# Litter-Paste

**Investigating the impact of context and style transfer on image synthesis for deep learning dashcam litter detection**

UNIVERSITY OF
LINCOLN

Oakleigh Weekes
WEE18678532

18678532@lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of MSc Intelligent Vision

*Supervisor* Dr. Wenting Duan

September 2023

# Acknowledgements

This project has taken place with great thanks to the Laboratory of Vision Engineering at the University of Lincoln. Thank you to Dr Wenting Duan for supporting the dashcam litter detection project which has not only helped my research reach a possibly publishable standard, but has encouraged me to pursue and secure and a PhD position at the University of Sheffield in the related field of small object tracking. Many thanks to Dr Xujiong Ye, who has also taken part in the hiring of myself as a Research Assistant, and has also co-hosted showcases and reading groups, which have been invaluable for networking and knowledge development. I also would like to send gratitude to Dr James Brown and Dr Petra Bosilj, who I have worked with on projects at Bachelors level, and have helped grow my seed of interest in Computer Vision. Finally, I would like to praise all members of LoVE, including the PhD students who have been exceptional educational and emotional support during my time at the University of Lincoln.

# Abstract

The roadside litter problem bears a negative financial impact for local councils but also delivers an environmental impact to natural surroundings. An artificially intelligent detection solution which surveys dashcam footage for litter could be an efficient way of quantifying such a problem, as well as seeding action strategies. There are a growing number of openly available litter image datasets which are crowd-sourced to help tackle the issue of waste with deep learning, but they exist with specific context to their areas of capture. A data synthesis software tool was produced which creates artificial dashcam frames using real video footage and an open-source litter dataset. Artificial litter was placed in a different context within road imagery for each created dataset. Native litter style was applied to imported litter samples using AdaIN neural style transfer. After training a YOLOv8L model with such datasets, it was found that placing external style-transferred litter in locations seen in training data increased recall by 3.3%. Blending litter into never-before-seen background frames increased precision by 0.5% and average precision by 0.3% without a substantial impact on other performance metrics. Simply adding the open-source litter dataset to training increased precision by 2.1%, but came with a decrease in recall and average precision. These beneficial improvements came about with an addition of less than 2,783 synthetic images to the base dash-cam frame "DashLit" dataset.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Aim, Objectives and Hypothesis

The aim of this project is to *investigate the effects of contextual placement and style transfer on copy-paste data synthesis for deep learning dashcam litter detection* and to *create a software solution that generates synthetic litter frame datasets*. The hypothesis is that training a neural network with external litter dataset items pasted within original dashcam frames improves either the precision or recall of the litter detection solution. In order to achieve the aims, the project is dissected into achievable objectives:

- Organise annotated dashcam litter frames into suitable training, validation and testing categories.
- Train & fine-tune a baseline YOLOv8 (Jocher, Chaurasia and Qiu, 2023) object detection model using organised original dashcam litter data.
- Create a solution which can copy litter samples out of the "TACO" (Proença and Simões, 2020) external litter dataset using their masks and paste onto dashcam frames.
- Investigate methods of adding context to pastes using training data and semantically segmented frame regions.
- Discover a method of transfering style from the local dashcam dataset litter samples to the imported TACO samples.

- Create a python interface which can generate new dashcam litter samples using all researched methods.
- Compare the precision and recall of the YOLOv8 detector when trained with generated datasets of different contextual paste methods and optional style transfer.
- Generate a dataset using unseen external dashcam frames with the external litter samples placed using the most accurate placement method. Compare the detection accuracy.
- Investigate the impact of litter placement blending on the best dataset creation strategy by comparing detection performance on such data.

## 1.2 Background

### 1.2.1 The Roadside Litter Problem

Expenditure on UK roads has increased steadily since 2014 (GOV.UK & HM Treasury, 2023). Although this may be slowly eroding private grasslands, it is growing a new hybrid habitat - the roadside "green" verge. These roadside verges could act as a means of nature conservation, carbon sequestration, air quality improvement and flood reduction (Phillips et al., 2021) - increasing the importance of safeguarding these developments. Deep learning research has helped to identify particular verges with high conservation potential by visually analyzing flora and greenery (Perrett et al., 2023).

Littering is the deposition of waste in our environment, and can occur illegally by human misconduct, or naturally due to environmental factors. When litter does not reach the appropriate collection facilities, such as bins, it can end up on city streets, roads, countrysides and natural waters. Litter can migrate to roadsides due to both wind and human behaviour (Karimi and Faghri, 2022). Concerning the impact of roadside pollution, run-off has a potential to contaminate local waters with inorganic materials (Philippe et al., 2023), clog drainage systems (Elzarka,

Buchberger and Meduri, 2020) and has negative effects on verge-residing wildlife (Environment Times, 2022). Littering as a whole has a monetary impact in costing the UK Government approximately a billion pounds yearly in clean-up efforts, but roadside litter surveillance and collection also adds an extra risk to life on dangerous highways (GOV.UK, 2022).

There currently exists no automated monitoring of road verge health in the UK, with councils (GOV.UK, 2023) and National Highways (National Highways, 2023) reliant on subjective reports by the general public on dense areas of verge littering. Automated litter detection could be a first step in quantifying such pollution, in order to focus surveillance on certain road regions or act as a flagging mechanism for waste clean-up efforts. Dash-cams are cheap and widely available recording devices (Adamová, 2020), which could allow for passive litter detection when incorporated into public-service vehicles, and would allow for objective monitoring strategies. These could possibly include mapping for increased surveillance which could inform collection in certain regions.

### 1.2.2 Detection using Computer Vision

Object detection in the field of computer vision exists as a means of automated localisation and recognition within images, and could potentially work with the passive dash-cam strategy on a frame-by-frame basis to find rubbish. Presently, object detection commonly involves training neural networks (M. Zhou, 2022) - deep learning systems which can extract information from images and pinpoint items of interest. These networks often require training with large datasets (Mahony et al., 2020). Collection involves gathering many images of objects of concern, and annotating them by "drawing" a bounding box around said objects. The network sees images as a matrix of pixels, and these boxes as coordinate and measurement information.

Before data is searched for containing such pieces of rubbish, an important question must be asked- "What *is* rubbish?". It becomes a philosophical question, in that “What is rubbish in one context is perfectly useful in another" (Hawkins, 2006, p.10). To establish context for a roadside litter detection solution, litter will be classed as

"any inanimate item that does not belong at the roadside", and will include the likes of small items such as discarded packaging, paper and bottles, and larger items such as car debris and tires. Temporary road signs, construction infrastructures and sandbags will be excluded.

There has been effort to collect varied litter datasets in different locations, and for dashcam litter detection, the Laboratory of Vision Engineering at the University of Lincoln, 2023 has collected a domain-specific collection of video frames. Although efforts have been made in unifying and cleaning up various external public collections (Majchrowska et al., 2022), little has been done to process these sets for use across domain. Whereas most public litter datasets are captured up close to objects with high-quality mobile cameras (see 2.1.1), dashcam data is often at a lower resolution and quality, partially due to motion of the vehicle taking the footage. It could be of use to harness the variety in highly-defined litter seen in other contexts for training the dashcam detection solution.

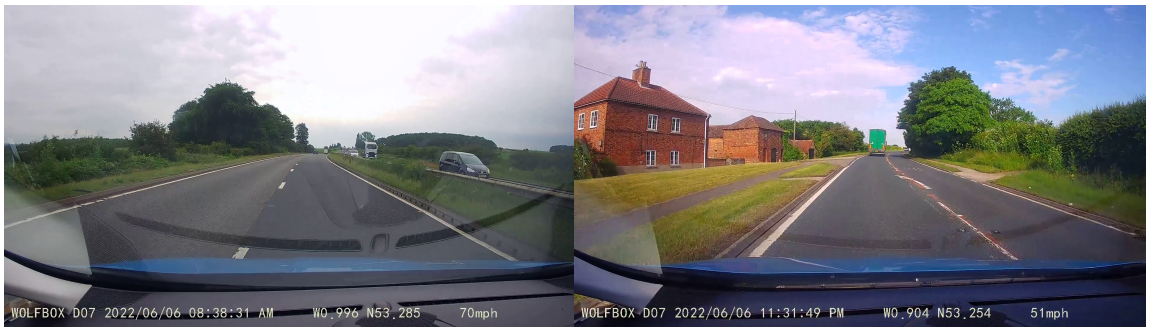


Figure 1.1: Example training frames from the LoVE - supplied "DashLit" dataset.

### 1.2.3 Image Augmentation & Synthesis

Image augmentation and image synthesis are two similar concepts that lead to producing more information in images for a deep learning vision network to learn.

Image augmentation involves using existing images to create further variation in the data. This often involves operations such as rotation, translation, scaling, flipping, contrast adjustment, noise injection, cropping and resizing (Shorten and Khoshgoftaar, 2019).

Image synthesis involves generating new images, often by using other sources. It has increased in popularity due to advancements in generative adversarial networks, which utilise a generator network to "fool" a discriminator network into recognising noise-transformed samples as an authentic image category (Creswell et al., 2018). Neural Style transfer is a related method, whereby a network learns to capture a datasets "uniqueness" in terms of "style". This "style" can be transferred onto new datasets. Style and content are defined as general feature representations learned at different layers of the network (Jing et al., 2020).

Both augmentation and synthesis can be seen as domain generalisation methods when they occur across datasets, whereby features are given more weight if they exist in all environments. Technically, both techniques can reduce a phenomena known as "overfitting" - whereby the model performs well (high accuracy metrics, reduced loss) on the dataset it is trained with, but poorly on "test" or unseen data it has not been trained on (Shorten and Khoshgoftaar, 2019). Increasing the diversity in training samples from limited data sources can help an algorithm perform more robustly to unseen data in a real-world scenarios of a stochastic nature.

