License Plate Detection Utilizing Synthetic Data from Superimposition

Olof Harrysson

Master's thesis
2019:E18

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

License Plate Detection Utilizing Synthetic Data from Superimposition

Olof Harrysson
harrysson.olof@gmail.com
I love getting emails. Email me!

May 27, 2019

Master's thesis work carried out at Axis Communications.

Supervisors: Anton Jakobsson, anton.jakobsson@axis.com
Henning Petzka, henning.petzka@math.lth.se
Ted Kronvall, ted.kronvall@math.lth.se

Examiner: Kalle Åström, kalle@maths.lth.se

Abstract

Machine learning projects are often constrained by data that is messy, scarce, sensitive or costly to produce. These are issues which could be mitigated by synthetic data.

This thesis tries to improve Swedish license plate localization in images by synthesizing images through superimposition, a process that produces data cheaply and in abundance.

A generative process algorithmically creates images of fake license plates which are glued on top of other images without any context awareness. The resulting images thus contain license plates where they normally don't exist, floating in the sky or across a person's face.

To evaluate the usefulness of this method, a machine learning algorithm called YOLOv3 is trained on synthesized data and tested on real images containing Swedish license plates.

Our findings show that training on synthesized data by itself is almost good enough to match the performance of training on publicly available real data containing international license plates. The best results are achieved by mixing real and synthetic data.

Keywords: AI, machine learning, deep learning, CycleGAN, YOLOv3

Acknowledgements

I'm grateful for the support given by my supervisors. Thank you Anton for showing me the ropes at Axis and many thanks to Henning and Ted for your involvement and contributions to the project.

I'd also like to thank Kalle for believing in the project, giving it the go ahead. Much love to my colleagues at Axis who've helped me in various ways and contributed to me having a wonderful time there.

Finally, I appreciate the support and interest shown from friends and family.

Contents

Chapter 1

Introduction

Interest for artificial intelligence (AI) has exploded in recent years. AIs can nowadays create realistic images of faces [1], play complex computer games better than humans [2] and write coherent articles [3].

These technologies are all based on a subfield of AI called machine learning. Machine learning algorithms work without explicit instructions from humans but instead infer knowledge from data. Today, all eyes are on machine learning models which make use of neural networks. Neural networks are algorithms loosely based on our brains where numbers propagate through spider-web like structures to form useful predictions. One reason why the networks are so powerful is their large amount of adjustable parameters which help them approximate very complex functions. However, this comes at the cost of requirements for vast amounts of data before they work properly. In many cases real data is messy, scarce, costly to produce or difficult to work with due to privacy concerns.

This thesis aims to investigate usefulness and limitations of synthetic data which has the potential to alleviate these pain points of real data. As suggested in [4], AI breakthroughs could be closer tied to new datasets or techniques to generate data than new algorithms.

To evaluate if it's possible to use a certain kind of synthetic data, explained shortly, the problem of **locating Swedish vehicle registration plates** in images was selected. The hypothesis is that since Swedish license plates follow a strict schema, they can be generated rather easily and truthfully compared to other entities such as cars or people. Perhaps this directed data of Swedish license plates could be leveraged to create a specialized detector that works well on Swedish cars. If so, other region specific license plate detectors would also be possible.

The generative process creates images of licence plates (LP) with approximate dimensions,

features and colors via a script. The **LPs are then superimposed** on (a.k.a. glued on top of) other images referred to as background images. The location within the background image is random thus LPs end up where you'd normally wouldn't find them - floating in the sky or squarely across a person's face.

To build upon this, a Generative Adverserial Networks (GAN) variant called **CycleGAN is used to transfer realistic features from real LPs to generated LPs** in a data driven manner. The motivation is given by the intuition that superimposition of more realistically looking LPs may improve performance for LP detection.

The generated images are used to train a popular **object detection algorithm called YOLOv3** which is tested on a **dataset of images containing vehicles with Swedish LPs**, collected for the purpose of this thesis. The results are compared against models that have been trained on real data or mixes of synthesized and real data.

It is shown that synthesized data by itself is almost good enough to match the performance of publicly available real data and that the best results are achieved by mixing real and synthetic data.

Chapter 2

Theory

2.1 License Plate Detection

There are a number of systems that are used to detect and read license plates. Police forces in Sweden utilize dashboard cameras in their cars to automatically alert the officer whenever the camera finds a car that's been stolen or isn't allowed to be on the road. This helps the police to screen large amounts of vehicles with fewer resources [5].

In Gothenburg, Sweden, cameras designed to read LPs form a perimeter around the city to charge drivers going through the city. If one drives through the city, the cameras will read the license plate and send a bill to the owner of the car. The cameras are mounted over the highway and shine light on the reflective LP to more easily find and read it.

The above mentioned systems use two stages to read the LP. The first stage detects where the LP is located and crops it out. The second stage uses the cropped image to read its contents. If one could improve the detection of LPs, the system would work better as a whole. There are also applications where one only wants to detect the LP, for example by blurring LPs to mask out identifying information. Or perhaps a camera that could change its exposure and other settings to get a clearer view of cars who steal fuel at gas stations. Finally, systems capable of spotting and identifying vehicles are crucial if our transportation industry is to develop towards autonomy.

This thesis focuses on the first stage, namely finding the location of the LP which falls under the suite of algorithms called object detection. **Reading the contents of the LP is never performed** to keep the thesis scope manageable.

Object detection (OD) is a subfield of computer vision where an algorithm tries to locate semantically meaningful objects within images or videos. Commonly, OD algorithms are

built to recognize objects of certain classes such as cats, faces or cars. The output from these algorithms could be segmentation masks marking the objects of interest or polygons that encapsulate the object. Rectangles, also called bounding boxes, are usually used to mark the location of objects. See Figure 2.1 for an example of bounding boxes found by an OD algorithm.

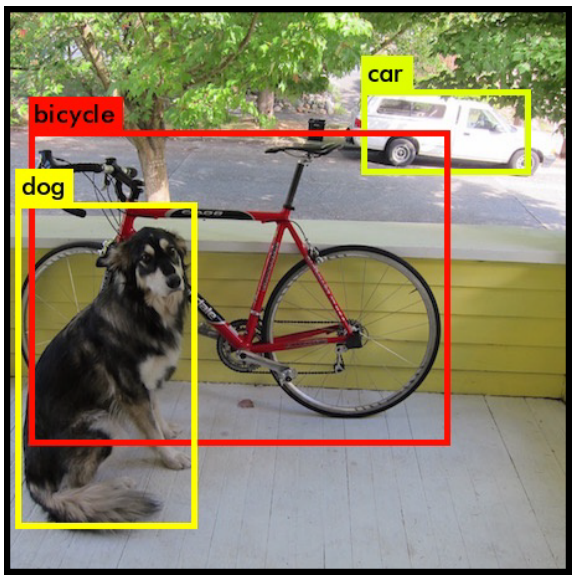


Figure 2.1: Bounding boxes that describe objects' location and class.

2.2 Machine Learning Intro

Machine learning models are algorithms that have the capacity to learn from data without explicit directives on how to arrive at sought conclusion. This section aims to introduce a few machine learning concepts to people who are unfamiliar to the field.

Teaching a machine learning model is done by feeding it data and is referred to as training the model on that data. To judge model performance, one wants to examine its behavior on a different dataset from the one it was trained on, to check whether the model can generalize to unseen data. This is similar to how, e.g., math exams contain slight alterations to the practice questions so that the student can't simply memorize the answers.

It's therefore common to divide a dataset into three pieces - training, validation and test data. Training data is the data which is accessible to the model to learn from. During the process of training a model on the training data, the validation data can be used and studied to get a sense of how well the model generalize to this, for the model, unseen data.

Once the programmer is happy with how the model works, all the parameter tuning and training stops. The, up until now hidden, test data is presented to the model to evaluate final performance. **The performance on test data best describes the usefulness and capabilities of a model** whereas validation data is used as a tool to improve the training procedure.

There are different types of machine learning models. This paper will solely work with the subfield called deep learning, which recently has caused much media hype. Intuitively deep learning can be thought of as a series of computations, arranged in layers, where each layer adds insights to the previous step. The word deep in deep learning corresponds to the number of layers the model has.

A common type of deep learning models are called artificial neural networks, originally built to mimic how the neurons in our brains are connected. Neural networks are directed graphs which consist of a large collection of connected nodes in layers where each layer is a structure capable of one or more mathematical functions, typically both linear and non-linear. The input data is transmitted through the layers, each one transforming the data in such a way that when passed through all layers, it has become the sought output.

One layer that is used often is the convolutional layer [6], thanks to its suitability for image data. It operates by convoluting a set of filters with the image, where different filters activate or search for different things. One filter might detect an ear, and another one a nose. The detected features are passed to the next layer which can combine them and conclude that, given an ear and a nose activation, it's probably an image of a face.

Another layer that is often put right behind the convolutional layer is called the batch normalization layer [7]. This layer normalizes its input data to save the next convolutional layer from dealing with either very large or small numbers, typically improving the stability and performance.

The model used in this thesis mainly consists of said layers and so called residual connections [8], which concatenates layer outputs from different depths. All of the model's layers can handle inputs of different spatial sizes which translates to the whole model inheriting this perk. The model is therefor fully convolutional and can operate on images of different sizes.

The goal is ultimately having the model find a mapping from input space X to output space Y, where in our case X represent images and Y locations of license plates. If the X and Y pairs are known, it's possible to steer the model's learning by letting it know whenever it makes a mistake; a method referred to as supervised learning.

This is done through an objective function capable of measuring proximity between model predictions and Y. By calculating the gradients of this function with respect to the data, one may update the model parameters in such a way that the objective function decreases.

The process of producing predictions and updating the model parameters is iterated a large number of times, resulting in the model making fewer and fewer mistakes. For computational reasons, the training data is divided into smaller chunks, called mini-batches, that separately take part in the update cycle.

2.3 Datasets

There aren't many publicly available datasets of LPs with a significant amount of images[1]. The dataset with the largest amount of LPs known to us is the Open Images dataset. **Open Images v4** [9] is an object detection dataset released by Google consisting of ~9 million images of which 1.74 million are selected as training data. There are 600 object classes and an average of 8.4 objects per image. The labeling is done by both humans and machines and it's quite common that there are missing or incorrect labels for objects in the dataset. Training data for the class "vehicle registration plate" contain ~5400 images and ~7850 annotated licence plates. The images are taken from different locations around the world and thus contain LPs with a wide range of diversity e.g. color and font.

As the thesis focuses on detecting Swedish LPs it wouldn't be suitable to test model performance on Open Images where LPs aren't Swedish and some labels are incorrect.

A dataset, dubbed **SweLP**, exclusively consisting of Swedish LPs were therefore assembled from two sources. The majority of the images were shot on a Sony Cybershot Dsc-Hx20V camera model in Lund, Sweden, with a low shutter speed whilst riding a bike. Under these circumstances many images came out quite noisy or blurry.

The second source are pictures taken of cars for sale donated by an affiliate. The dataset was then further filtered to only contain Swedish licence plates from civilian cars i.e. no taxi or military plates. It consists of 761 training images containing 829 licence plates. The validation and test data each contain 290 images and ~320 LPs.

Swedish LPs are much more standardized in comparison to the mix of LPs found in Open Images. If the goal is to train a detector that works well on Swedish LPs, one can imagine that there could be gains in conveying the structure of a Swedish LP to the detector. During the thesis a method to create **Superimposed Licence Plates** datasets was created. It consists of artificially generated Swedish licence plates on background images from the COCO dataset. The method can create datasets to specific sizes with different attributes by changing its parameters, see chapter 4.1 Data Generation by Superimposition.

The **COCO** dataset [10] is a joint venture between several companies and universities. Its newest object detection fragment released in 2017 consists of approximately 118 000 images spanning 80 classes of objects. This project only utilized COCO as background images for the synthesized datasets and could have been replaced by other images. However, COCO was implicitly used further since YOLO, our LP detector, was pretrained on it which will be explained in section 2.4 YOLOv3.

See Figure 2.2 for image samples from **Open Images v4**, **SweLP** and **COCO**.

[1] See platerecognizer.com/number-plate-datasets/ for most available LP datasets

(a) Open Images v4

(b) SweLP

(c) COCO

Figure 2.2: Sample images from datasets used in the project. Note that some of the images are cropped to fit in the collage.

