536 | 9 | 1min 9s | Go, warning, stop |
| Night clip 5 | night, urban | 1,454 | 3,242 | 19 | 1min 31s | Go, go left, warning, stop, stop left |
| | | 43,007 | 119,231 | 304 | 44min 41s | |

## 2.2 Exploratory Visualization

To get a better overview of where most of the annotations in the dataset lay, I have included the heatmap from [2] Fig.11:

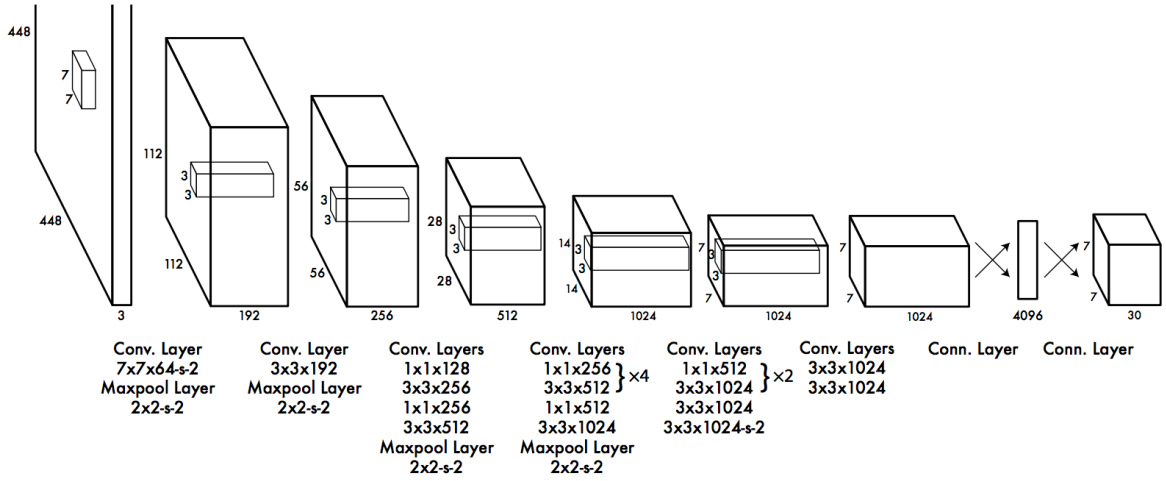



We can clearly see that most of the annotations are in the upper two thirds of each image which makes sense considering the positioning of Traffic Lights.

## 2.3 Algorithms and Techniques

The YOLO (You only look once) algorithm is popular because it achieves high accuracy while also being able to run in real-time. The algorithm looks once at the image and processes it through one forward propagation to make a prediction. After non-max suppression, we can then output bounding boxes drawn on the images to visualize the detections. For our model we have a batch of images of shape $(m, 608, 608, 3)$ as the input and get a list of bounding boxes $(p_c, b_x, b_y, b_h, b_w, c)$ along with the recognized classes as an output. We will then use non-max suppression on the output which basically means that we get rid of boxes with a low score, meaning to get rid of boxes with low confidence about detecting a class. After than we select only one box when several boxes overlap with each other and detect the same object. Note that this procedure is not part of the YOLO algorithm. It is just a nice way of visualizing it.
To process the images we use Convolutional Neural Networks which are a class of Deep Neural Networks that have been successfully applied to analyzing visual imagery.
They were inspired by the biological process in which we see images.