### Possible advantages in terms of dashcam litter dataset processing

The collection of dashcam litter samples is difficult for a number of reasons. Manual annotation is time-consuming, which reduces hours that could be spent in other research and development areas such as fine-tuning models or testing deployments. Video annotation in particular involves annotating frame-by-frame, and even bounding box interpolation between frames fails in software such as CVAT (Sekachev et al., 2020) due to the changes in constant speed of the vehicle, the natural verticle sway, changes in appearance due to occlusion, motion blur, and distance from object. Manual annotation is also prone to human error, due to the monotony of such frame-by-frame inspection but also dealing with small objects - those that can be both easily missed (due to appearance or frame-skip) and easily miss-annotated (by bounding objects such as flowers, or forgetting to zoom in accurately). If image augmentation or synthesis techniques are successful, they could negate most or all of these problems by automating the process. Furthermore in terms of model per-

formance, augmentation is known to increase the accuracy of small object detection, not only by producing more samples, but also enlargening samples such that shape and texture class features can be more accurately represented, and thereby learned (Cheng et al., 2023).

## 1.3 Report Structure

The project report structure is as follows. Firstly, there will be a literature review. Previous experiments motivate the deep learning approach to litter detection, and this will be displayed along with their dataset descriptions. Then there will be a look into the currently used techniques for data synthesis, with a focus on composition, as it can produce new datasets simply by cutting out objects in one and adding to the target set. Context is a variable for such synthesis, and so quantifying it will be looked into, as well as how research has applied style cross dataset domains for increasing accuracies.

The methodology is the most substantial section of this report. It goes through project management, in how time was regulated to ensure both research and software elements were developed suitably. Experimental design looks into what research questions were established, and what requirements are necessary to answer them. Then the acquired datasets are described, as they remain specific to litter or dashcam domains. The contextual options are described for litter placement, in particular segmentation, with methods focused on low-annotation time. The model for style transfer is described, as this is necessary to train before being implemented in the composition pipeline. The object detector network is then detailed, and how its particulars make it a great choice for litter detection, and what metrics are used to quantify its performance. There is also a brief look into a possible blending strategy after litter placement.

The methodology continues into how the research was enacted, firstly by examining the best method of segmentation for the "region of interest" (ROI) contextual litter placement method. Next, the main experiment procedure is detailed in how datasets were synthesised, and how the object detection framework was run.

Next is a look into the software development aspect. The solution was required to be planned which produced such synthesised datasets. Design documentation is presented, and requirements are broken down further into what necessary software and hardware items were necessary. Implementation describes how the various software modules were designed piece-by-piece and how they come together to form a program that accepts options to generate new images from supplied datasets. A brief testing section describes how it was known that the software produced sound samples.

The following results are that of the detector when trained with baselines and the generated datasets, and include synthesised object sample images, loss graphs, and the outcome detection performance metrics. These are evaluated in the following discussion chapter, and a reflection details potential improvements to both software and research workflow. Finally, a conclusion summarises the key insights this project has unearthed along with future work which could further benefit the field of litter detection.

# Chapter 2

# Literature Review

## 2.1 Litter Detection using Deep Learning

### 2.1.1 Datasets

There have been some examples of dataset collection of litter image data over a range of contexts and environments. Some recent datasets are as follows:

- UAVaste (Kraft et al., 2021) contains litter samples on grasslands in the context of a birds-eye-view from aerial drone cameras.
- MJU-Waste (T. Wang et al., 2020) contains images with a depth channel containing human-held objects of one class "litter".
- Drinking-Waste (Serezhkin, 2020) contains includes 4 classes solely consisting of bottles and cans.
- TrashNet (Thung and Yang, 2016) contains six classes of litter with a purposed white-background.
- Wade-AI ('Let's Do it' Foundation, 2016) contains images of litter in the context of Google Streetview. This project birthed "Trash-Ai" (Fey et al., 2022), a litter auto-annotation tool which works best with photos taken one metre away from the item.
- Trash-ICRA19 (Fulton et al., 2018) and TrashCan (Hong, Fulton and Sattar, 2020) are two datasets collected of underwater litter and debris.

- PlastOPol (Córdova et al., 2022) contains varied size rubbish of the single class "litter" in its images.
- The Litterati (Kirschner, Ayres and Doherty, 2012) online platform aims to collect labeled litter data on a global scale, and submissions are logged as a pin on an interactive map. OpenLitterMap (Lynch, 2018) has a similar premise, but rewards submissions with the blockchain incentive "littercoin".

The TACO "*Trash Annotations in Context*" (Proença and Simões, 2020) dataset is commonly used in the literature, as it attempts to establish an open and crowd-sourced collection of litter data in various environments. The core dataset contains 1500 images with segmentation-mask annotations spanning 28 main categories with 60 sub-categories, in the widely-used COCO format. Environments include backgrounds of other litter, vegetation, sand, water, indoor locations and pavements. It has grown into what seems to be known as the "extended-TACO" dataset, including "unofficial" publically-drawn annotations on new images.

The Detect-Waste benchmark (Majchrowska et al., 2022) was created in a pursuit to pool waste datasets into a uniform annotation format with unambiguous categories, filtered for annotation error and labeling consistency. It corrects errors and biases seen in the likes of the extended-TACO dataset and also includes samples from UAVVaste, TrashCan, TrashICRA, MJU-Waste, drink-waste, Wade-AI, and TrashNet, along with images from a miscellaneous waste pictures and Google Images.

### 2.1.2 Published Results

Projects using previously mentioned datasets have begun to achieve commendable results using neural object detection networks. The best results are seen in table 2.1, and seem to show that YOLO (Redmon et al., 2015) and EfficientDet (Tan, Pang and Le, 2020) networks are capable of the litter detection task. Included is also work from W. Zhou et al. (2023) , which attempts to reduce the amount of litter data needed to train an object detection network using few-shot learning. Few-shot involves a network learning meta-knowledge from prior training with a large dataset such as MS-COCO (Lin, Maire et al., 2014), then appending this knowledge from

training with few specific samples such as litter. The network is based on Faster R-CNN (Ren et al., 2017), and is capable of sub-classifying localized litter samples. The network contains a novel feature-fusion mechanism which highlights regions which may contain waste. Unlike the other experiments, results for that study are presented as mean average precision over all IoU thresholds, as opposed to the common 0.5 threshold [AP is described in detail in 3.14].

Table 2.1: Table showing best results from deep learning litter detection in literature.

| Source | Dataset | Best Model | mAP @ .5 IoU | mAP |
|---|---|---|---|---|
| Kraft et al. (2021) | UAVVaste | YOLOv4 | 78.5 | |
| Majchrowska et al. (2022) | Detect-Waste | EfficientNet-B2 | 65.5 | |
| Córdova et al. (2022) | TACO | YOLOv5x | 63.3 | |
| Córdova et al. (2022) | PlastoPol | YOLOv5x | 84.9 | |
| W. Zhou et al. (2023) | TACO | Few-Shot | - | 31.16 |

### 2.1.3 Other Anti-Litter Solutions

Litter-tackling research does not only contain that of litter detection. Smart cleaning machines have been constructed which vaccuum and sweep (Ramalingam et al., 2021) and contain purpose build grippers (Almanzor et al., 2022) to aid in clean-up efforts. To act as a littering deterrent, multi-tasked learning has been employed to detect littering as it happens, combining pose estimation to aid imposing fine action (Bae et al., 2020). It is seen that work in litter detection research benefits such downstream ventures.

## 2.2 Advances in Data Synthesis

### 2.2.1 Overview

Supervised deep learning models in the world of computer vision often require large amounts of data to train, which is not only costly in terms of storage and computation, but also in terms of manual collection, preprocessing and annotation time. This leads to the concept of data synthesis; manufacturing new images that reduces such

overheads and can adapt currently available datasets to generate new content, that can perhaps better suit the specific context of the task domain. In recent years there has been an idealogical shift to prioritising data quality over data quantity, which can be measured by the variety, integrity and distribution of samples (Nowruzi et al., 2019). These measurements could be boosted with such synthesis. For the case of discussion, data augmentation will be included as it is often used within such "synthset" creation.

Paulin and Ivašić-Kos (2023) reviews the popular methods of dataset synthesis used specifically in computer vision. Techniques include creating content digitally, using computer game, physics or simulation engines, generative adversarial training and compositing. In terms of object detection, it has been found that when creating images for use in such architectures, it is important for a synthetic image to contain similar features to that of the original "real" data (Rozantsev, Lepetit and Fua, 2015). It has also been found that a synthesised dataset should share a similar distribution of certain features, such as adding too many synthesised "rainy" images can negatively impact training while adding a few can boost segmentation accuracy of a relative model (Veeravasarapu et al., 2015).

### 2.2.2 The "Copy-Paste" Strategy

Of particular interest is the compositing method, whereby images can be seen as containing two layers, a foreground and a background. The strategy takes 2D objects from one set, and overlays them onto a picture's foreground to create a 'new' image. Variance can be added via intermediate processing, such as partially occluding objects, and objects can be blended to aid photo-realism. Generative technologies can be used to create new objects before pasting. Logging placement location can act as an automated annotation method- greatly saving time resources. Paulin and Ivašić-Kos (2023) has created a concise summary of a maximal composition pipeline seen in fig 2.1.