2.4 YOLOv3

2.4.1 Overview

YOLOv3 [11] is the third version of a popular object detection algorithm praised for its speed and accuracy released in March 2018. Under the hood YOLO is primarily a neural network consisting of mostly convolutional layers, batch normalization layers and residual connections. It belongs to the school of Single Shot Detectors (SSD) which only needs a single series of computations to detect objects. This explains the acronym You Only Look Once. Detectors that aren't SSDs (e.g. Faster R-CNN [12]) often propose regions of interest that are processed separately, typically making it slower and more accuracy.

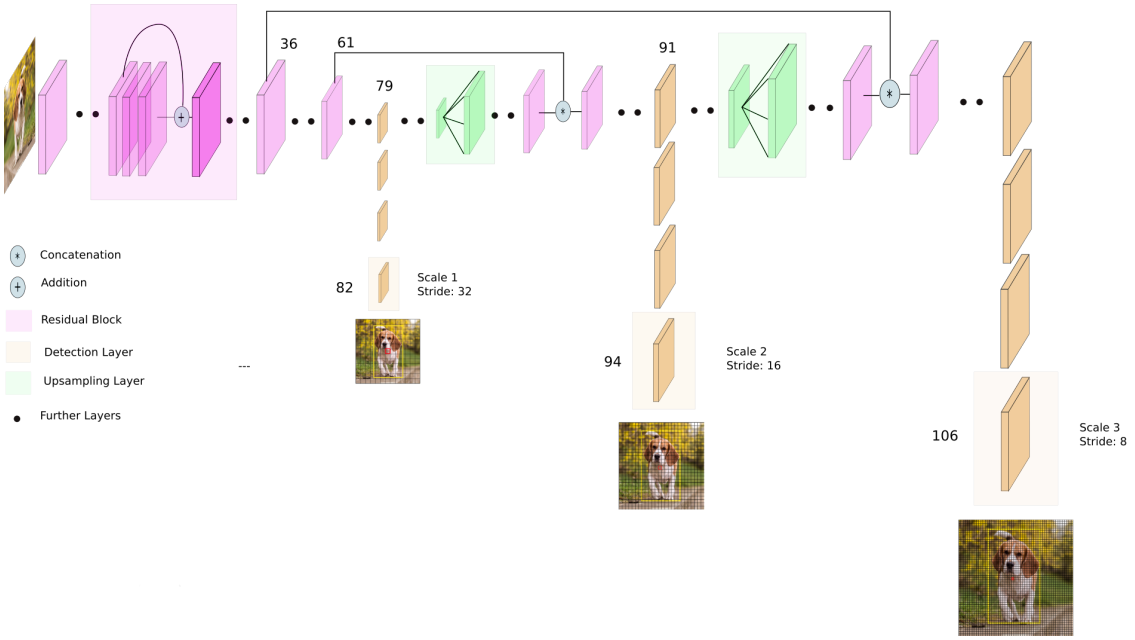


Figure 2.3: YOLOv3 architecture. Credit goes to Ayoosh Kathuria [13].

2.4.2 Input, Computation & Output

Input

As most object detection algorithms, YOLO operates on color images as input. Since YOLO is fully convolutional it can process images of different sizes where larger images give better accuracy but sacrifices speed. Typically the image size is chosen from within the range of 300 to 600 pixels and needs to be evenly divisible by 32 as the image is down sampled by 32 in the network. If the input images aren't already compliant with the size requirements, they have to be resized, cropped or padded before they can be processed by YOLO.

Computation

YOLO detects objects at three stages (also referred to as scales) throughout its network. This is done to better handle objects of varying sizes. The first stage is represented by the smallest dog picture in Figure 2.3. In this stage the image has been down scaled by a large factor which leads to the dispersion of fine grained spatial information. This stage is therefore better at detecting large objects where small spatial errors don't affect the result as much. To demonstrate this, see Figure 2.4.

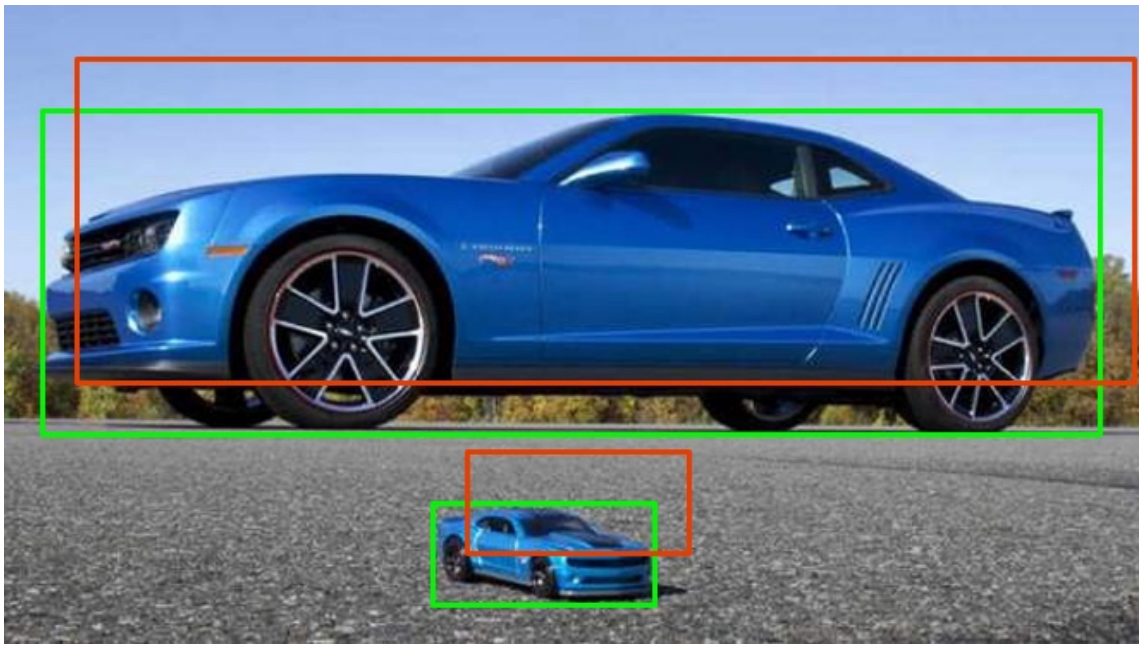


Figure 2.4: Green squares show the correct bounding boxes for the two cars. The same absolute offset for the bounding boxes, shown in red, gives higher relative error for the small car.

After the first stage the signal flowing through the network is scaled to a larger spatial size and concatenated with a signal produced earlier, again refer to Figure 2.3 for a visual illustration. The resulting signal is used in the second stage where medium sized objects are detected. The third and last stage works analogously and detects small objects.

Output

Once the heavy computations are done, YOLO needs to transform the numbers given by the neural network to bounding boxes that encircle the found objects. It does this rather uniquely by dividing the image in grids and predicts several bounding box candidates for each cell. Each bounding box candidate consists of a

- x & y-coordinate
- box width & height
- confidence score which specifies how likely the box contains a sought for object
- list of class certainties that determine what kind of object it is [2].

[2]Class certainties aren't needed when there only is one class e.g. license plates

The x and y-coordinates doesn't directly represent the coordinates on the image but rather the coordinate within the grid cell. To translate this representation to absolute coordinates on the image, x and y are offset differently depending on which cell they exist in.

YOLO also utilizes anchor boxes that serve as priors for the box width and height. A set of anchor boxes, composed by a width and a height, is constructed to roughly take the same shapes as common objects in a dataset. The boxes could be constructed manually or algorithmically via e.g. k-means. The idea behind anchor boxes is that it's easier to predict the needed changes for them than directly predicting the width and height of an object. The box width and height that YOLO produces are in fact multipliers that are used to scale the priors to the final bounding box shapes.

If one were to visualize the bounding boxes at this stage it wouldn't even be possible to see the actual image due to the sheer quantity of bounding boxes. Fortunately, a well trained model will only produce a small amount of bounding boxes with high confidence scores whilst the **lower confidence boxes are disregarded**. Changing the confidence threshold is how one can tune the sensitivity of the model.

2.4.3 Pretraining & Training

A common method used today is to repurpose an already optimized neural network to solve similar problems. It turns out that networks capable of detecting a class e.g. cars, learns representations useful for detecting other objects as well. This is because early layers of a neural network learns to find simple features such as edges or corners. These features are combined throughout the network to build more complex representations such as wheels, headlights or windows. A network which already knows how to spot cars needs smaller adjustments to find classes with similar features e.g. buses, than a randomly initialized network. The early layers capable of finding edges and corners prove beneficial for finding seemingly unrelated classes such as dogs.

YOLO utilizes pretraining by first training its submodule Darknet53 on a large dataset named **ImageNet** [14]. This is done in a classification matter which means that the network learns to classify the primary object contained in an image but doesn't need to locate it. The complete network, containing 75 convolutional layers, was then optimized to find objects from the **COCO** dataset.

2.5 CycleGAN

2.5.1 Overview

CycleGAN [15] is an unsupervised machine learning method capable of image-to-image translation. See Figure 2.5 for an example of CycleGAN turning zebras into horses without specifying what a zebra or horse is. Ideally everything "zebra style" should turn into "horse style" whilst image structure stays the same. As you see in Figure 2.5b, the transformation has shifted the grass' color from green to yellow, a common grass color where zebras live.

What is really wonderful about CycleGAN is that it doesn't require paired data. This means that it doesn't need horses and zebras to stand in the same position and occupy the same part of the image for the translation to work. It just requires a number of images (or other types of data) for the two domains.

(a) Zebras ⇒ Horses

(b) Horses ⇒ Zebras

Figure 2.5: CycleGAN translating images between the zebra and horse domain.

2.5.2 Generative Adverserial Networks

CycleGAN belongs to the set of algorithms called Generative Adverserial Networks (GAN) [16]. As the name suggests, GANs generate data by the use of neural networks. Some GANs can generate e.g. horses out of zebras whilst others generate outputs from inputs consisting of pure noise.

A common process of generating images from noise starts with generating a random vector z. z is used as the input to a neural network which outputs numbers that represent pixel values for an image. If the image is to be in color with a width and height of 10 the network needs to output $3 * 10 * 10$ numbers. How the network creates *meaningful* images isn't obvious and will be explained shortly.

There are many ways to generate z and decide the structure of the network but the original GAN created by Goodfellow et al. used uniform random noise and a fully connected network. It consists of two main components, the generator and the discriminator whose role we explain in the following.

The goal of a GAN is to create data that resembles a target distribution as closely as possible. This task is given to the generator. Unfortunately, the generator isn't yet aware of what real data entails so starts of by producing outputs at random. These guesses are given to the discriminator along with real examples of data.

The second component, the discriminator, is given the guesses from the generator and real samples from a dataset representing the target distribution. Its job is to figure out which samples are real and which ones are fake. If the discriminator correctly identifies the fake data, that means that the generator didn't produce a good enough counterfeit. When this happens, the generator is adjusted to produce data that *would* have fooled the discriminator. On the other hand, if the discriminator is fooled by the fake data it gets adjusted to *not* be fooled by that particular example again.

The generator and discriminator are pitted against each other in this adverserial game of cat and mouse. The generator becomes better at generating data that fools the discriminator and the discriminator becomes better at spotting fakes. If the generator would be able to create a perfect copy of an image, albeit without seeing it, there would be no way for the discriminator to tell them apart. In practice this never happens but after a while the generator (hopefully) produces data that is very similar to real data. The process is then complete and one can use the generator to produce data without further need for the discriminator.

GANs have recently been a hot topic with new architectures and ideas sprouting up everywhere. They all share the notion of generators and discriminators but improve upon the original to get better results or solve other problems. CycleGAN is no exception.

2.5.3 Architecture & Loss

Before diving into the inner workings of CycleGAN, let's recap on its purpose. CycleGAN is capable of converting data between two domains e.g. horses and zebras or in our case real and generated LPs. This is called image-to-image translation. CycleGAN differs from the original GAN explained in 2.5.2 in that it uses an existing image as input, rather than random noise, to produce an output image.

To do this, CycleGAN utilizes two generators and two discriminators. The first generator, G_{A2B}, translates data from the input domain A to the output domain B whilst the second generator, G_{B2A}, does the opposite translation - B to A. The discriminators D_A and D_B try to tell real and generated data apart in their corresponding domains. To exemplify this, D_A tries to separate real data in domain A from the data that is generated by G_{B2A}.