For example, the early layers in a CNN might detect simple features and shapes while the deeper layers connect those earlier features and detect bigger features like eyes, nose and mouth for the example of the human face.
Compared to other image-classification algorithms they use relatively little preprocessing and the filters that traditional algorithms apply are learned by the network. A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers.
Their basic building blocks are convolutional, pooling and fully connected layers, where Im gonna go more in depth while explaining the architecture we used. To get a deeper understanding of how CNNs work feel free to check out this paper [8].
To wrap up YOLO: We run an Image through a CNN which outputs a 19x19x5x12 dimensional volume. This encoding can be seen as a grid where each of the 19x19 cells contains information about 5 boxes. We then filter these boxes using non-max suppression where we are only keeping accurate (high probability) boxes and eliminate overlapping boxes. The CNN architecture being used is as follows:




The architecture is quite deep but in a nutshell we only apply a couple of Convolutional layers to detect the features and Max Pooling layers to scale down the size of our matrices. The Convolutional layers are specified by their kernel size, which is the size of the frame were using to go over the whole image and try to detect the features.
We slide the given filter over the whole image using the specified step size and as a result the network learns filter that activate when it detects some specific type of feature at some position in the input. Their filter size specifies how many filters are being used on the image and their stride size defines how many pixels are taken at a time. The Max Pooling operation looks at a frame which is specified by a kernel size and takes the maximum pixel value in that frame, which scales down the image but keeps the important features.
The intuition behind that is that the exact location of the feature is less important than its rought location relative to other features.

The pooling layer not only reduces the size and parameters and therefore the amount of computation in the network but also prevents overfitting.
Between our Convolutional and Max Pooling operations we also apply Batch normalization and use the Leaky ReLU activity function.

## 2.4 Benchmark

In [2] Table 3 we can observe the performance of many Traffic Light Detection systems with different approaches. The algorithms specifically used on our Dataset achieved a Precision of 30.1 during mixed day-time illumination conditions and 65.2 at night time and a Recall of 50.0 respectively.
Both are using aggregated channel features for detection. The only algorithm that used CNNs for classification was able to score 97.83% on Accuracy. In that case prior knowledge of the TL was used for detection.
The benchmark for our approach is the YOLO model that is being used for object detection. This model is able to find traffic lights with a good accuracy but it is not able to differentiate between the colors or identify the bulbs. Therefore our goal is to fine tune this model so it is able to detect our given classes of traffic light state with a decent precision and recall.

# 3 Methodology

## 3.1 Data Preprocessing

For the LISA dataset we get a folder of images and two *.csv* files with the Annotations for Boxes and Bulbs. Most of the Preprocessing steps consisted of concatenating the images with their annotations to create the training/validation sets. A typical row in the from one of the Annotations files would look like this (after preprocessing):

| Filename | Annotation tag | Upper left corner X | Upper left corner Y | Lower right corner X | Lower right corner Y |
|---|---|---|---|---|---|
| daySequence1–00000.jpg | stop | 710 | 481 | 714 | 486 |

The packaging of the dataset consisted of the following steps: We concatenated the Bulb- and Box-annotations in a DataFrame which we then sort to keep the original order. After that we drop the columns Origin file, Origin frame number, Origin track, Origin track frame number, as we only need the Boxes and output classes. To convert the output class label to its respective number we use a label_dict as follows:
label_dict = {"go": 0, "goforward":1, "goleft":2, "warning":3, "warningleft":4, "stop":5, "stopleft":6}.
Finally we loop over all the images and use their filename to find the corresponding labels. We und up with two numpy arrays which we can save in a *.npz* file and use as an input for our YOLO algorithm.

## 3.2 Implementation

Working with the data was quite challenging, because of the sheer masses. Manually reading, processing and packaging the data took a while and produced MemoryErrors. Therefore I had to split up the data into 12 subsets and reload the trained weights to keep optimizing them.
After retraining the model on all subsets I saved the weights and started with the necessary steps to visualize the results. The outputs of the YOLO algorithm are converted to usable bounding boxes and only the relevant boxes are kept.
The rest is filtered out using non-max suppression, which basically means that we sort out all the boxes with a confidence score below a certain threshold (we used 0.5). After those operations we can run our tensorflow session and draw the output boxes on our pictures to observe the results. For the implementation I used numpy and pandas for the processing steps of the data. To build the model I used Keras as a main wrapper to load the pretrained model from the YOLO website and then to save it after the training process. Keras is a high level implementation on top of tensorflow and makes it really easy to deploy a model quickly and efficiently.