Significant research by Dwibedi, Misra and Hebert (2017) laid some of the groundwork regarding the technique of the coined "copy-paste" image synthesis technique

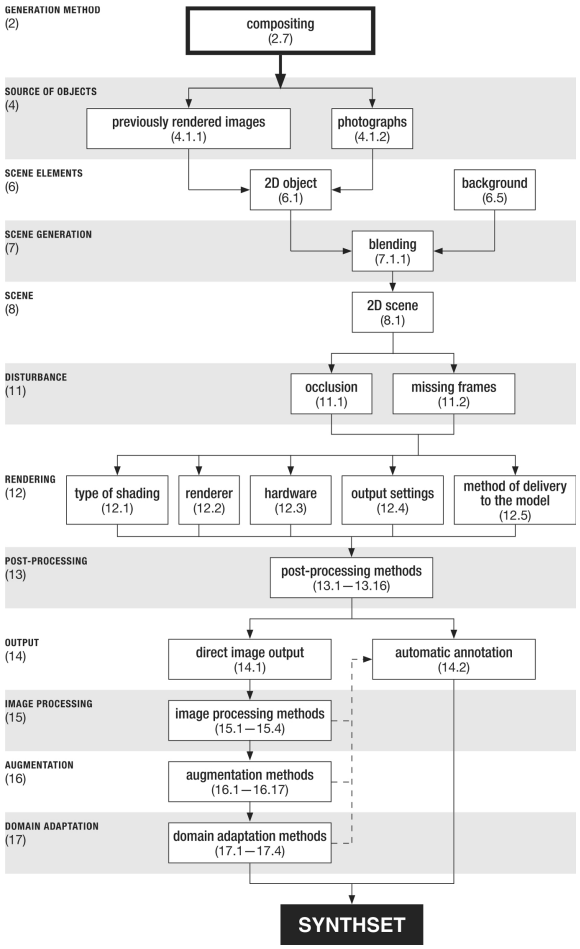


Figure 2.1: Compositing-synthesis pipeline (Paulin and Ivašić-Kos, 2023)

for instance detection. They used objects from the BigBIRD dataset (Singh et al., 2014) of 600 images and pasted them onto the 1548 images in the UW Scenes (Lai et al., 2011) dataset. A Faster-RCNN (Ren et al., 2017) model was then trained with these and evaluated on the GMU Kitchen dataset (Georgakis et al., 2016). Prior to pasting, objects were cut using masks obtained from training a segmentation model on out-of-sample images. Objects were augmented using 2D rotations, occlusions, and 3D rotation, the latter being allowed due to the 3D nature of the source object dataset. Distractor objects were also placed from the source dataset which were not of the classes to be detected. Various blending strategies were applied to integrate the objects into the images and remove so-called "boundary" artefacts which degraded detection performance. It was found that their synthetically-created data combined with only 10% of real data outperforms models trained on all real data.

With regards to placement of such items, research suggests copy-pasting items for

instance segmentation in random locations is a feasible technique for increasing model performance (Ghiasi et al., 2021). Fang et al. (2019) suggests using probability maps to guide pasting in their "InstaBoost" method of increasing instance segmentation model mean average precision. Dvornik, Mairal and Schmid (2018) goes further as to model visual context for pastes as a learning task - where inputs to a placement model are annotated boxes of reasonable placement. Their experiments showed that context copy-paste increased average precision per class by 4%. However, this may introduce further annotation, and would negate one of the benefits of such a synthesis technique.

Other discoveries have found that the copy-paste strategy leads to more robust detection in occasions of overlapping items in crowded scenarios (Jiangfan Deng et al., 2023). Such strategies have been introduced to object tracking pipelines as "continuous copy-paste", which boosts performance as a data augmentation method (Xu et al., 2021). In agriculture, the copy-paste method has been used with denoising diffusion probabilistic models; generating whitefly samples to paste onto backgrounds for tackling a lack of image data (Giakoumoglou, Pechlivani and Tzovaras, 2023). Also in the algricultural domain, Higuchi et al. (2023) created an interactive data synthesis interface, which allows more control over placement of items in creating tomato-detection model images.

Research by Chaturvedi et al. (2023) uses a generative adversarial network to learn foreground masks and automatically create composite copy-paste style images of birds and flowers. They found that their SS-CPGAN avoids generating trivial masks (all-zeros or all-ones). This automates the process of segmenting the source object dataset.

## 2.3 Segmentation for Context

The question arises: *what is context for roadside litter placements?* In other words, where is litter usually located alongside roadsides? While it is seen along curbs and within verges; this exact location needs to be localized in every possible setting. Segmenting these regions may provide the context needed for regions of placement.

Although used downstream and not as a dataset creation method, a process for finding particular regions has been seen with the creation of segDeepM (Zhu et al., 2015), a deep learning model that extracts regions of interest to aid object detection, which achieved a 1.4% mAP increase over the best reported method at the time (a 7-layer network) on the PASCAL VOC 2010 dataset (Everingham et al., 2010).

Numerous segmentation techniques are available both in the classical form and deep-learning domain. Color-based segmentation involves colour quantization within the HSV colour space before ROI locking, and has been found useful in segmenting road signs (L. Q. Chen et al., 2011). This may prove a challenge in both segmenting green grass verges and grey pavement regions with the same algorithm. Road sign segmentation has also been achieved using binarization search strategies which blend with edge detection approaches (Ashwini et al., 2023). Mean shift clustering is a non-parametric segmentation method that divides an image into homogeneous regions by shifting centers towards a vector until convergence and has been recently used for forests (Andrei and Grigore, 2023), another area of greenery. Litter detection locations seen in training data may be used to guide such clusters. If the number of verges in an image is known, parametric segmentation algorithms such as K-means Clustering (Dhanachandra, Manglem and Chanu, 2015) or Markov Random Fields may be of use in localizing regions of interest, the latter having been used with point clouds in autonomous driving solutions (Rummelhard et al., 2017). If it is difficult to localise verges themselves, which may not be present in every frame, perhaps localising the road first may be of use, using techniques such as watershed transform (Beucher and Bilodeau, 2005) to find catchment basins or texture-based search (Zhang and D, 1994) to determine the oriented road patterns.

Deep learning may also be of use as a segmentation step. Neural networks that perform segmentation usually have an encoder-decoder type structure, as the image input is converted into a 2-D mask. RCNN, a convolutional based region-proposal network, laid the groundwork for both object detection and segmentation in 2014 (Girshick et al., 2014). In 2017, Mask-RCNN tuned the structure further for segmentation, by including a dedicated branch while increasing efficiency by computing feature maps in parallel (He, Gkioxari et al., 2017). The U-Net convolutional

model has had success in medical vision communities with its high accuracy with low amounts of training data, in fields of orthodontics (X. Wang et al., 2023), diabetic retinopathy (Durai and Jaya, 2023) and cancer research (Hossain et al., 2023). Spatial information is preserved by passing information to the decoder before the encoder terminates, concatenating layers from both elements to withhold fine-detailed features (Ronneberger, Fischer and Brox, 2015). Aswell as being a current state-of-the-art model for object detection, YOLOv8 (Jocher, Chaurasia and Qiu, 2023) itself also has a variation which includes a module for semantic segmentation. Unfortunately, deep learning methods may require many annotated samples of verges which may prove difficult in the time limit of this research project. However, the 'zero-shot' "segment anything" model (Kirillov et al., 2023) has proven useful in fast annotation for segmentation tasks with its zero-shot promptable mechanism trained on 1 billion masks.

## 2.4 Style Transfer

Publicly available litter datasets vary in terms of context (environment, contrast, general appearance) from the sourced dash-cam litter dataset- where waste samples are much smaller in a lower resolution. However, it has been shown that adapting the domain by style transfer to harness externally-sourced dataset information may increase performance of the target object-detection solution.

Y. Wang et al. (2022) develops a feature-based style randomization as an augmentation strategy which integrates random noise for the classification task. In this work, the source datasets are the public benchmarks PACS (D. Li et al., 2017), Office-Home (Venkateswara et al., 2017) and VLCS (Torralba and Efros, 2011) with the target domain being a random noise generator. Random noise variables are created with the same dimension as the original feature style embeddings. The AdaIN model (Gatys, Ecker and Bethge, 2015) is used to replace convolution features of the content images of those with the style images. Compared to their baseline model, the method increased average accuracy from 96.17 to 96.62%.

Kadish, Risi and Løvlie (2021) tackle the challenge of object detection in art im-

ages, quoting the cross-depiction problem: neural networks tend to prioritize texture features over shape features on identification. An ImageNet (Jia Deng et al., 2009) pretrained Faster-RCNN network (Ren et al., 2017) was trained using StyleCOCO - a subset of the original COCO dataset (Lin, Maire et al., 2014) whereby images are stylised with an artistic style using AdaIN (Gatys, Ecker and Bethge, 2015). The model improved upon the average precision at a .5 IoU threshold of the baseline from 0.58 to 0.68.

## 2.5 Literature Review Outcomes

The assessment of domain-adjacent research helped guide the research design. The TACO (Proença and Simões, 2020) dataset is commonly used in literature, indicating that the annotations are of a satisfactory quality. The YOLO model has been suitable for waste detection, and so could be used in the context of dashcam litter detection (Redmon et al., 2015). The copy-paste composition strategy for image synthesis appears feasible with the improvements it has made on baseline detection results in literature, given a defined source and target dataset. It is not clear what exact contextual method of pasting works the best over the range of scenarios, and so this motivates a comparison experiment. Segmentation strategies are diverse, even in related domains, and so these will also need to be compared. The AdaIn model (Gatys, Ecker and Bethge, 2015) appears to be a sound method of style transfer and so this will be carried forward. The chosen methods will be explained in detail in the following section.

# Chapter 3

# Methodology

## 3.1 Project Management

At the stage of proposal, the project aimed to find an optimum segmentation strategy to find ways to increase litter detection accuracy by highlighting regions of interest. Since then, the project had taken a different turn, leading to the dismissal of the originally created Gantt chart. The segmentation element has now been implemented as a "context" option for the copy-paste synthesis strategy, i.e. determining the areas of images in which to place synthetic litter pieces to increase object detection performance metrics. On discovering the project is more investigative, a Pert chart seemed more apt, as different avenues were to be explored depending on progress of findings. The pert chart is displayed in figure 3.1.

The PERT chart was adhered to and meant that little supervisor provision was necessary. Few supervisor meetings took place over Microsoft Teams on an ad-hoc basis initially for establishing direction, but focus was maintained during the course of the project due to well designed research questions.