For any generator G that translates from domain X to Y and D_Y, a discriminator of samples in Y, the loss function is as follows:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y, X, Y) = &\mathbb{E}_{y\sim p_{data}(y)}[logD_y(y)] \\ &+ \mathbb{E}_{x\sim p_{data}(x)}[log(1 - D_yG(x))]\end{aligned}$$

A generator tries to fool its discriminator by creating realistic data according to the target domain. However, $\mathcal{L}_{GAN}$ doesn't care *how* this done. $\mathcal{L}_{GAN}$ doesn't incentivize the generators to turn zebras into horses but rather to create horses by any means necessary. Something else is needed when the goal is to do image-to-image translation.

CycleGAN tries to solve this by introducing a cycle loss. After G_{A2B} has transferred a data point to domain B, that data goes through G_{B2A} back into domain A. The final result is compared against the original data and any differences add to the loss.

Because the data eventually has to be mapped back to itself, the generators have to preserve the information in the data and at the same time produce forgeries that can fool the discriminator. The cycle loss is what made CycleGAN so innovative.

Cycle Loss:

$$\begin{aligned}\mathcal{L}_{cycle}(G, F) = &\mathbb{E}_{x\sim p_{data}(x)}[\ \|F(G(x)) - x\|_1\ ] \\ &+ \mathbb{E}_{y\sim p_{data}(y)}[\ \|G(F(y)) - y\|_1\ ]\end{aligned}$$

The total loss is obtained by:

$$\begin{aligned}\mathcal{L}(G_{B2A}, G_{A2B}, D_A, D_B) = &\mathcal{L}_{GAN}(G_{B2A}, D_A, A_{fake}, A_{real}) \\ &+\mathcal{L}_{GAN}(G_{A2B}, D_B, B_{fake}, B_{real}) \\ &+\mathcal{L}_{cycle}(G_{A2B}, G_{B2A})\end{aligned}$$

The objective is to minimize the total loss for the generators and maximize it for the discriminators. If the objective is solved well, the generators can be used to transfer data between the domains and the undertaking is complete.

$$G_{B2A}, G_{A2B} = \arg \min_{G_{B2A}, G_{A2B}} \max_{D_A, D_B} \mathcal{L}(G_{B2A}, G_{A2B}, D_A, D_B)$$

Chapter 3

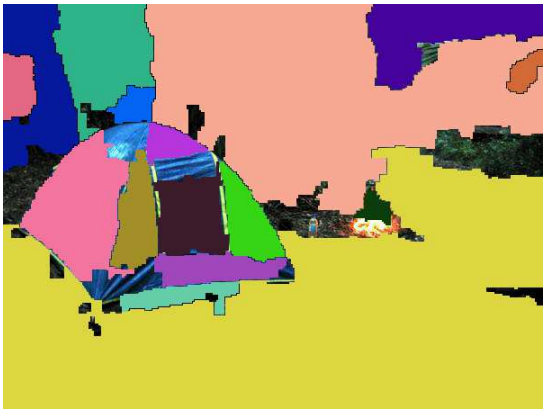
Related Work

3.1 Synthetic Text for Detection

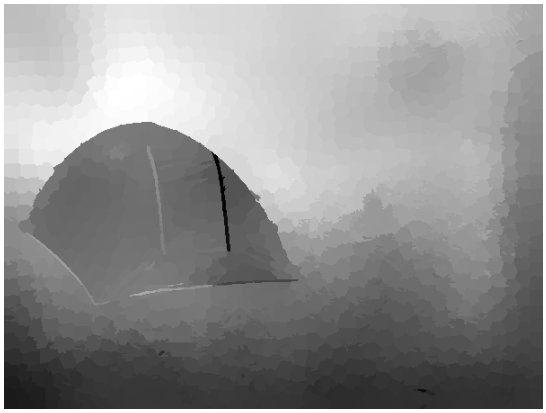
Gupta et al [17] showed that it was possible to create a competitive text detector trained exclusively on superimposed data. This superimposed dataset used natural background images and generated text which is superimposed, or glued onto, the background image. They found regions characterized by uniform color and texture as locations where text could be added. This was done to avoid the superimposed text overlapping regions without contextual meaning e.g. a horizon and a rooftop. Additionally they used the found regions to predict a depth map which was used to transform the text to fit the image curvature. The generated text was drawn in different fonts, sizes, colors etc. The text to be added was sampled from the **Newsgroup20** dataset and consisted of words, lines or paragraphs. See Figure 3.1 for an example of the process and resulting images.

As superimposed text has been shown to work as training data for text detection, this thesis investigates whether the same holds true for superimposed LPs.

(a) Input Image



(b) Found Regions



(c) Predicted Depth Map

(d) Final Image

Figure 3.1: Data generation pipeline to superimpose text in images [17].

3.2 Synthetic License Plates

License plate recognition (LPR) is a technology that tries to read the information within a LP. Most LPR-systems read the LP given a cropped image of a LP but it is also possible to include the task of detecting the location of the LP into the LPR-system in an, so called, end-to-end fashion.

In [18], H. Li, P. Wang and C. Shen did end-to-end reading by creating a network with two branches. One was tasked with finding the plate and the other with reading it .

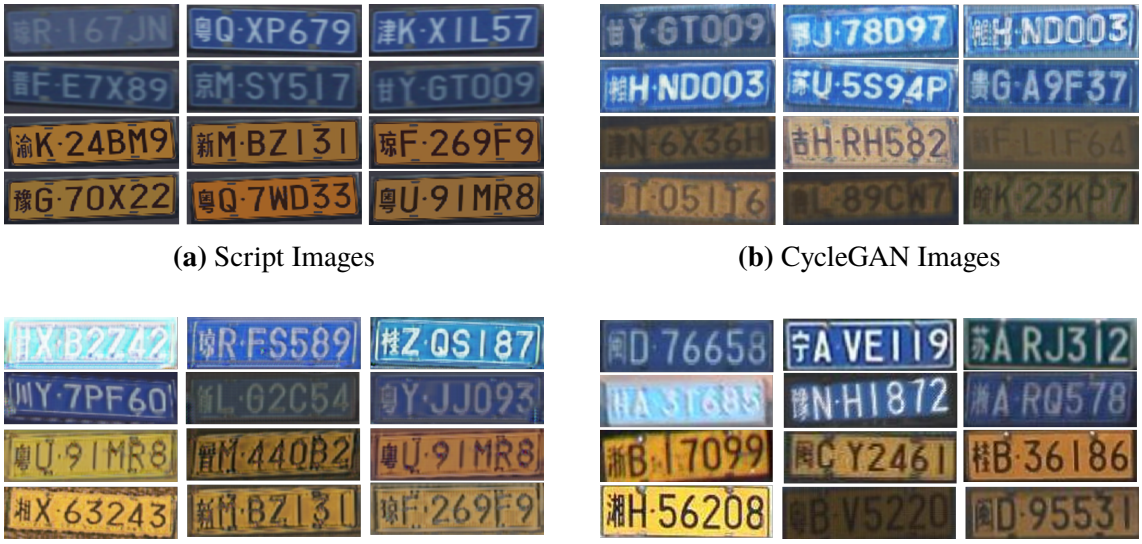
In the work carried out by Wang et al. [19] the effect of training a deep learning model on synthetic data for LPR was investigated. They focused on recognizing cropped images of Chinese LPs which means that the algorithm didn't need to detect the LPs.

Three approaches to synthesize training data were evaluated. The first one generated LPs via a script in accordance to the structure of Chinese LPs. The second method used the script images and ran them through a CycleGAN with real plates as the target domain. The last dataset was created the same way with a CycleGAN variant called CycleWGAN. See

Figure 3.2 for examples of the different datasets.

It was shown that adding any of these datasets to real data increased the models performance, most significantly when the amount of real data was low. It was also shown that training exclusively on images produced by the CycleWGAN could give decent results.

Given that their results showed that a Cycle(W)GAN could increase the LPR score it could perhaps also help in detecting LPs.



(a) Script Images

(b) CycleGAN Images

(c) CycleWGAN Images

(d) Real Image

Figure 3.2: Examples of generated images of LPs used in [19].

Chapter 4

Method

4.1 Data Generation by Superimposition

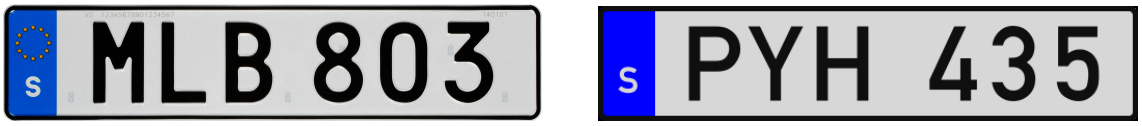
4.1.1 Overview

As mentioned in the 2.3 Datasets section, an automated way to create synthetic Swedish LP data was developed during this thesis. The created images don't resemble real data as the LPs are superimposed onto arbitrary backgrounds and appear misplaced. The goal is to create realistic looking LPs, even though the LP's context is unnatural. To generate a realistic looking LP, one first has to know its characteristics.

4.1.2 Swedish License Plates

The Swedish license plates produced today follow a strict schema. The identifying information always consists of three letters followed by three numbers[1], which are written in a closed-source font specifically created for Swedish license plates. The plate behind the letters is primarily light-gray and made in a reflective material. Swedish LPs have shape- and other requirements but as this isn't a centerpiece for understanding the thesis they won't be defined. Instead see Figure 4.1a for an example of a LP.

[1] As of 2019 the schema has changed due to a shortage of available plates. The last number can also be a letter.



(a) Real License Plate **(b)** Fake License Plate

Figure 4.1: Comparison of a real and generated license plate.

4.1.3 Generation

The generation process consists of three steps. A LP is **conceived**, **augmented** and **superimposed** onto a background.

Conception

In the conception phase, a plain image of a LP is created through a Python script in the SVG file format. The creation process starts of with creating a light-gray rectangle which has its leftmost part colored blue and and a black border around them both. Letters and numbers are sampled randomly before adding them to the plate along with a tiny S for "Sweden" in the blue area. The circle of starts representing the EU is not reproduced. Since the font for Swedish license plates isn't publicly available, a similar font called DIN 1451 [20] was used instead. See Figure 4.1 for a comparison between a real LP and one generated via this method.

Augmentation

The augmentation pipeline is a result of trial and error. The augmentation parameters are tweaked to produce LPs that both look realistic to the human eye and give good results in the experiments.

By inspecting a histogram of LP widths from **SweLP's** training data it was observed that the histogram resembled a gamma distribution. This histogram was approximated with a scaled version of $\Gamma(k = 3, \theta = 2)$ and sampled from to scale the LPs to **different sizes**.

For the generated LPs to resemble LPs in reality they had to appear to be viewed from **different angles**. A geometric transformation was applied to the LPs to mimic this - see Figure 4.2. The LP was first rotated around the x and y axis with $\hat{x}$ & $\hat{y}$ degrees. This was followed by a rotation around the z axis with $\hat{z}$ degrees. The angles were sampled uniformly from the intervals:

$$\hat{x} \in \mathbb{R} \mid \hat{x} \in [-10, 10]$$
$$\hat{y} \in \mathbb{R} \mid \hat{y} \in [-75, 75]$$
$$\hat{z} \in \mathbb{R} \mid \hat{z} \in [-10, 10]$$

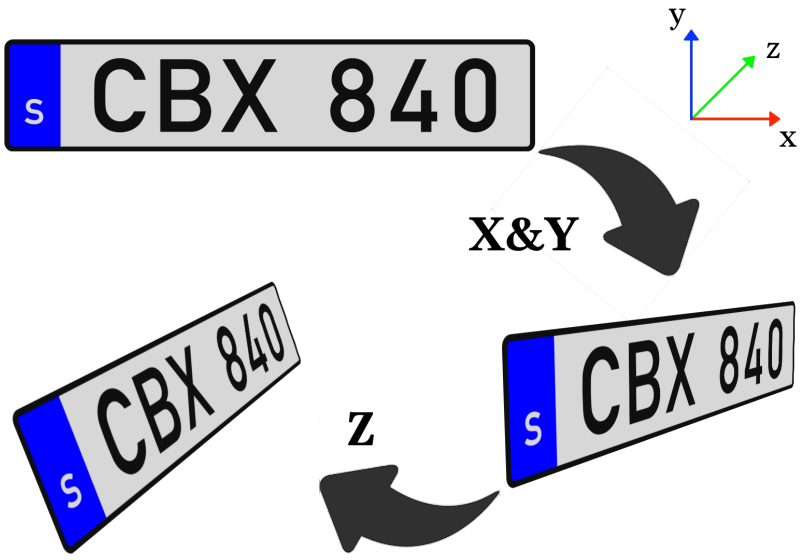


Figure 4.2: The image is first rotated around the x and y-axis. Thereafter the resulting image is rotated around the z (depth) axis.