## 3.3 Refinement

As the pretrained YOLO weights are available on the official website [website] our challenge consisted of fine-tuning them to our dataset. After some research on this problem I was able to find a Github [git] implementation which could be used to retrain the weights. Using the model architecture described in 2 on page 3 we take away the last layer of the downloaded model and substitute it with a layer that provides the shape we need for our number of outputs classes. We compile the model using the AdamOptimizer and the YOLO loss.

To compile the model we use three stages:

1. batch_size = 32, epochs = 5
2. batch_size = 8, epochs = 30
3. batch_size = 8, epochs = 30

For the training phase a Tesla K80 on an AWS instance was used. Total training time was 5 days. After training we save the weights and continue with the visualization of our output. To reach the best results for precision and recall I tried multiple values for the IoU (Intersection over Union) threshold, the outcomes can be seen in the following table:

| | Day | | Night | |
|---|---|---|---|---|
| Threshold | Precision | Recall | Precision | Recall |
| 0.5 | 26.8 | 49.4 | 4.5 | 17.6 |
| 0.4 | 36.38 | 67.01 | 7.92 | 22.53 |
| 0.3 | 46.58 | 85.79 | 14.39 | 32.04 |
| 0.2 | / | / | 26.22 | 50.13 |

# 4 Results

## 4.1 Model Evaluation and Validation

The final parameters were chosen based on the YOLO loss. After training it on the whole training set we achieved a training set loss of 2.1432 and a validation loss of 7.1652. Those values did not further decrease and considering resources and the time the model had trained by then I decided to choose this model as final. A discussion on why our model does not achieve expected performance will be delivered in chapter 5.3 on page 13. Below we can see some results from the model. The upper two pictures represent examples where it performed well. The lower pictures show examples where performance lacked.


(a) Good performance Day


(b) Good Performance Night


(a) Bad performance Day


(b) Bad Performance Night

The final mAP Score for the day and night sample frames:

| Data | mAP |
|---|---|
| day Sample | 0.05228 |
| night Sample | 0.28822 |

To elaborate on the robustness of the model I included an analysis of the different scores I achieved on subsets of the data.

| Data set | Precision | Recall |
|---|---|---|
| Day Train 4 | 31.0 | 52.3 |
| Day Train 6 | 14.4 | 32.2 |
| Day Train 10 | 15.6 | 78.8 |
| Day Train 12 | 31.8 | 58.6 |
| Day Train 13 | 33.8 | 65.7 |
| Night Train 1 | 46.6 | 85.8 |
| Day Sequence | 9.4 | 43.2 |
| Night Sequence | 29.1 | 72.0 |
| Mean | 26.46 | 61.08 |
| Std | 12.36 | 18.14 |
| Variance | 152.86 | 328.95 |

After running multiple tests with data from the training set and data from the test set we can draw the conclusion that the model is not very robust. Overall we get a standard variation of 12.36 for precision and 18.14 for recall respectively.
Those high variances show that the model has a hard time adapting to data it has never seen before. To improve the model we could add more regularization techniques like dropout or try to get more data by adding noise to the training set and generate new examples. In our case we have a high standard deviation inbetween the training samples as well which is another indicator that our model is not a sufficient solution for this problem.
We can also observe that the model generally performs better on the footage which was recorded at night so for improvement we could concentrate on fitting it better to the daytime data as the scores for the night sets are quite acceptable.

## 4.2 Justification

We improved our model so that we are able to classify the states of the traffic light now. Therefore the model is much more specialised on our domain. Observing the scores in the table above, our results are not perfect.