## 3.2 Research Methods

### 3.2.1 Experimental Design

The overall experiment involves determining whether adding synthesised dashcam data to current datasets increases the performance of a YOLOv8 (Jocher, Chaurasia

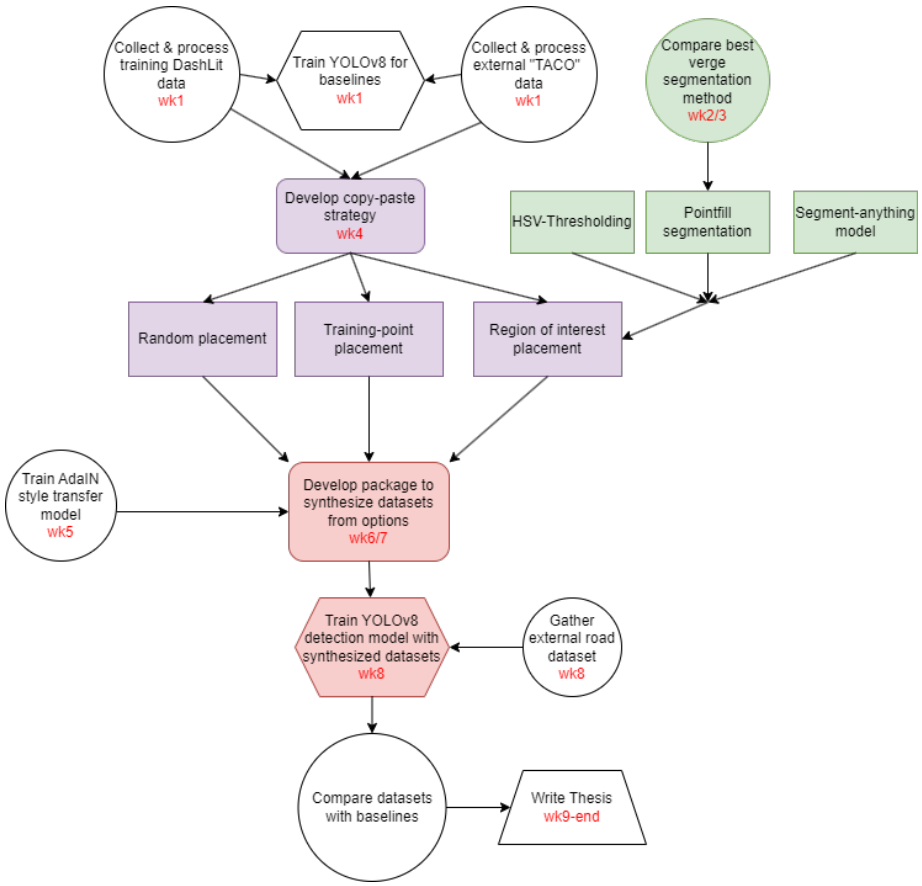


Figure 3.1: PERT chart displaying project flow and week assignment.

and Qiu, 2023) detection model when trained with such. Within the comparative experiment, questions are explored to determine the relevant variables in creating such synthesised data. In terms of segmenting roads as a contextual option for placement, the question is asked:

- *Which form of verge-segmentation achieves the best performance?* This will be determined by comparing DICE and IOU scores - as well as visual output of methods - against a road dataset annotated with segmentation masks.

The remaining questions involve detection performance when varying location, style, data sources, and blending. In particular, the precision and recall at a 0.3 IOU threshold and average precision at 0.5 and 0.5-0.95 thresholds when YOLOv8 (Jocher, Chaurasia and Qiu, 2023) is trained with such synthesised sets.

- *Which pasting location strategy creates the most detection performance-boosting datasets?*

- *Does transfering target style to pasted items increase detection performance?*
- *Does pasting on out-of-sample road images increase detection performance?*
- *Does blending pasted items have a significant impact on detection performance?*

### 3.2.2 Requirements

The requirements for the project have been informed by literature review and best practices, and consist of data, hardware and software. They are as follows:

#### Data

1. Dashcam image data with bounding-box litter annotation files in a format for a suitable object detection neural network.
2. Litter image data with annotated segmentation masks suitable for cutting and pasting.
3. External dashcam-style road image data not seen in the original dashcam data.

#### Hardware

1. Computing power to run dataset creation and querying.
2. Graphical processing units (GPUs) capable of running both training of deep neural detection networks and inference using such networks.
3. Storage to house original datasets and those created using dataset synthesis.

#### Software

1. A programming framework suited for both image data manipulation and running deep learning models.
2. A library for image processing and augmentation.
3. Libraries for data manipulation in storage formats such as matrices and dataframes.
4. Software tools for reading and converting annotation formats.

5. A deep learning library which can define and aid compilation of detection networks.
6. A library for displaying graphs.
7. Deep learning output management tools.

### 3.2.3 Data Sources

#### DashLit

The "*DashLit*" dataset is a series of frames taken from dashcam footage that contain litter. Footage was collected by the Laboratory of Vision Engineering (LoVE) at the University of Lincoln, 2023. The footage consists of country highways around Lincolnshire, with some varied samples of rain and lighting conditions. Litter items were annotated by bounding boxes with a single class "litter" using both CVAT (Sekachev et al., 2020) and the University of Oxford's VGG Annotation tool (Dutta and Zisserman, 2019). Annotations were supplied in csv format which was later converted to the YOLO format for processing by the relevant object detector. In total there are 11,565 images with a total of 20,927 instances of litter. Note that there is repetition of instances- whereby subsequent frames may contain the same piece of litter, but at a slight varied position and lighting condition (as the capturing vehicle travels along the road).

#### Source Litter (Litter to Paste)

The images chosen for synthetic litter placement were that of the original TACO dataset (Proença and Simões, 2020). This was chosen due to its use in other litter research, but also due to the supplied segmentation mask and bounding box of each item. This would allow for litter pieces to be "cut out" using their supplied mask, and the bounding box data used in auto-labeling synthesised litter samples. This original dataset includes 1500 images of 4784 completely unique samples.

### For Segmentation Evaluation

Image data was needed to evaluate different segmentation methods prior to using the method for contextual region litter placement. 40 highly varied dashcam images were collected from both the established training set and out-of-sample LoVE-collected dashcam footage (including some inner town and dark, night samples). Ground truth verge and pavement masks were annotated using the "intelligent scissors" option of the CVAT annotation software (Sekachev et al., 2020). Verges were defined as green grass adjacent to the road and pavement and the bush verges directly adjacent to these (not including trees). Intelligent scissors is a live-wire boundary detection technique solving a two-dimensional graph searching problem for mask creation (Mortensen and Barrett, 1995).

### External Road Data

One of the planned experiments was to use new dashcam backgrounds for the synthetic dataset creation. These backgrounds were taken from the supervisely country road dataset for road segmentation (Supervisely et al., 2022). No information on the locations captured within the images is supplied, but on using Google Translate's (Google, 2023) image-reading capabilities on road signs, it appears some if not all of the data is from Bulgaria. There were 917 images in total. It is important to note that possible occurrences of natural litter were not annotated.

Figure 3.2: Example training frames from the external "supervisely-roads" dataset.

### 3.2.4 Ethics

Ethics in this research primarily concerns image data used for synthesis processing and in deep learning models. The TACO dataset carries an MIT license allowing for use in experiments, and contains no identifiable people. The DashLit dataset was collected in-house, and does not contain identifiable people. The Supervisely Country Roads dataset does not contain licensing information on the Github page, however a linked blog states that the data is free to use for machine learning purposes, also providing a tutorial on use. People contained within its images are unidentifiable. Images in dashcam data do identify some license plates of vehicles, which are in the public domain, however in any publications these are censored out of courtesy.

### 3.2.5 Determining Context

Different methods of contextual litter placement will be compared.

#### Random Placement

Random placement involves placing litter samples in random locations in dashcam frames as a synthesis technique. Ghiasi et al. (2021) suggests this may be suitable as a standalone synthesis method, while Dvornik, Mairal and Schmid (2018) uses this as a context baseline, which this research follows.

#### Training-point Placement

Training-point placement involves placing new litter instances in areas seen in the training data (matching their bounding-box centres). Validation and test points are not used as this would be a form of data snooping, and would effect the soundness of experiment. Training-point placement follows the logic that litter in these points are "reasonable" locations, similar to that of the Fang et al. (2019) probability map copy-paste augmentation technique. In case of varying frame size, points are normalized then fitted to each frames dimensions before processing.

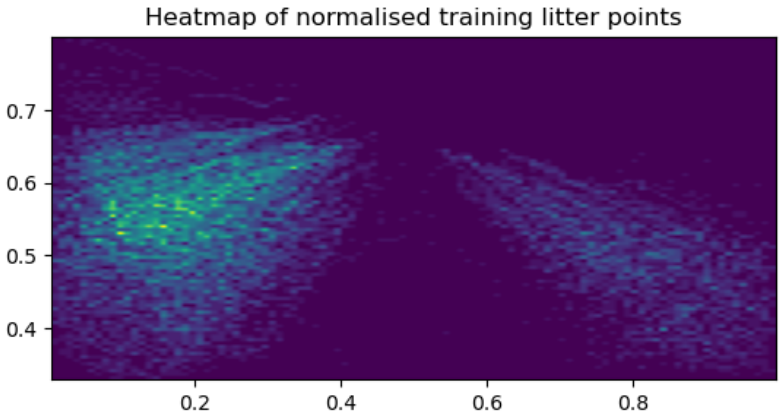


Figure 3.3: Heatmap of normalised litter centre points found in training data.

#### Region of Interest Placement

The final method of establishing context for placing litter items is through calculating regions of interest. These would be the areas of grass verges and pavements at the sides of roads where litter tends to reside. Finding these regions autonomously occured via segmentation strategies. Colour thresholding, region filling, and deep learning methods utilising training data litter locations were used to localize these regions, described in the following subsection.

### 3.2.6 Segmentation Methods

As a precursor for comparing contexual methods of litter placement, an optimum segmentation strategy must be found for the "region of interest" contextual placement option. Segmentation strategies were chosen based on both their suitability with current data and absence of annotation time required (due to the time limitations of the project).

#### HSV-Threshold Segmentation

HSV-thresholding is a segmentation method which highlights regions of interest based on colour information; the range of hue (red, green, blue), saturation (colour intensity) and value (brightness). As this has been successful in segmenting road signs based on colour (L. Q. Chen et al., 2011), this may prove successful in highlighting green verges of interest for litter placement in similar dash-view road

imagery. The benefit of HSV colour representation over RGB is that it is less sensitive to illumination changes- which would be of benefit under varied light conditions viewed through the dashcam. Another benefit is that it is easier to define colour ranges due to such brightness separation from hue and saturation. Conversion is made through OpenCV's *cvtcolor()* functionality (Bradski, 2000). A range was set from light green (80,175,200) to dark green (36,25,50).