Because photos of real LPs are taken under different lighting and weather circumstances a series of **color augmentations** were added to the generated LP in hopes that the result would be more realistic and diverse. Brightness (in all color channels and separately), noise, motion blur and jpeg compression was added randomly with the Imgaug [21] library. See Figure 4.3 for a sample of generated plates.

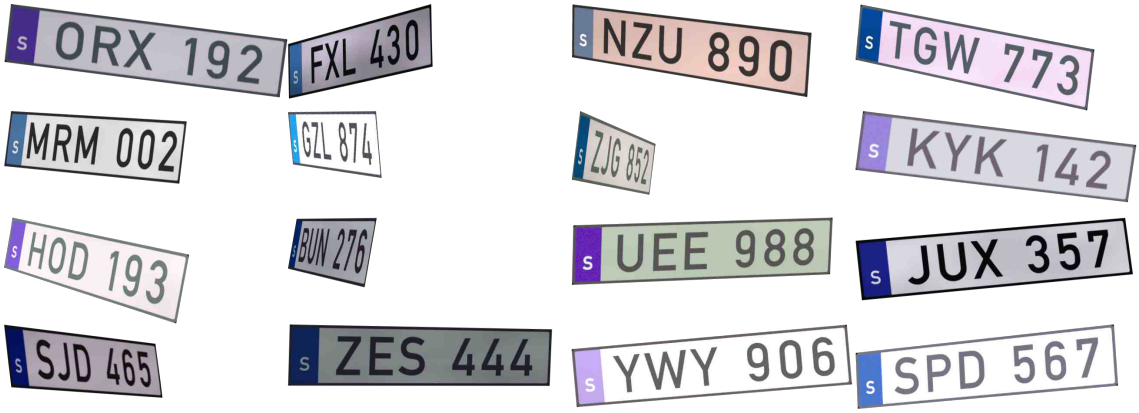


Figure 4.3: A collage of license plates that have been augmented.

Superimposition

It isn't enough to create a synthetic LP if the goal is to find LPs within images. The last step of the generation process is therefore to superimpose LPs onto backgrounds.

A first attempt at selecting background images was to find images set in an urban environment, a setting cars normally appear in. Open Images provide such images but unfortunately they contain a significant amount of unlabeled LPs which render them unusable for our needs. Unlabeled LPs would confuse YOLO and result in sub-optimal performance.

The background images were thus taken from the COCO dataset. This dataset contains images of cars, motorcycles, buses, trains and trucks - all images that might contain unlabeled LPs. The images that contained any of said classes were thrown away and 8192 background images remained. It's worth noting that a subset of COCO consisting of 10 000 images was utilized rather than the full dataset of over 100 000 images.

The last step of the process was quite straightforward. The generated LP was superimposed onto a background image at a random position and the position and dimension of the LP were saved and used as a label. The background images were picked sequentially and recycled for large datasets. **This method of creating datasets will be referred to as ScriptLP** and a few examples can be seen in Figure 4.4.

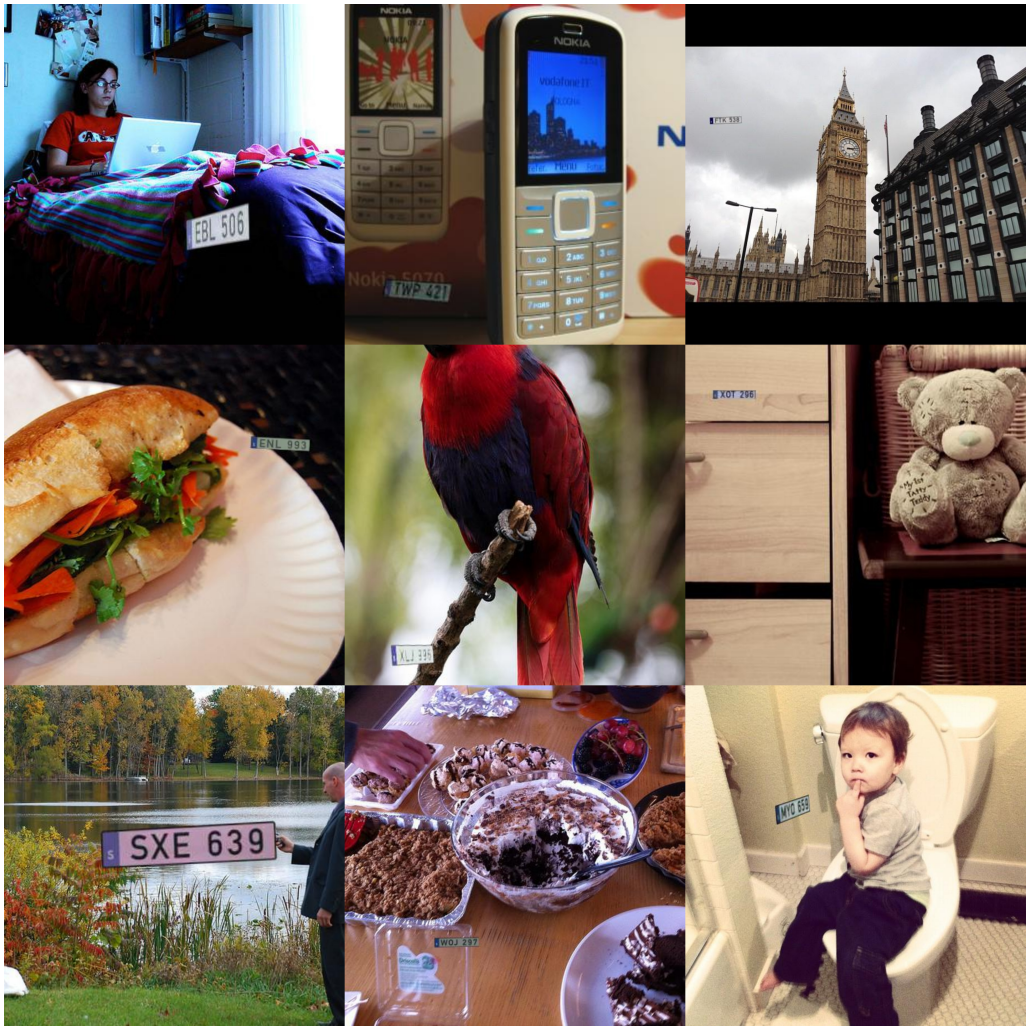


Figure 4.4: Images that were created through superimposition.

4.2 YOLOv3 Implementation

A significant amount of effort was put into re-implementing the YOLOv3 algorithm in Pytorch, a deep learning library for Python. This was done to more easily be able to change code surrounding YOLO such as learning rate annealing or training progress visualization. One useful feature developed in this project was the ability to validate the progress on several datasets instead of merely one. This helped to check how the model was performing on the **SweLP** dataset during training on another dataset.

Unfortunately, implementing YOLO comes at a cost other than the time spent doing it. There is no guarantee that the created program works the same way as the original as there is a high risk of introducing bugs. An attempt to validate this version was made by benchmarking on the COCO dataset but this proved harder than first expected and wasn't pursued further. If there are bugs, they don't appear to be critical as this YOLO appears to perform normal. Nevertheless, one crucial component that is completely lacking in this implementation is the data augmentation pipeline that is often present in machine learning projects training on images. This wasn't done due to implementation issues and time constraints.

4.3 CycleGAN Implementation

As discussed in chapter 2.5, CycleGAN is capable of image-to-image transformation e.g. translating images of zebras into horses. As there is little need to transform anything into a horse in this project, the intent of using CycleGAN is to introduce realistic features, such as shadows or dirt, onto fake LPs. To conceptualize this, the goal is to transform synthesized ScriptLP data onto the real domain of SweLP to make the synthesized data look more realistic. The created LP image is then superimposed onto a background in the same way as in 4.1 Data Generation by Superimposition.

To achieve this, an implementation of CycleGAN by Aitor Ruano was utilized. [22] This version was used instead of the original because of its ease of use. There are some differences to the original. Most importantly, one key difference is that the loss function differs.

The first modification replaced the logarithmic error by a squared error such that the GAN loss becomes:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y, X, Y) = &\mathbb{E}_{y \sim p_{data}(y)}[D_y(y)^2] \\ &+ \mathbb{E}_{x \sim p_{data}(x)}[(1 - D_y G(x))^2]\end{aligned}$$

The second introduced change is an identity loss. The idea is that data from domain A shouldn't change when it's fed through the G_{B2A} (which translates a data point from domain B to A) since it's already a member of the target domain. In other words, data going through the opposite generator should map back onto itself, $f(x) = x$. This identity loss is added

to the total loss and is defined as:

$$\begin{aligned}\mathcal{L}_{identity}(G,F) = &\mathbb{E}_{y\sim p_{data}(y)}[\,\|G(y)-y\|_1\,] \\ &+\mathbb{E}_{x\sim p_{data}(x)}[\,\|F(x)-x\|_1\,]\end{aligned}$$

The motivation for these modifications is to add training stability and avoid issues common to GANs.

4.3.1 Data Preparation

To use as input to the CycleGAN, a large collection of generated LPs was generated by the process explained in section 4.1. The pipeline was interrupted after the initial LP had been conceived so the images were uniform in size, not augmented or superimposed onto a background.

The real LP images were extracted from the training portion of the SweLP dataset to serve as real samples for the discriminators. These LPs were often rotated and a bounding box that wasn't rotated couldn't encapsulate the LP without also capturing large parts of the car. Therefore, the images were rotated so that the LPs were straight before being cropped out. This was done with the help of a Python script.

The resulting images can be seen in 4.5.

Figure 4.5: SweLP images before rotation and cropping and after. The images to the right make up the real domain in the CycleGAN training.

4.3.2 Training

The CycleGAN was trained with the purpose of creating a generator that could transform the generated LP images to look more realistic. During execution time of the program, images from both domains were collected and randomly cropped to create smaller images of 128x128 pixels. This was done to save on GPU memory, in this case an Nvidia GTX 1070 8GB, which could handle a batch size of 8 with this setting. No further changes were made.

Training carried on for a total of 80 epochs which took about three days, saving the model after each epoch. Some of the produced images throughout the training can be seen in Figure 4.6. The generator responsible for transforming to the SweLP domain created vastly different images depending on how long it had been trained. Disappointingly, the images produced by one model looked similar to each other so no single model could be used to generate a great variety of images.