Our benchmark model was able to detect traffic lights, as well as many other classes of objects. In the process of improving this model we fine tuned the weights to be able to classify only traffic lights and a variety of classes such as go, warning, stop.
In the figure above we can see the achieved Precision and Recall which compares well to the methods from the paper [2] where learning based detector achieved a Precision of 30% and 65% and a Recall of 50% on a bigger test set.
We can conclude that we were able to improve our model compared to the selected benchmark but our results are not good enough to be deployed in a real life domain, especially because we are responsible for human lives in our case.

# 5 Conclusion

## 5.1 Free-Form Visualization


In our given Visualization we can observe one major issue we face when trying to classify traffic lights.
Due to different weather conditions and reflections the model can have trouble differencing between traffic lights and trailing backlights of the driver in front of us. As seen in the heatmap displayed in chapter 2.2 on page 6 , most of the traffic lights are in the upper two thirds of the picture so we could try to solve that problem by only looking for traffic lights in that space.
Another solution would be to feed in negative examples to make sure backlights are not misclassified.

## 5.2 Reflection

My approach to the problem of traffic light detection was an end-to-end deep learning algorithm called YOLO (You only look once).
To apply it to the given training data I downloaded the weights from the official website and used Transfer Learning to retrain the weights to our given classes. An important step in that process was the concatenation of the annotations and picture data.
The model was trained for multiple days on a GPU. The final results are discussed in chapter 4.
In conclusion the project is not yet ready to be deployed in a real-life setting.
Especially in the domain of autonomous vehicles we need a zero tolerance policy towards errors considering the human lives we are responsible for.
The model still shows weaknesses with trailing backlights of the car in front of our driver and sometimes misclassifies traffic lights, which can be crucial if for example a stop light is considered a warning light.
Further ideas for improvement will be discussed in the next section. Working on this problem was really interesting. It took a deep dive into the mechanics of the YOLO algorithm, one of the most powerful object detection algorithms existing, to be able to tweak it and understand how to use it for our problem. In addition it was challenging to work with this much unstructured data and being able to understand the underlying of a framework like Keras to be able to implement the Transfer Learning.

## 5.3 Improvement

In the field of end-to-end deep learning techniques I do not believe there is a better solution than YOLO right now, as it performs real-time object detection with state-of-the-art precision.
One weakness of YOLO is the correct classification of two objects that are really close together. As we recall from the earlier explanation we end up with a 19x19 matrix as the output of our model. We then look for objects in each centered in each cell.
Considering the size of the traffic lights and especially the bulbs we tried to classify, our model had significant problems finding all of the boxes. We were only able to find about half of the boxes in each picture.
A way to improve our models precision regarding this problem could be to decide on either the classification of boxes around the traffic lights or solely the classification of bulbs. If we decide to detect boxes we might have issues of detecting them at night time as one can barely recognize it then.
If we focus on detecting the bulbs we will most likely face a more extreme version of the trailing back light problem. Something that seems promising is the new version YOLOv3, but it might face the same issues as our version in terms of classifying objects that are very close.
For this problem the YOLO algorithm does not provide a solution that can be used in real-world applications and traditional methods with pipelines still outperform the end-to-end deep learning solution.

# References


[1] https://www.kaggle.com/mbornoe/lisa-traffic-light-dataset.

[2] Morten Bornø Jensen, Mark Philip Philipsen, Andreas Møgelmose, Thomas Baltzer Moeslund, and Mohan Manubhai Trivedi. Vision for looking at traffic lights: Issues, survey, and perspectives. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):1800–1815, 2016.

[3] Mark Philip Philipsen, Morten Bornø Jensen, Andreas Møgelmose, Thomas B Moeslund, and Mohan M Trivedi. Traffic light detection: A learning algorithm and evaluations on challenging dataset. In *intelligent transportation systems (ITSC), 2015 IEEE 18th international conference on*, pages 2341–2345. IEEE, 2015.

[4] https://www.kaggle.com.

[5] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[6] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[7] https://pjreddie.com/darknet/yolov2/.

[8] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.