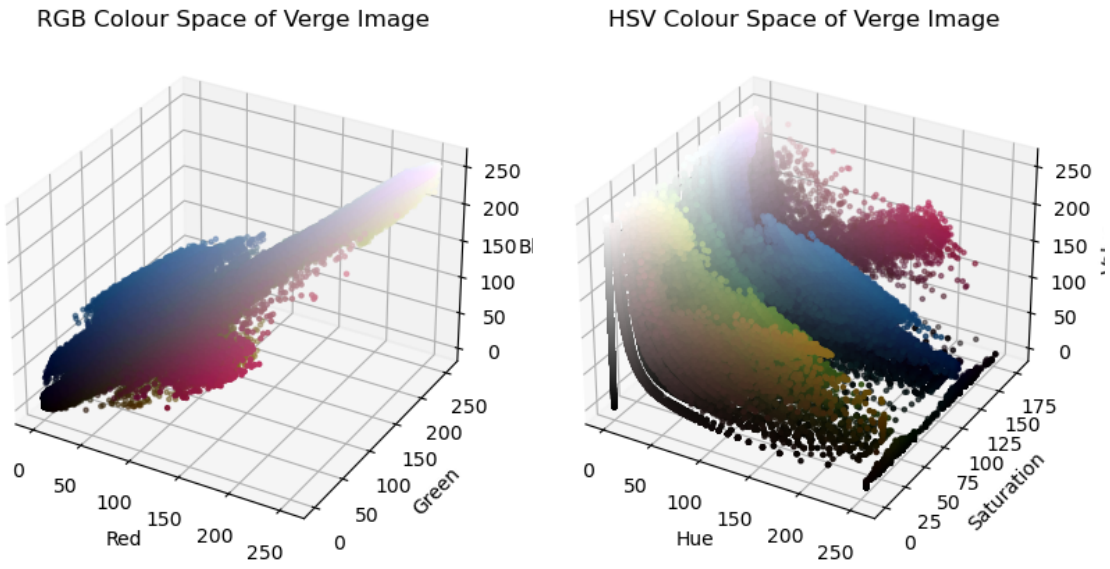


Figure 3.4: RGB and HSV colour spaces of a dashcam frame.

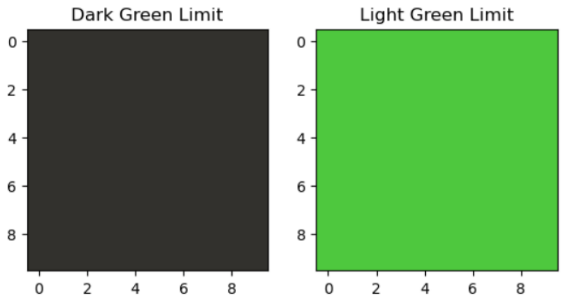


Figure 3.5: HSV Range for thresholding segmentation.

### K-Means Clustering for Seeding Segmentation

Due to training data containing annotated litter points (bounding box centres), it was investigated whether this information could be used to localize a litter placement area in every frame. Reducing these points could hint at a more general area of interest (as dashcam frame environments are dynamic, pavements and verges may

not be in the same relative area in every frame). Reducing points may also reduce computational time in using these within a segmentation technique.

K centroids would be assigned as some of the litter training points. Distance is calculated between each point and the centroids. Points are then assigned to the nearest centroid as a cluster. Centroids are then adjusted to the centre of these points (the mean). This system repeats until there is no significant change to the centroid locations. Returned is the k general cluster centres that can be fed into a segmentation algorithm as a seed to determine the region of interest in each image.

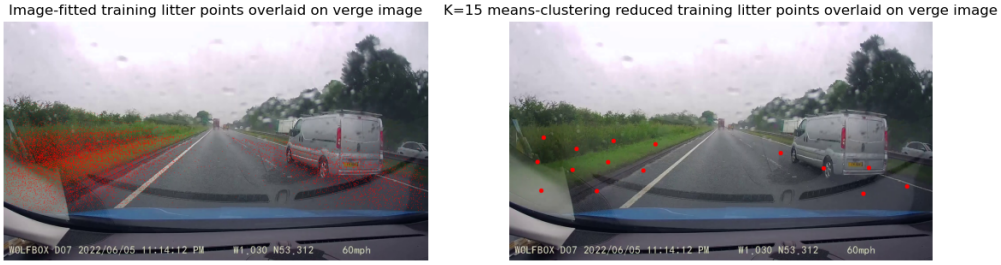


Figure 3.6: Training litter points before and after being reduced using K-Means Clustering (K=15), overlaid on a frame sample.

### Seeded Pointfill Segmentation

Region-filling, point-filling or flood-filling is an algorithm that identifies adjacent values in an image based on a similarity to a point seed. Working in grey-scale, this similarity is a "tolerance" factor. The tolerance is the maximum absolute difference in pixel intensity between the seed point and the neighbouring pixel in order for it to be considered the same flood region. Therefore, a high tolerance would be prone to filling more neighbouring pixels, while a low tolerance would be prone to filling less neighbouring pixels. Filled regions can be converted and concatenated into a mask- which would ideally be the region of litter placement more specific to the current frame.

### Seeded Segment Anything Model

The Segment-Anything Model (Kirillov et al., 2023) allows for the production of segmented region masks without the need for training (coined "zero-shot"). It leverages this using transfer-learning, whereby the neural network model has been trained

Figure 3.7: Converting an image to grey-scale and flood-filling a point seen in the training data (tolerance = 10).

on 1 billion masks over 11 million images (the SA-1B dataset), and can use this information to generalise masks in new images based on a prompt.

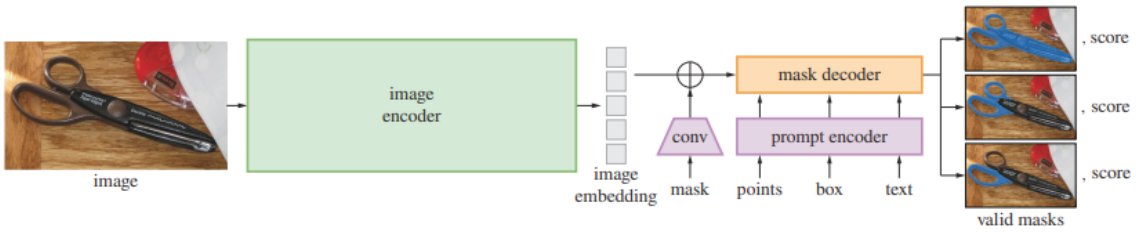


Figure 3.8: SAM model overview (Kirillov et al., 2023).

The image encoder is a mask auto-encoder (MAE) (He, X. Chen et al., 2021) pre-trained Vision Transformer (ViT). While the prompt encoder comes with a sparse and dense (mask) encoder, the former will be extensively used for these experiments as it can handle points (available), boxes and text. The mask decoder maps the image and prompt embeddings aswell as an output token to a mask using a transformer decoder block and a dynamic mask prediction head. The overall structure is seen in the paper's supplied figure 3.8.

The SAM model outputs three predicted masks when given a single prompt. These are ranked by an estimated intersection-over-union (IoU) score. The IoU prediction head is supplied trained with mean-squared-error loss between the IoU prediction and the predicted mask's IoU with the ground truth mask of the SA1-B dataset.

The result of a SAM mask inference using k=15 reduced training points as a prompt can be seen in figure 3.9.

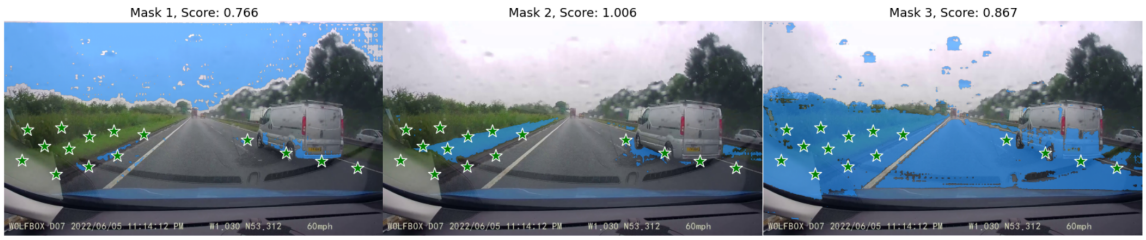


Figure 3.9: A SAM output when queried with a dashcam frame and 15 points (Best = Mask 2).

#### Segmentation Performance Measures

The segmentation method with the highest performance measures will be chosen as the region-of-interest form of context for litter placement. These measures are DICE and IOU score.

Where X = predicted mask and Y = ground truth, the metrics are calculated as:

$$DICE = \frac{2 * |X \cap Y|}{|X| + |Y|} \tag{3.1}$$

$$IOU = \frac{|X \cap Y|}{|X \cup Y|} \tag{3.2}$$

The IOU score penalises a greater difference between the predicted mask and ground truth mask more so than the DICE metric. However, they are positively correlated, and so both are used to give a greater overview of segmentation performance. The median and mean of these metrics over the 40 evaluation dataset will be mostly scrutinised for insight.

### 3.2.7 Style Transfer: AdaIN

The motivation for transferring style onto external litter samples was such that appearance could better match those seen natively in dashcam footage. As can be seen in figure 3.10, there is a stark difference in appearance between the waste samples in DashLit and TACO.

The adaIN model (Gatys, Ecker and Bethge, 2015) is used for style transfer as it is

(a) DashLit litter samples. (b) TACO litter samples.

Figure 3.10: Comparing the appearances of DashLit and TACO litter.

not only used often in literature for similar works (Y. Wang et al., 2022; Kadish, Risi and Løvlie, 2021), but also as a modular component in StyleGAN (Karras, Laine and Aila, 2018), a generative adversarial network used to generate unseen content with a certain style.