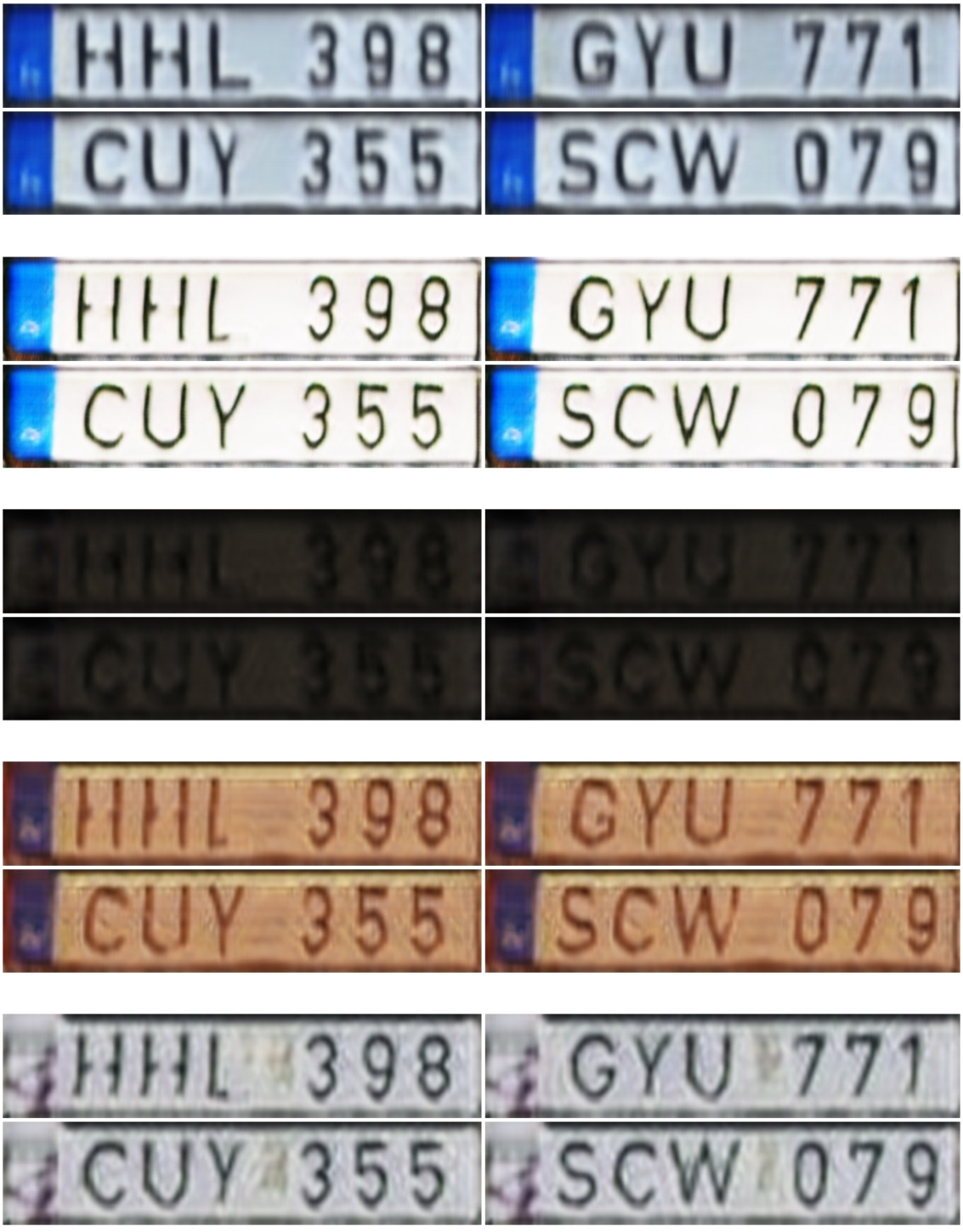


Figure 4.6: CycleGAN produced images from different epochs. The models suffer from mode collapse and produces similar looking plates. Most of the models not shown produce plates similar to the ones at the top. Amusingly, the very bottom plates have no blue area and a color spot between the letters and numbers. This is because there are some old Swedish LPs with that design in the real domain.

4.4 Evaluation

4.4.1 Metrics

To quantify how well a model performs the average precision from the Pascal VOC 2012 benchmark [23] was measured. This method relies on a few concepts.

Intersection over Union (IoU) measures the overlap of two areas divided by their union. The minimum IoU possible is 0, when the two areas don't overlap at all and the maximum IoU is 1, when they overlap perfectly.

Precision measures how accurate the predictions are as a percentage. It is calculated by dividing the number of correct predictions by the total number of predictions.

Recall measures how many of the ground truths that are identified. It is calculated by dividing the number of correctly found ground truths by the total number of ground truths.

F1 is often measured when it's important that a model perform well on both precision and recall. It is calculated through the harmonic mean of precision and recall, lowering the score significantly if one of the two is low.

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

$$Precision = \frac{\#\text{CorrectPredictions}}{\#\text{Predictions}}$$

$$Recall = \frac{\#\text{IdentifiedGroundTruths}}{\#\text{GroundTruths}}$$

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In the **PascalVOC benchmark**, one has to declare bounding boxes for the predictions and ground truths. A predicted box is considered correct if its $IoU >= 0.5$ for any ground truth box. If several predicted boxes overlap a ground truth box, one of the boxes is considered correct while the rest are incorrect.

The model has to provide each prediction box with a confidence score which conveys its certainty of the prediction. For every confidence $\hat{c}$ that exists in the predictions, the evaluation algorithm finds all predictions $\hat{p}$ with a confidence greater or equal to $\hat{c}$. $\hat{p}$ is checked against the ground truths to calculate precision and recall.

The values for precision and recall form a point on a graph with precision and recall as axes. When the algorithm spans every value of $\hat{c}$, more points are added to the graph that together form a curve.

Precision and recall typically counteracts each other i.e. a better result for one entail a worse result for the other. This leads to the precision-recall curve often having roughly the same shape as in Figure 4.7.

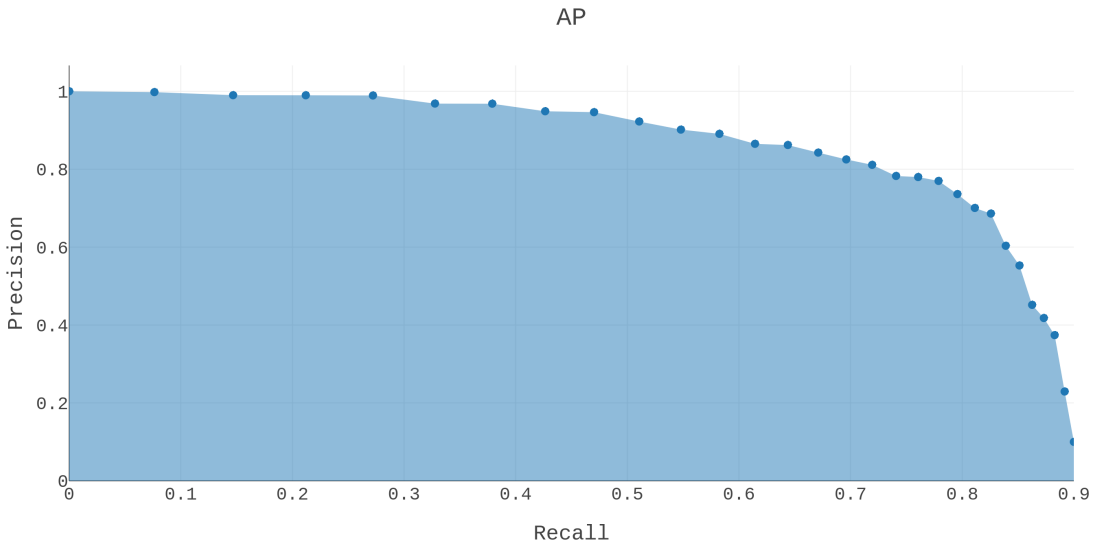


Figure 4.7: An example of a precision-recall curve. Average precision is the area under the curve.

Average Precision (AP) is the area under the precision-recall curve which is the metric that finally decides how well the model performed. It might go without saying but a higher AP score is better. AP punishes models which produce false predictions with high confidence. These errors will be incorporated early in the precision and recall calculations and thus affect multiple points - lowering the area. AP also punishes models that completely miss some ground truths.

4.4.2 Model Performance Score

The AP metric from PascalVOC is used to evaluate models on the SweLP validation- and test set.

The models are regularly validated against the SweLP validation dataset during training and the best AP score is saved which determine the **model performance for the validation set**.

This method of checking model performance multiple times is not allowed when measuring the score on the test set as the test set should be hidden until all training and parameter tuning has completed. Therefore the best model(s) should be saved during the training process and later used to assess test score. It was decided that three models were to be saved during training, when AP, F1 and loss are at their best. These three metrics are representative for a good detector and easy to calculate. **Whichever model that has the best AP score on the test dataset determines the test score**.

This elaborate system of saving several models was to hedge our bets against overfitting to the evaluation dataset. Imagine the case where only the AP model is saved. The model is evaluated against the validation set several times during training and one of those times it is *coincidentally* very good for that *particular* dataset but it might not generalize to the test set very well at all.

One might argue that the overfitting to the validation dataset wouldn't be as explicit if the F1 model was to be saved instead. The problem with only saving the F1 model is that different models need different confidence thresholds to function properly. Some models are overconfident and would make many incorrect predictions if the threshold is set too low, whilst others are underconfident and wouldn't make any predictions with a high threshold. The two models might perform equally given their optimal threshold but introducing an arbitrarily set confidence threshold for all models would create an unfair comparison since some models could be punished more than others.

A confidence threshold still needed to be decided so the program wouldn't run too slow during the post-processing computations. The predictions with very weak confidence scores rarely contain correct predictions and could be filtered out without any significant change in AP score. The threshold was set to 0.25 throughout the training, evaluation and testing for all models the same used in the original YOLO. The image resolution during evaluation and testing was set to 448 pixels in both height and width.

It was found that the models that were saved when AP was at its best consistently outperformed the others but for the cases where this isn't true the results will be subscripted with either F1 or Loss like so $0.85_{\text{F1 or Loss}}$ to indicate which model the results come from.

Chapter 5

Experiments

A series of experiments were conducted to investigate how synthetic data affect the training of a YOLO network learning to detect Swedish LPs. **All the AP scores presented are on the SweLP dataset**. Each experiment consists of a batch of sub-experiments or model trainings. The sub-experiments take approximately 9 hours each to complete on a Nvidia 1080-Ti GPU.

Overview of Experiments

- Baseline: Something to compare against and hopefully outperform
- Strictly synthetic data: Varying the dataset size
- Strictly synthetic data: Varying the number of background images
- Mixing images of LPs from around the world with synthetic data in different ratios
- Mixing low numbers of Swedish LP images with synthetic data
- CycleGAN images on their own and mixed with other datasets

All the experiments used the same following methods and hyperparameters unless otherwise specified.

The model started from YOLO weights which are trained on the COCO dataset and the first 52 layers (out of 75) had their weights frozen to save time and GPU memory. Anchor boxes were created via k-means from our SweLP training dataset.

ADAM [24], a popular optimization algorithm that utilizes adaptive momentum to update the model parameters was selected with a starting learning rate of 1^{-3}. A cosine learning rate annealing scheduler decreased the learning rate every 1000th step to 1^{-4} over the

course of the training. As in the original YOLO, weight decay [25], a regularization technique that punishes large weights, was set to 5^{-4}.

The predictions were post-processed by filtering out the boxes with a confidence score of less than 25% and non-maximal suppression [26] was used with a threshold of 10% IoU to further filter out overlapping predictions.

Batch size was set to 16 and each batch was randomly resized to a width and height of 352 to 544 pixels. No further data augmentation was used. All batches were resized to a size of 448 pixels during evaluation.

5.1 Baseline

By constructing a baseline, one has results to compare against which can support evaluation of the approach. **The baseline was made from Open Images**, the largest publicly available dataset of LPs. To remind the reader, the dataset consist of ~5400 training images of LPs from around the globe. YOLO was trained with the parameters mentioned above for a total of 10 times in an attempt to strengthen the certainty of the result. The baseline is formed by the mean of these experiments on the SweLP dataset and reported in Figure 5.1.

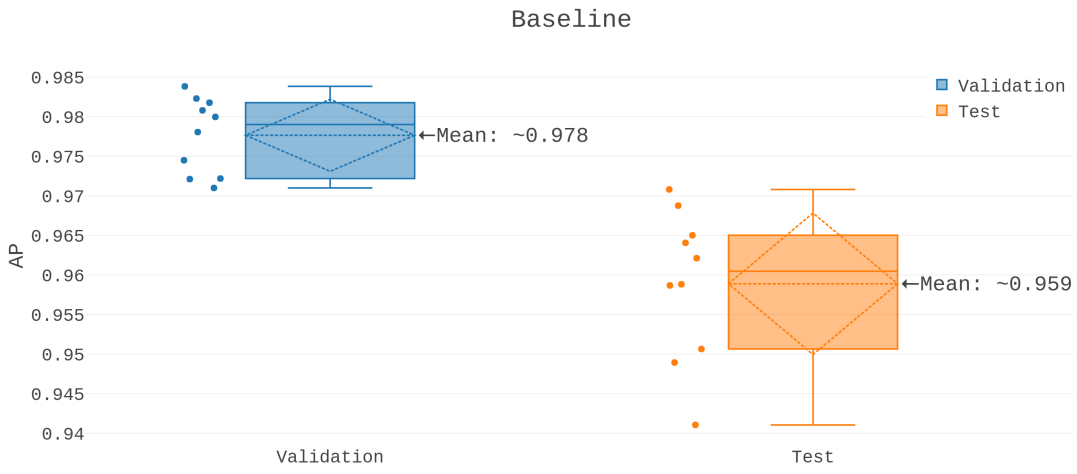


Figure 5.1: SweLP AP scores for a series of experiments trained on Open Images. Jagged lines shows mean and standard deviation of 1. Note that the validation score is (unsurprisingly) better than the test score.