The network takes a content image and style image and returns an output that recombines the content of the former image with the style of the latter. The encoder is defined as the first layers of a pretrained VGG-19 model (Simonyan and Zisserman, 2015). Once the content and style images are encoded, both feature maps are passed to an AdaIN (Adaptive Instance Normalisation) layer that aligns the mean and variance of both, producing a new, target, feature map (equation 3.3). A trained decoder upscales the produced target feature map, which mirrors the encoder without normalization layers.

$$\text{AdaIN}(x, y) = \sigma(y) \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y) \tag{3.3}$$

The network produces two loss metrics, one for content and one for style. The content loss is the euclidean distance between the target features and the features of the output image, while the style loss is the mean squared difference between the normalized style features of the content and style images. The loss function is a weighted combination of the content loss and style loss. This loss can be minimized with the ADAM optimizer.

On training the AdaIN model to transfer the style from DashLit to TACO, training

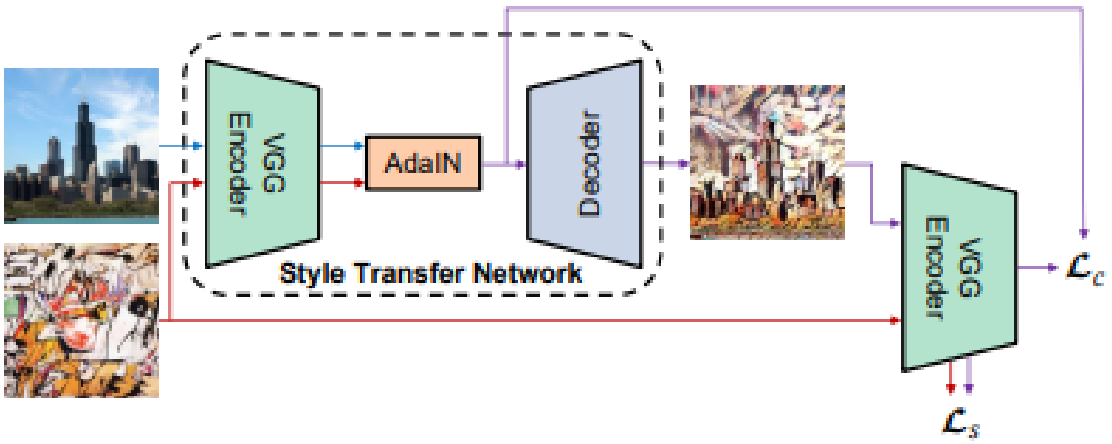


Figure 3.11: AdaIN structure from Gatys, Ecker and Bethge (2015)

can be said to be complete when the losses converge, i.e., there is no longer a significant change between epochs (training interations). Style transfer performance was assessed using qualitative assessment by the researcher.

### 3.2.8 Object Detection: YoloV8

YOLO "You Only Look Once" object detection models have been iteratively improved on by different development teams over the years, with the first created by Redmon et al. (2015). The network introduced single-stage detection using a grid-based system, which combined localization and classification by determining the probability of an object residing in each of these image cells. Proposed bounding box predictions can be reduced using non-maximum-suppression. This improved upon previous two-stage methods (such as RCNN (Ren et al., 2017)) in terms of speed, as these relied on extracting region proposals before passing a classifier over each separate region.

The fifth YOLO iteration (Jocher, Stoken et al., 2020) has had success in previous litter detection experiments (Córdova et al., 2022). It was developed by the Ultralytics team who have since released a new version, YOLOv8 (Jocher, Chaurasia and Qiu, 2023). The Ultralytics versions are controversial in that there are still no published papers for version 5 or 8 at times of writing, however, the community is highly active on GitHub amassing over 160 contributors and over 4,000 users, with active bug-solving members, available tutorials, and recent published work using the

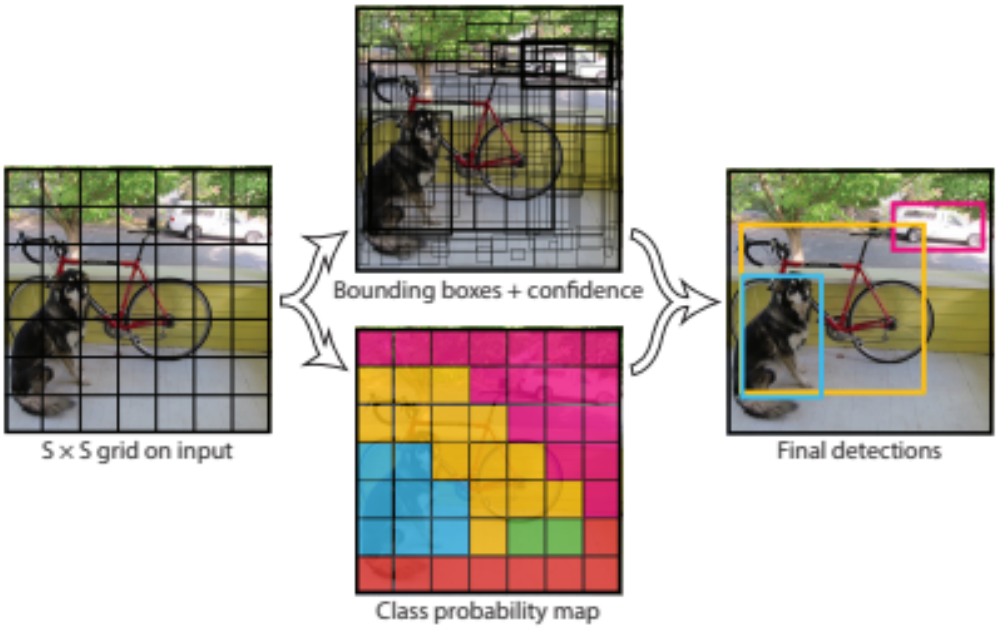


Figure 3.12: The YOLO strategy (Redmon et al., 2015).

YOLOv8 model (Aboah et al., 2023; Y. Li et al., 2023; Lou et al., 2023; Talaat and ZainEldin, 2023). This eighth iteration boasts the mosaic augmentation (which stitches several training images together to reduce overfitting), anchor free detection (object centre localization and size regression for bounding box proposals) and a supposed increase in performance over version 5 on the COCO benchmark (Lin, Maire et al., 2014). Also of interest for litter detection is its inclusion of an FPN (Feature Pyramidal Network) module, which is known to improve detection of small objects such as those in question (Lin, Dollár et al., 2017). It does so by combining feature maps from different scale convolutional layers such that small details are not lost in the downsampling process.

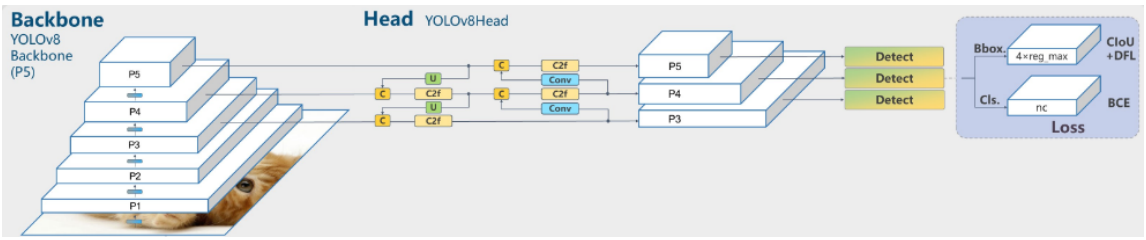


Figure 3.13: YOLOv8 model architecture graph by GitHub user RangeKing (2023).

YOLOv8 outputs three loss metrics, namely class, box and DFL. Class loss (varifocal loss) focuses on the correct object prediction (the difference between the pre-

dicted class probability and the ground truth label). The box loss (mean squared error) focuses on accurate bounding box proposal localisation (the difference between the predicted bounding box coordinates and the ground truth coordinates). DFL (Distribution Focal Loss) concerns the class imbalance issue in object detection, as -especially in small object detection- the number of background instances (area outside of objects) far exceeds that of the objects themselves. This leads to the model optimising for background prediction, rather than object. DFL loss takes this into account by including the distribution of object instances in its calculation, focusing on the underrepresented classes.

A number of metrics will be assessed for the performance of this model on the created litter datasets. These consist of recall, precision and average precision. Recall is calculated as the number of correct litter detections out of all litter detections (equation 3.4); concerning how sensitive the model is to detect. Precision is calculated as the number of correct litter detections out of all detections (equation 3.5); concerning how robust the model is against making false positive detections.

$$Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \tag{3.4}$$

$$Precision = \frac{True\ Positives}{(True\ Positives + False\ Positives)} \tag{3.5}$$

Average precision (equation 3.6) is a metric more suited to models which produce proposed detections across multiple intersection-over-union thresholds, such as YOLOv8. Average precision is calculated by ordering the detected instances by their confidence scores, calculating precision and recall, plotting the precision-recall curve, and calculating the area under the curve for each threshold.

Where $R = recall$, $P = precision$, $n = number\ of\ thresholds$, $R_n = 0$ and $P_n = 1$:

$$AP = \sum_{k=0}^{k=n-1} [R_k - R_{k+1}] * P_k \tag{3.6}$$

As the target to detect ("litter") is one class, average precision (AP) = mean average precision (mAP). Average precision will be assessed at different IoU thresholds, i.e. objects count as detected when a certain area of the predicted bounding box overlaps the ground truth bounding box.

This leads the detection performance metrics to be:

1. Precision (P) @ 0.3 IoU Threshold.
2. Recall (R) @ 0.3 IoU Threshold.
3. Average Precision (AP) @ 0.5 IoU Threshold.
4. Average Precision (AP) @ 0.5-0.95 IoU Thresholds.

As the litter objects are small, examining performance at a low (0.3) threshold is more lenient, such that litter is counted as a detection if there is only 30% overlap between the predicted bounding box and ground truth bounding box. Examining at both 0.5 IoU and the 0.5-0.95 IoU overlap average allow comparison with litter detection in other papers, as this is a common baseline metric for analysing such average precision.

### 3.2.9 Blending: Poisson

Blending is a strategy to integrate pasted objects into the image environment naturally, to create a visual sense of belonging. The Poisson method (Pérez, Gangnet and Blake, 2003) removes the phenomena of boundary artefacts (borders around pastes) in the work of Dwibedi, Misra and Hebert (2017) and improves performance.

Firstly, there exists the source object and the destination image. Gradients in the destination image are calculated which describes the intensity change between pixels, visually seen as lighting and shading. This information is given to the Poisson equation along with the masked source object. The source objects gradients and

boundary are converted into a form consistent with that of the destination image, so that it "blends" in. This occurs through solving the equation. This method is also known as "seamless cloning".

For litter placements, this will occur as a last stage after finding the most optimum litter placement strategy through context and style experiments.

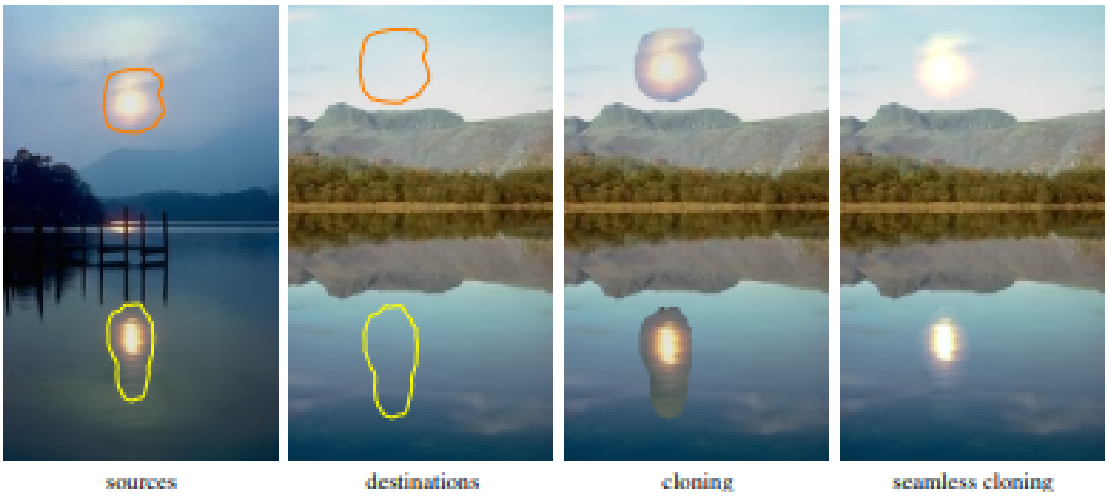


Figure 3.14: Seamlessly cloning the sun and its reflection into an image as proposed by the Poisson image editing paper (Pérez, Gangnet and Blake, 2003).

### 3.2.10 Experimental Procedure I - Best Segmentation Method for Context

As a precursor to the synthesised dataset detection performance comparison, the best method of verge segmentation was investigated as an option for the contextual litter placement.

#### HSV-Thresholding Results

While testing the hsv-thresholding between light and dark green values on an image frame, it was seen that the method was subject to noise- green-tinted items that were not part of the grass verge areas were highlighted and therefore became part of the pixel selection mask. To negate this, the largest connected component was selected, which forfeited smaller possible verge regions. To clean this new mask, it was dilated

with a disk morphological element of size 15. Holes were filled by combining said mask with the flood-filled inversion using the bitwise OR operation.

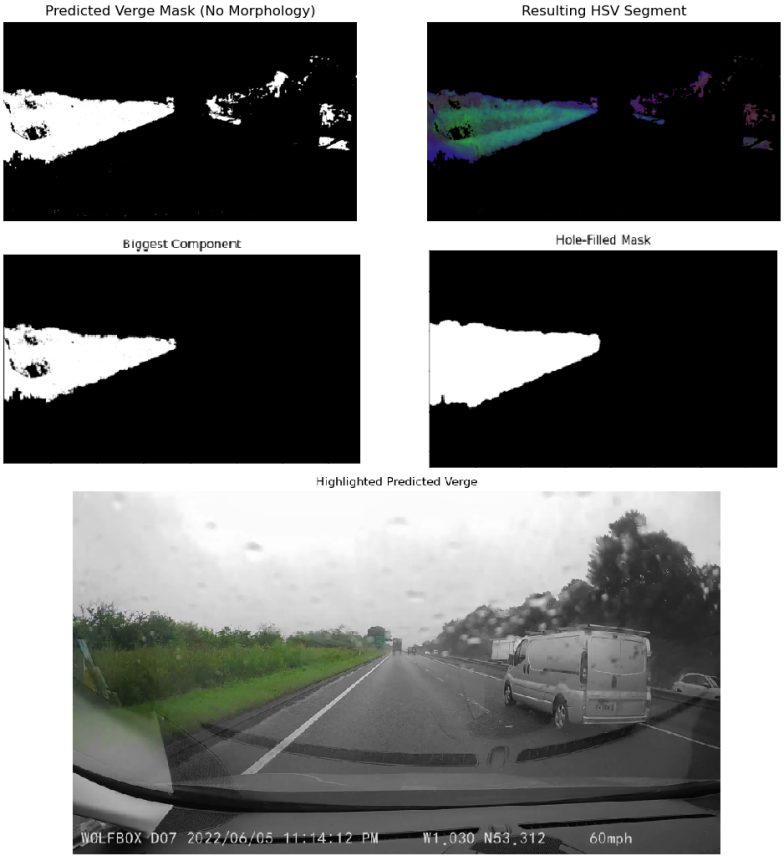


Figure 3.15: Verge HSV-thresholding segmentation pipeline.

Visualising the predicted mask against the ground truth also shows how specific the algorithm is in that only the grass area is selected, not the curb areas where litter could also reside.

The algorithm performed poorly by producing little to no mask in over exposed and dry-grass verges, night frames, and as expected, frames with no grassy areas.

### Pointfill Segmentation Results

The pointfill segmentation method was to load litter training point centres, reduce these points by k-means clustering, and flood them to produce a joined mask. This mask was slightly dilated (10 times with a disk morphological element of 1) for a more general area that would be used as a litter placement region.

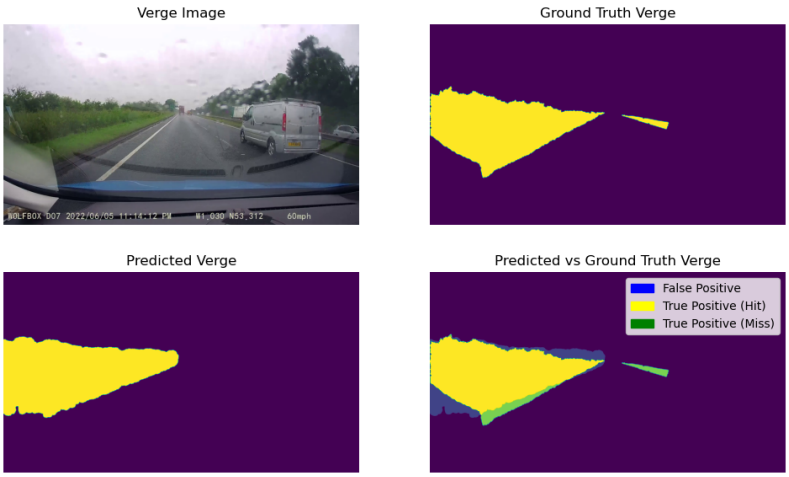


Figure 3.16: Visualising a HSV-thresholded prediction against ground truth.

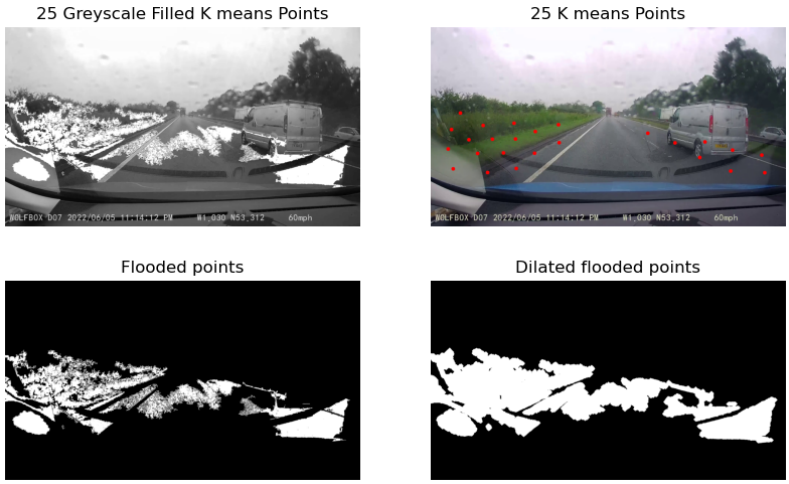


Figure 3.17: Verge pointfill segmentation pipeline.

To find the optimum pointfill strategy, the two variables were explored; the value of 'k' for reducing training points, and the level of tolerance - how large the pixel intensity gradient is to prevent (or enact) an adjacent region-fill.

To find the best k value, the tolerance was controlled at '7'. K values were assessed from 5 to 40 in skips of 5. Each of the 40 evaluation images were evaluated based on mean and median IoU score.

The IOU score rose until it hit approximately k=15 for the median and k=20 for the mean. As the mean graph converged more smoothly, the k plateau value of 20 was chosen. Following this, the tolerance value was assessed between values 1 to 10 with the k value controlled at 20.

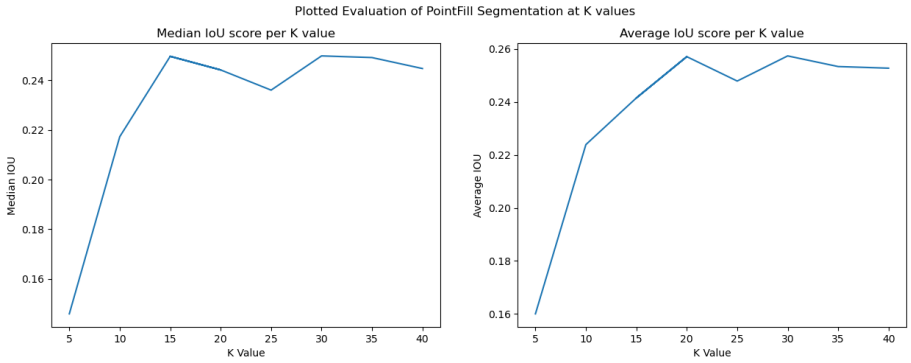


Figure 3.18: Pointfill K-value determination based on mean and median IoU score.