5.2 Modifying ScriptLP Size

One central benefit of working with synthesized data is that it's possible to generate additional data at the push of a button. This experiment was setup to investigate how different amounts of synthesized ScriptLP data affect performance. A sufficiently large master dataset was generated and split into 11 datasets of different sizes. The smallest dataset contained 100 images and subsequent datasets were enlarged with a factor of two. Thus, the largest dataset contained $100 * 2^{11-1} = 102400$ images. The results of the experiment are presented in Figure 5.2.

The experiment shows that larger amounts of data give higher AP scores. That large amounts of data gives better results is common knowledge in the machine learning world and therefore the experiment results shouldn't come as a surprise. What's interesting is that the rather crude method of creating ScriptLP offers continued improvement up to 100 000 images and, as we will show later, beyond. It indicates that the synthesization process creates images varied from one another since 50 000 images produced this way didn't encapsulate all the knowledge in the process.

The ScriptLP datasets doesn't reach benchmark performance but is admittedly quite impressive.

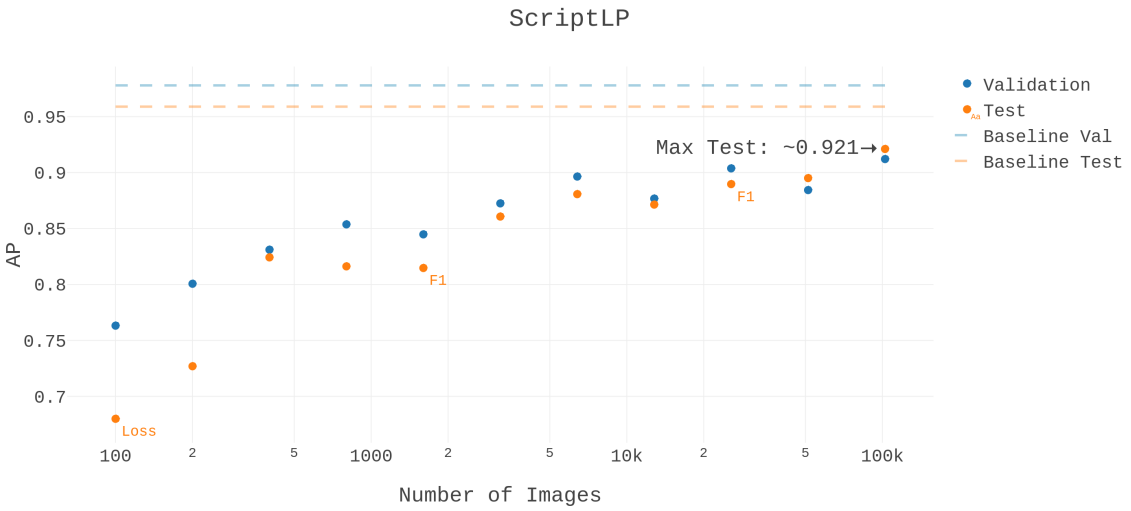


Figure 5.2: The figure shows different amounts of ScriptLP data. It clearly indicates that larger amounts of data give better performance. The performance increase seems to taper off for increasingly larger datasets, even more so considering the logarithmic x-axis.

5.3 ScriptLP - Background Images

As mentioned in 4.1 Data Generation by Superimposition, a subset of COCO images served as backgrounds for the superimposed images that form the ScriptLP datasets. If

one wants to generate a ScriptLP dataset bigger than the amount of available background images, the background images would have to be recycled. By reusing a background image, two resulting images will largely look the same considering the superimposed LP is small in comparison to the background image, see Figure 5.3.

Figure 5.3: Two images from ScriptLP with the same background

It is not clear whether including both of these images in the dataset helps the network to focus on what's important, the LP, or creates confusion. This experiments aims to give clarity in how reusing background images affect the results.

14 datasets were created, each consisting of 8192 generated ScriptLP images. The datasets used different amounts of background images calculated with the formula 2^{i-1} where i ranges from 1 to the number of datasets. The two datasets at the endpoints thus reused the same background images for every image (Figure 5.3) or never reused any background image. The results can be seen in Figure 5.4.

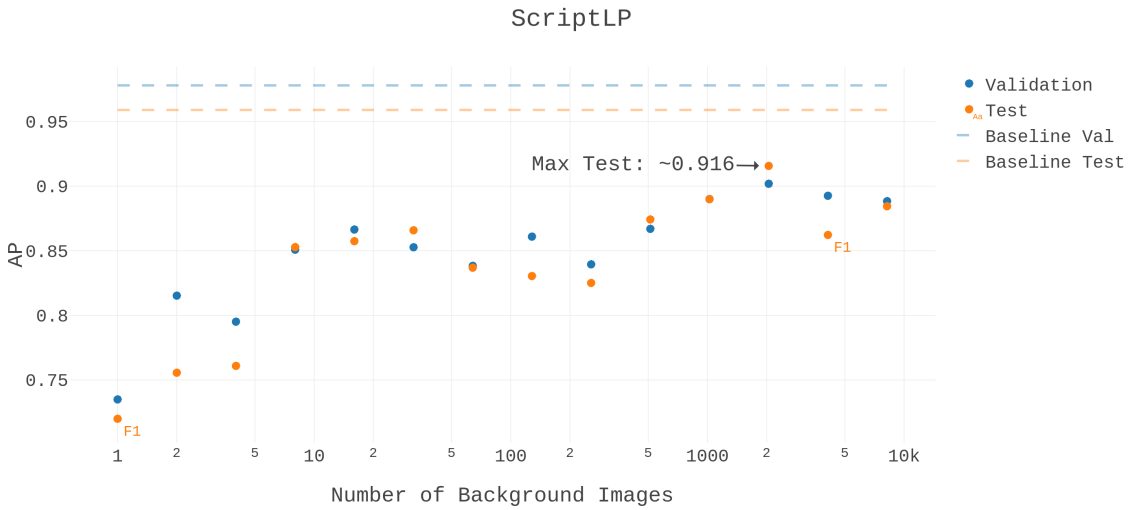


Figure 5.4: ScriptLP datasets created from different amounts of background images are compared against each other. Larger amounts of background images tend to give better results.

One can see that reusing a few background images for the whole dataset give poor performance. There is an oscillating pattern in the middle of the graph indicating that a larger set

of background images isn't always better. If this is true or due to noise isn't clear from this experiment alone. Taking a step back, the figure suggests that a larger set of background images tends to do better, albeit not very decisively.

5.4 Mixes of Open Images & ScriptLP

The previous experiments show that real data outperforms synthetic data of equal size and that more data is better than a little. This might spark the idea of augmenting all the real data with synthetic data to get the best of both worlds. One can theorize that mixing two different datasets could pull the model's parameters in two directions resulting in a terrible result. Or could mixing Open Images (cars without Swedish license plates) with synthetic data (Swedish license plates without cars) help the model to understand that it's supposed to find Swedish license plates on cars?

This experiment mixes the Open Images dataset with different amounts of ScriptLP data. 9 datasets with different ratios of real and synthetic data were put together this way. All of the datasets contains the full Open Images dataset and were extended with ScriptLP data to get the ratios of 1:0.01, 1:0.05, 1:0.1, 1:0.25, 1:0.5, 1:1, 1:2, 1:4, 1:8 real data to synthetic. Note that the 1:0.01 dataset doesn't contain 1% ScriptLP data but rather has one ScriptLP image for 100 real images from Open Images. The 1:1 dataset therefore has a 50% real to synthetic ratio. The results are presented in Figure 5.5.

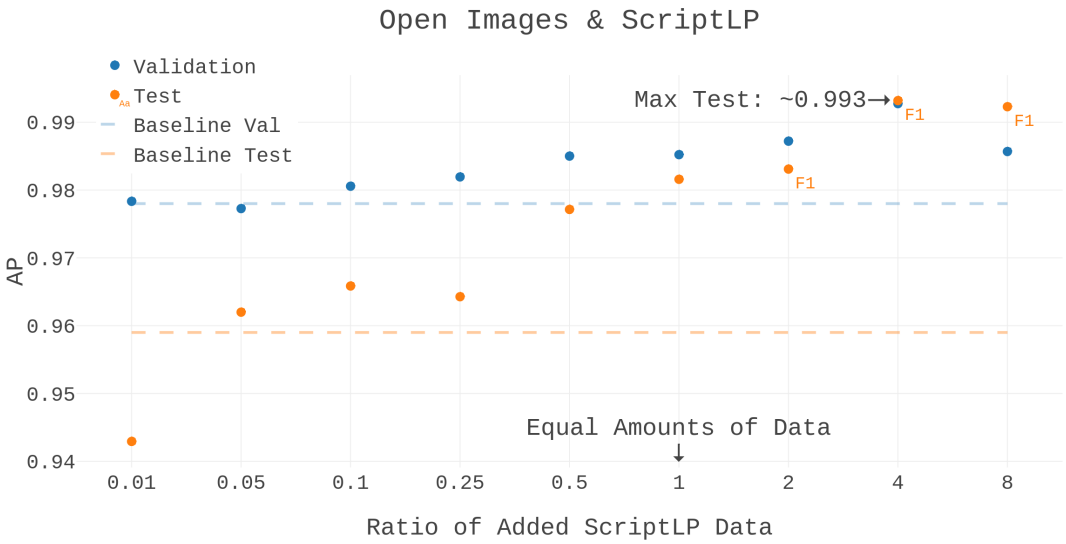


Figure 5.5: Different mixes of Open Images and ScriptLP data. Adding in ScriptLP data definitively increases performance. The F1 models seems to perform best for the datasets with high amounts of ScriptLP.

It is shown in the figure that mixing real and synthetic data give increases over baseline for both validation and test data. The best performing models have a large ratio of synthesized data which strengthens the usefulness of the method.

By comparing the error rate for the best model in this experiment to the best model in the baseline, it's seen that it has decreased from 2.9% to 0.7%, a reduction by more than 4 times.

Two precision-recall curves are presented in Figure 5.6 to further highlight the difference between a baseline model and a model made from mixed data. The mixed model is more reliable and inclusive, outperforming the baseline definitely.

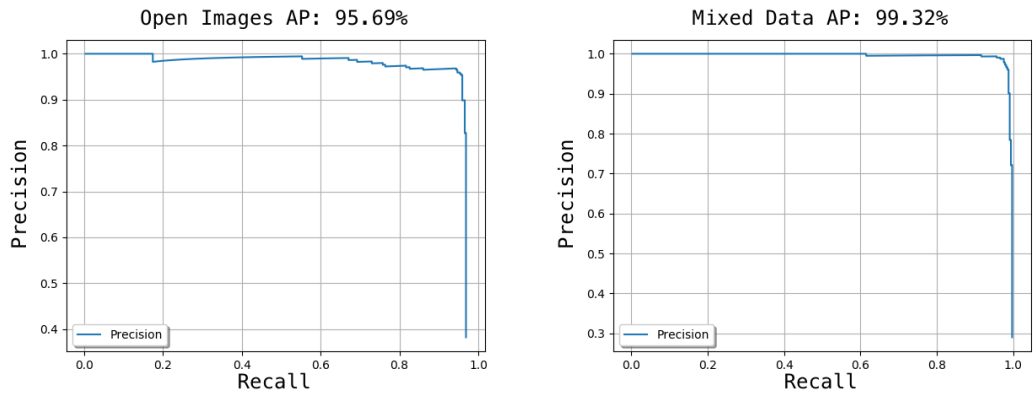


Figure 5.6: Precision-recall curves for a model trained on Open Images compared to one where Open Images is mixed with synthetic data.

It is worth mentioning that the datasets with a lot of synthesized data decrease the number of times the real data is presented to the model. This is due to the experiment running for a fixed number of steps and not epochs. As the real data is more valuable than synthetic data, this can be seen as a disadvantage to the models with a lot of synthetic data. Still, these models perform the best.

5.5 SweLP

The previous experiment investigated how different mixes of real and synthetic data behaved by adding different amounts of synthetic data. This experiment is similar but aims to explore how results are affected if the real amount of data is scarce. A common obstacle in machine learning projects is that there's little or no labeled data. If one can augment the existing real data with synthetic data and achieve good detection results, the tedious task of labeling data would be lessened.

A control experiment with strictly real data was done to serve as a comparison. Datasets containing different amounts of SweLP were created and compared against each other. Note that the SweLP dataset was used and **not Open Images** as in the previous experiment. Unsurprisingly, more data yields better results as seen in Figure 5.7. 761 images of Swedish LPs gave the same performance as the baseline on the test set where a larger amount of foreign LPs was used.

Then 8192 ScriptLP images were added to each of the datasets and the experiment was carried out once more. In hindsight 8192 synthetic data is quite a lot since the smallest

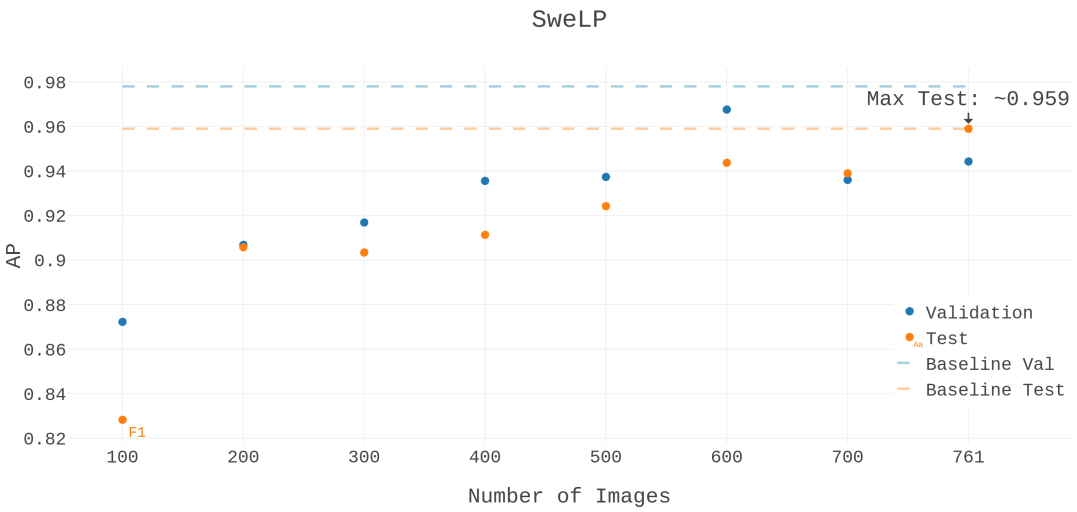


Figure 5.7: Higher amounts of SweLP data give better performance. 761 SweLP images give approximately the same results as 5400 Open Images pictures on the test data.

dataset only contains 100 real images. Nonetheless, this dataset showed a 12% absolute test performance increase for a total of 0.946 AP, higher than any synthetic dataset in isolation. The outcome of this experiment can be seen in Figure 5.8.

It seems advantageous to add synthetic data to low amounts of real data whilst this effect tapers off or even decreases performance as the number of real images increases.

5.6 CycleGAN

A series of experiments done on CycleGAN produced data was conducted to see if this type of data could further enhance results. CycleGAN was only used to produce "realistic" images of LPs out of context. Therefore, the augmentation and superimposition steps from 4.1 Data Generation by Superimposition were still carried out. However, the color augmentation step wasn't done, as this could destroy the realistic features the CycleGAN had already added.

5.6.1 Exploratory Experiment

It was mentioned in section 4.3 that several models from different epochs were saved and that these models produced LPs of a specific characteristics like blurry, dark or red tinted. Some of the models would surely perform better in the YOLO training than others, but it would be too time consuming to test them all.

A compromise was struck of testing 9 datasets against each other. The data was generated by using the model parameters corresponding to epoch number 10, 20, 30, ..., 80. The 9th

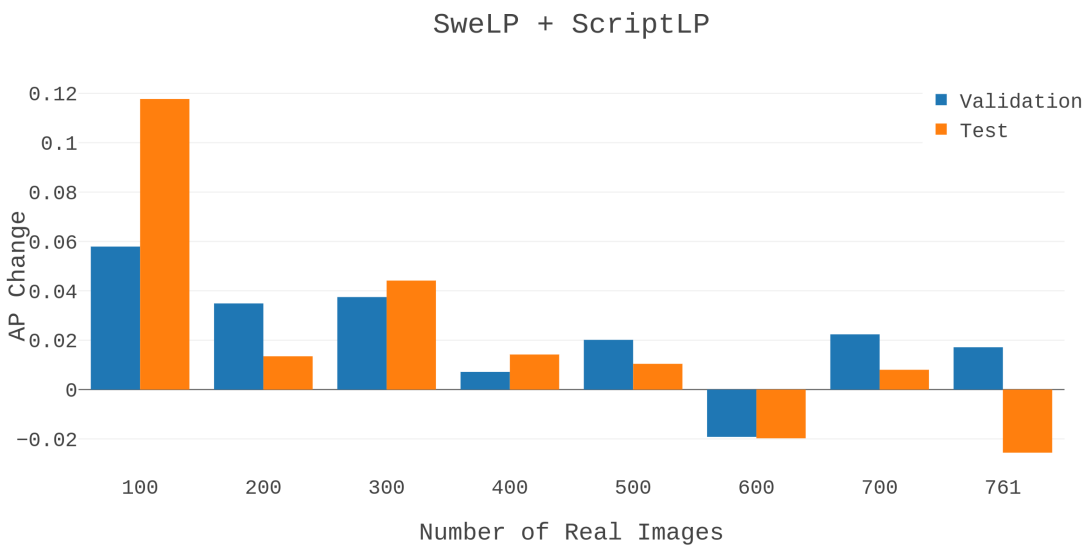


Figure 5.8: AP change in relation to the experiment showed in 5.7. Lower amounts of SweLP data enjoy the greatest increase from added synthetic data.

dataset was created by mixing the other datasets to see if merging varying kinds of LPs e.g. blurry, dark and red tinted would prove beneficial. This last dataset was made up from 1/8 of each of the other datasets so that it wouldn't be bigger.

Unfortunately the AP scores of the experiment won't be presented as the experiments were done at an earlier stage of the project where circumstances differed. Even so, the findings were that some epochs indeed gave better AP scores and the mix gave a slightly better AP score than any other model in isolation. Because of this, a mixed dataset consisting of 100 images from each epoch was created to get the maximum amount of variety. The dataset contain a total of 8000 images and was used in the rest of the CycleGAN experiments.

5.6.2 CycleGAN & Mixes

Five experiments on CycleGAN data were conducted to see how YOLO reacts. There wasn't any visible difference from training on ScriptLP except a slight performance drop. It doesn't seem like CycleGAN data is beneficial in isolation, but perhaps it could be mixed in with other datasets. The results from mixing CycleGAN with ScriptLP and Open Images in addition to pure CycleGAN data can be seen in Figure 5.9.

The CycleGAN and ScriptLP mix unfortunately also didn't show great promise as the performance was that of purely ScriptLP data. Mixing CycleGAN and Open Images did offer a substantial performance boost over baselines. The test score wasn't quite as good as ScriptLP mixed with Open Images but the validation score increased. In short, using CycleGAN this way did not prove beneficial.

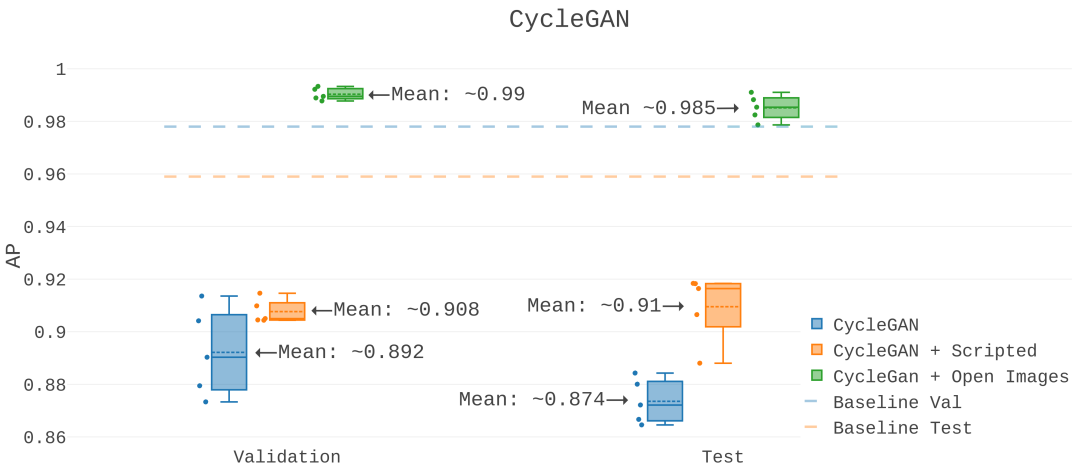


Figure 5.9: CycleGAN data compares slightly worse than ScriptLP. Mixing the two doesn't yield any increase over purely ScriptLP data. CycleGAN + Open Images outperforms everything previously seen on the validation data.

5.7 Buried Experiments

There were a great deal of experiments that were carried out, but deemed not interesting enough to elaborate on. This section will briefly summarize some of them.

Images with different **numbers of LPs** were synthesized and compared against each other. More LPs in an image gave worse precision scores during training but the same AP scores.

Adding **color augmentation** from section 4.1 to CycleGAN images gave strictly worse results, probably because the realistic features that were added by the CycleGAN were destroyed.

Backpropagation through the whole network was deemed unstable as the results sometime took a nosedive during training. Perhaps this could be remedied with a burn in period as done in the original YOLO, where they initially freeze the first layers in YOLO to decrease the risk of gradient explosion.

Training the **CycleGAN** on images with **different rotations** produced poor visual results where LP borders were jagged.

Different **loss weights** to see if one can train quicker or better. It was established that there wasn't a significant difference.

5.8 Summary

Some of the previous results will be aggregated to more easily compare them. One last experiment will be conducted and added to this comparison. This last experiment encapsulates all the knowledge from the previous experiments to produce the best result on purely synthetic data.

This was done by creating a ScriptLP dataset to a length of ~575 000 images utilizing ~80 000 COCO backgrounds. The mean AP score were 0.922 and 0.93 on the validation and test data respectively. Full results can be seen in Figure 5.10 along with other key results.

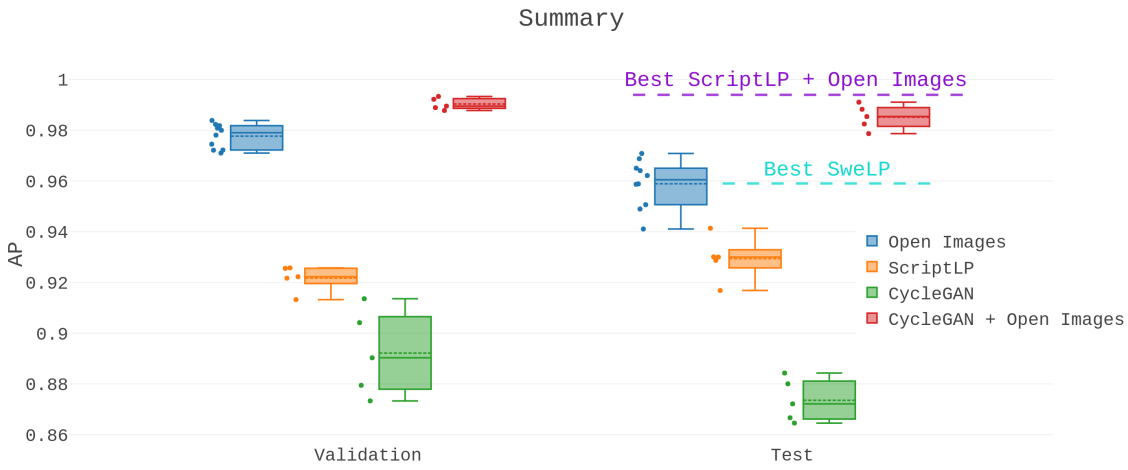


Figure 5.10: A comparison of many scores produced in the project. Synthetic data doesn't quite match up to real data but is rather close. The best results are achieved when mixing synthetic data with Open Images.

Chapter 6

Discussion and Conclusions

Mixing Real and Synthesized Data
To our delight, it turns out that the synthesized data is indeed helpful for finding Swedish LPs. Solely training on synthesized data gave impressive results in itself, although worse than baseline, but mixing it with Open Images produced models that exceeded baseline scores decidedly. As the models can't easily communicate why this mix is beneficial one can only reason about it.

We believe that the two image sources complement each other. The Open Images dataset teach the model to primarily locate cars which are often easier to find due to their larger size. Odds are that cars contain a LP between two wheels, under a windshield etc and the model learns these relationships whilst not explicitly looking for the LP more than any other common feature. This is unfortunately a problem when there is no LP on the car or if the location is anything but standard. The phenomenon is observed more explicitly by looking at some of the failure cases that a baseline model produced, see Figure 6.1. We have also noticed that these models tend to trigger many detections around the area where LPs are mounted, lacking the fine-grained knowledge of finding the exact position.

This argument is strengthened further by looking at the Figure 6.2 where a model trained on Open Images data gets worse at finding ScriptLP data as the training progresses.

Superimposed data help remedy this issue by forcing the model to also consider the actual LP and not solely base its predictions on the features given by the car. If the model is partly trained to find LPs without car features, the knowledge learned can help reduce false positives around the LP by more precisely target the right location.

On the other hand, this theory doesn't hold up if one considers the results from mixing SweLP and ScriptLP data. This experiment showed that there were no further gains by mixing in synthesized data once there was a sufficient amount of real data containing

Figure 6.1: Models trained on Open Images finds the LP by looking for cars. This becomes apparent on images where there is no LP.

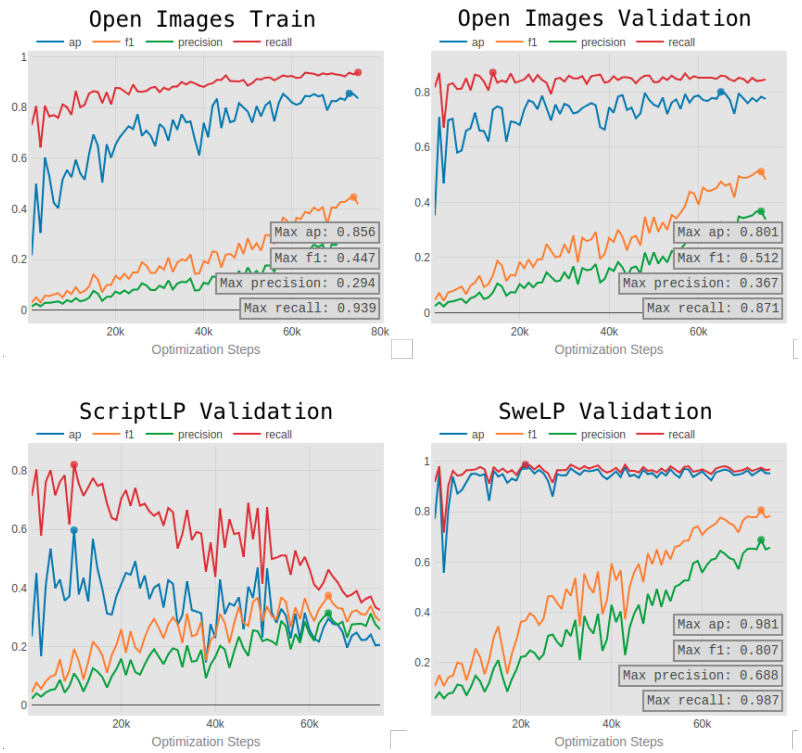


Figure 6.2: Metric scores for several datasets throughout training on Open Images. The model gets worse on ScriptLP data.

Swedish LPs. The increase seen by mixing Open Images and ScriptLP could therefore simply come from the introduction of more domain knowledge. By training on Open Images it's not clear to the model how a LP looks, there are tons of different types. If we narrow this domain by explicitly telling the model to search for Swedish LPs (blue area, same width to height ratio etc), we introduce new features that can be aggregated together with car features to more robustly find LPs.

Regardless whether these theories are correct or not, it can be concluded that regional LP detectors most likely will benefit from synthesized data if regional data is scarce or non-existent.

Synthesized Data
During this project both upsides and downsides revolving synthesized data has been recognized. Thankfully, tedious manual labeling isn't needed and once the generation pipeline is complete it's easy to create copious amounts of data. On the other hand, creating and calibrating a generation process isn't trivial and will likely require more effort than gathering real data. We saw that 600 SweLP images outperformed any kind and amount of synthesized data. These 600 images were gathered and labeled in less than a days work and could be done quicker by having better processes and programs. By comparison, weeks of programming went into creating the synthesizing pipelines.

One major concern that we have about the pipeline used to create ScriptLP datasets is that it requires domain expertise and many handcrafted features or arbitrary parameter selections. How much motion blur, color shift or rotation should we add to the LPs before superimposing them? Does our judgment of realism necessarily correlate to good results produced by YOLO? The only way to find out is by performing experiments, a painfully slow iterative process.

It is possible to eliminate many of these handcrafted features by analyzing real data but not necessarily effortless and straightforward. This is where CycleGAN comes into play. In theory it could help remove the need for domain expertise and reduce the amount of bias introduced into the synthesized datasets via a more data driven pipeline. However, this isn't what we found out in this project. The models we managed to train suffered so heavily from mode collapse that no model produced LP data less biased than our ScriptLP data. Furthermore, we noticed unwanted convolutional artifacts, looking like a checker board or cloth fabric, in the images produced by most CycleGAN models.

To not discourage future probing in the area we'd like to point out a few things. Many of the LPs CycleGAN created looked very realistic and could not easily have been produced through a scripted augmentation. This was the author's first practical experience with GANs and a larger understanding of the field would definitely help in selecting a model less prone to mode collapse and in getting it to work properly.

There are heaps of projects investigating the area of realistic image synthesization but where this thesis differentiates is that we use superimposition to generate data. The method is inherently disadvantaged when it comes to finding LPs in their natural setting as there are no car features that can help. This is especially true when the LPs are very small, perhaps only a few pixels. This is even more true if you consider that the images are often downsized before being processed by YOLO. Although we believe that our results can be

improved upon by creating more realistic looking LPs, it's not clear how big these gains would be considering the image re-sampling before YOLO.

If the goal is to train a useful detector, our recommendations are to first try to gather real data as that seems like the method that takes the least amount of effort. A mere 600 images, a days work, outperformed our synthetic data. With that said, the potential of synthesized data is still unknown. A team of experts could very well produce better synthesized data, possibly outperforming real data altogether.

If real data is scarce or raises other concerns (e.g. privacy concerns) there is definitely room for synthesized data. Even in the cases where real data is abundant, there might be merit to synthesizing data that can highlight certain aspects. It's notoriously hard to understand and control the inner workings of deep learning models so perhaps controlling the data could help us guide the models to where we want them to go.

Chapter 7

Future work

Detection Through a Proxy Class
There are a lot of annotated images of cars available but the LP annotation is often missing. For example, in Open Images there are ~90000 images of cars but only ~5400 of LPs. Since the classes are so correlated, would it be possible to leverage the large amount of car data to better find LPs?

The best of our models were trained on mixed Open Images and ScriptLP data. We theorized that superimposed data helped to both reduce false positives in the approximate LP position and more precisely pinpoint the right location. Models purely trained on Open Images didn't seem to pay too much attention to the LP but rather made its predictions based on car features. Having YOLO more robustly detect car features could therefore help performance.

By explicitly training the detector on the proxy task of finding cars we hope to develop better car features to aid detecting LPs. A detector capable of finding two classes, car and LP, is needed. The large set of car data is aggregated with LP data and used to train this detector.

It's important to not penalize the detector if it spots an unlabeled LP within the car images. This can be implemented in many ways and one way would be to only incur loss for LP predictions if the image comes from the LP data. Clearly the same reasoning would be applied to cars.

We don't believe that we are the first to suggest a method of finding LPs by searching for cars. One can think of a system that works by finding a car, cropping the region around it and proceeds to further analyze the cropped image to find the LP. Our proposed method wouldn't need two stages, saving on complexity and computational resources.

It's also possible that the approach could generalize to other areas. If one is interested in finding objects of one class that almost always is spatially correlated to another class, that other class could be brought in to the training process and serve as a proxy objective.

Another suitable set of classes would be people and faces. Faces are always attached to peoples shoulders and if we want to detect faces it might be easier to learn the proxy task of detecting bodies.

Superimpose Other Objects

Future work could explore superimposition of other objects. We suspect that one gets better results if the object is easy to model realistically and not too small if there normally is important context. Perhaps road signs (or other types of signs) could lend itself well to this purpose.

Realistic Generation & Superimposition

A clear cut continuation of this thesis would be to more realistically synthesize the LPs. More meticulous crafting of the LP conception, where every detail of the LP matches the original might help. Using a different model or training the CycleGAN better to introduce realistic features without mode collapse could be another exploratory path.

And perhaps something could be done to improve the superimposition technique. Does the background choice play any difference? Can we find more appropriate locations for the LPs or somewhat blend them into the background to make the superimposition less obvious? We present some impressive work done by Fujun Luan et al [27] in their paper Deep Painterly Harmonization, see Figure 7.1.

Figure 7.1: Superimposing an image onto a background to fit the style of the background

Bibliography

[1] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *CoRR* abs/1812.04948 (2018). arXiv: 1812.04948. URL: http://arxiv.org/abs/1812.04948.

[2] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. (2019).

[3] Alec Radford et al. "Language models are unsupervised multitask learners". In: *URL https://openai. com/blog/better-language-models* (2019).

[4] *DATASETS OVER ALGORITHMS*. 2016. URL: http://www.spacemachine.net/views/2016/3/datasets-over-algorithms (visited on 03/20/2019).

[5] *Police Automatic Number Plate Recognition*. 2014. URL: https://www.youtube.com/watch?v=W9ERlIFjEac (visited on 03/20/2019).

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

[7] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[8] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] Alina Kuznetsova et al. "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale". In: *arXiv preprint arXiv:1811.00982* (2018).

[10] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[11] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[12] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[13] Ayoosh Kathuria. *What’s new in YOLO v3?* 2018. URL: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b (visited on 03/12/2019).

[14] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[15] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[16] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[17] A. Gupta, A. Vedaldi, and A. Zisserman. “Synthetic Data for Text Localisation in Natural Images”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[18] Hui Li, Peng Wang, and Chunhua Shen. “Towards End-to-End Car License Plates Detection and Recognition with Deep Neural Networks”. In: *CoRR* abs/1709.08828 (2017). arXiv: 1709.08828. URL: http://arxiv.org/abs/1709.08828.

[19] Xinlong Wang et al. “Adversarial generation of training examples: applications to moving vehicle license plate recognition”. In: *arXiv preprint arXiv:1707.03124* (2017).

[20] *DIN 1451 font*. 2019. URL: https://en.wikipedia.org/wiki/DIN_1451 (visited on 04/14/2019).

[21] *Image Augmentation with Imgaug*. 2019. URL: https://imgaug.readthedocs.io/en/latest/ (visited on 04/14/2019).

[22] *PyTorch CycleGAN Github*. 2018. URL: https://github.com/aitorzip/PyTorch-CycleGAN (visited on 04/13/2019).

[23] Mark Everingham et al. “The pascal visual object classes challenge: A retrospective”. In: *International journal of computer vision* 111.1 (2015), pp. 98–136.

[24] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[25] Anders Krogh and John A. Hertz. “A Simple Weight Decay Can Improve Generalization”. In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. NIPS’91. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1991, pp. 950–957. ISBN: 1-55860-222-4. URL: http://dl.acm.org/citation.cfm?id=2986916.2987033.

[26] Alexander Neubeck and Luc Van Gool. “Efficient non-maximum suppression”. In: *18th International Conference on Pattern Recognition (ICPR’06)*. Vol. 3. IEEE. 2006, pp. 850–855.

[27] Fujun Luan et al. “Deep painterly harmonization”. In: *Computer Graphics Forum*. Vol. 37. 4. Wiley Online Library. 2018, pp. 95–106.

Master's Theses in Mathematical Sciences 2019:E18
ISSN 1404-6342

LUTFMA-3380-2019

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/