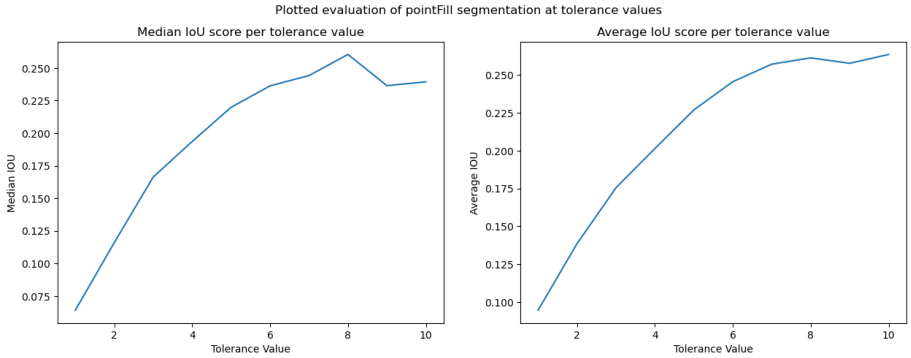


Figure 3.19: Pointfill tolerance determination based on mean and median IoU score.

The tolerance value of 8 was chosen due to returning peak median IoU score and converging mean IoU score.

Using k=20 and tolerance = 8, the 40 images were evaluated. The algorithm visually performed better than HSV-thresholding, working on some night data that had a certain level of street lighting. However, the algorithm tended to over-estimate, including not only the road, but significant sections of vehicles and buildings.

### Segment Anything Model Results

In a similar fashion to pointfill, the SAM model strategy was to use k-reduced training litter point centres as a prompt to return a mask.

K-values were first assessed to find the optimum value. Values between 20 to 45 were examined to check the change in IoU score on the evaluation set. Changes per increase in K were more erratic and noisy than with pointfill segmentation, especially

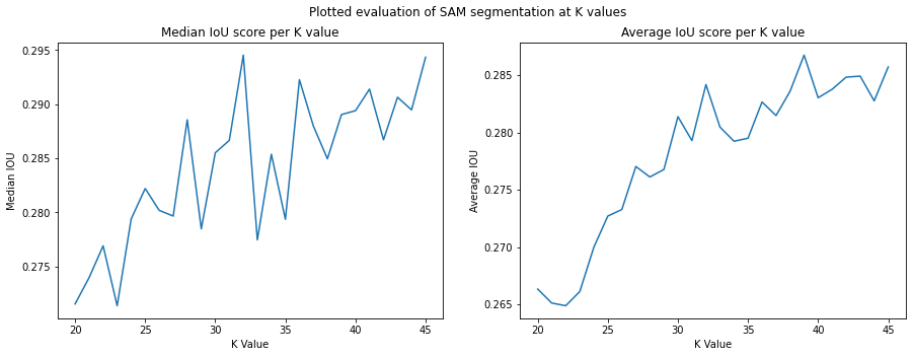


Figure 3.20: SAM K-value determination based on mean and median IoU score.

with the median scores, and so k = 40 was chosen due to convergence in the average IoU score graph.

**Evaluation for optimal segmentation strategy.**

The mean dice and IoU scores for all segmentation can be seen in table 3.1, showing per-image scores as bar charts in figure 3.21 .

The HSV-Thresholding method is the least favourable segmentation method, with the lowest IoU/DICE scores, and the bar chart showing that some frames have no verge detection, due to lighting conditions or absence of greenery. Both pointfill and SAM scores have similar mean performance values, both higher than HSV-thresholding, but with the SAM model scoring the highest. On viewing the bar charts however, it seems that the SAM model performs more consistently at segmentation over the evaluation set, including over the challenging scenarios, and as such will be chosen to be the 'region of interest' context experiment for litter placement in the main work. It is of note that the scores are still not the highest compared to ground truth segmentation, however the segmentation strategy still manages to separate items such as the sky and dashboard, with which litter should not be placed.

Table 3.1: Table comparing mean DICE and IoU scores for segmentation methods.

| **Method** | **Mean IoU** | **Mean DICE** |
|---|---|---|
| HSV-Thresholding | 0.175 | 0.26 |
| PointFill | 0.261 | 0.4 |
| Segment Anything Model | 0.283 | 0.429 |

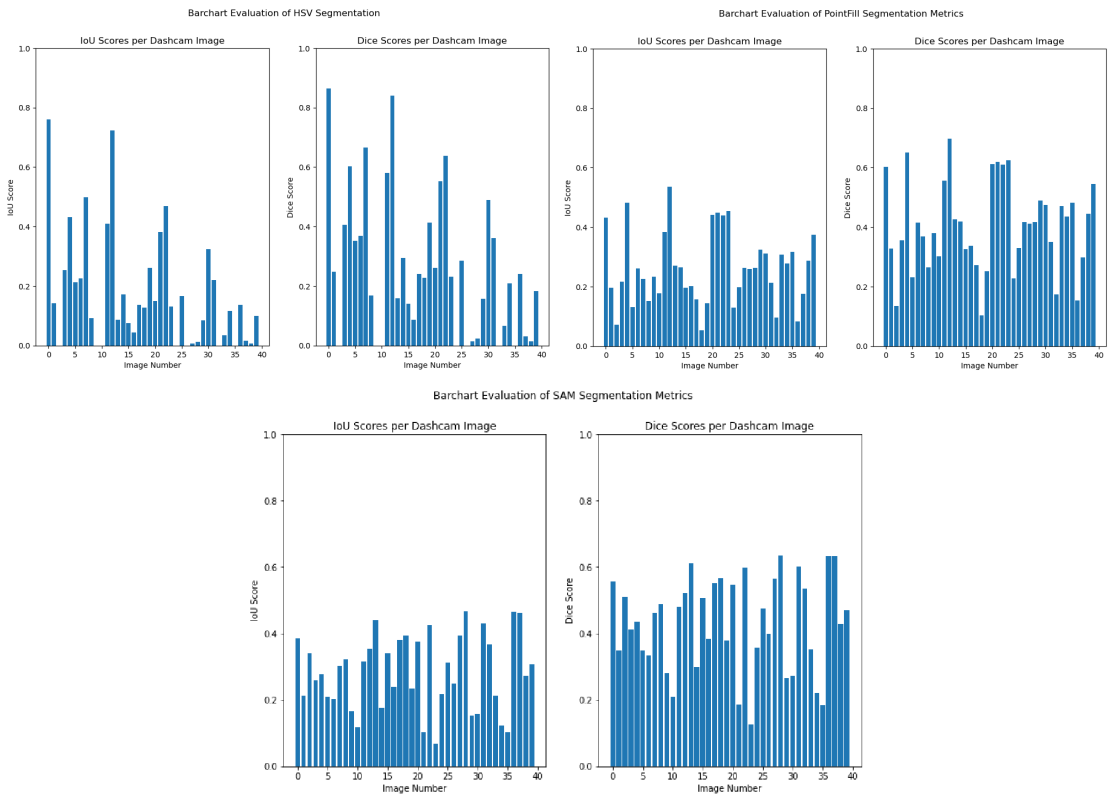


Figure 3.21: IOU and DICE scores of segmentation techniques on verge/pavement segmentation of the 40 evaluation images.

### 3.2.11 Experimental Procedure II - Dataset Comparison

#### Litter Detection Baselines

The first task for the dataset comparison was establishing some baseline control metrics. It was decided on that there would be two baselines - a pretrained YOLOv8L model trained with only the original DashLit frames, and a pretrained YOLOv8L model trained with both DashLit frames and untouched 1500 TACO litter images (DashLit-TACO).

The DashLit images were split into training, validation and test sets. The split was a ratio of approximately 70%; 10%; 20% respectively in terms of images. The count of images and litter instances in each are seen in table 3.2. As the experiment focuses on changes in the training data, the amount of sole DashLit data in all comparison validation and test sets remain the same.

The model comes pretrained with the COCO (Lin, Maire et al., 2014) dataset,

Table 3.2: Table showing the data in each dataset split. On the left of each item (|) is the image count, and on the right is the instance count (litter samples).

| Data | Train | Validation | Test |
|---|---|---|---|
| DashLit | 7990\|14644 | 1201\|2904 | 2374\|4189 |
| DashLit-TACO | 9490\|19428 | 1201\|2904 | 2374\|4189 |

the 'L' equivalent having 43.7M parameters. The hyper-parameters for the original YOLOv8L - DashLit baseline were established using manual grid search in prior experiments run by the Laboratory of Vision Engineering (University of Lincoln, 2023) team. These were kept constant for all experiments in this research and are shown in table 3.3.

Table 3.3: Table showing YOLOv8 hyper-parameters used in experiments.

| Hyper-parameter | Value |
|---|---|
| Epochs | 20 |
| Batch Size | 5 |
| lr0 (Learning Rate) | 0.001 |
| Weight Decay | 0.05 |
| Image Size | 896 |

Other YOLOv8 defaults remain the same- mosaic, scale, translate, HSV, and random horizontal flip augmentations are left as standard.

### Generated DashLit-frame datasets

Synthetic datasets were generated to answer the research questions using tools made in the software development section 4.1. 2783 images were generated for most experiments offline to add to the DashLit baseline. These were from 2783 training frames of the DashLit dataset.

A random amount of litter between 2 and 7 inclusive out of the total TACO items was selected to be pasted in each frame. Litter items were resized. The method of resizing was to take the largest of the height/width, and resize it to a random size between 20 and 50 pixels, while keeping the pasted litter's size ratio. This sizing was determined after examining the litter bounding box area in the DashLit dataset, as seen in figure 3.22. The DashLit sample area histogram is left-skewed with the majority of samples having an area between a few pixels and 1500. To capture the

outliers, and possibly feed the model more information with larger samples, artificial litter could in theory be resized between  20 pixels and 2500 pixels in area using the resizing boundaries.

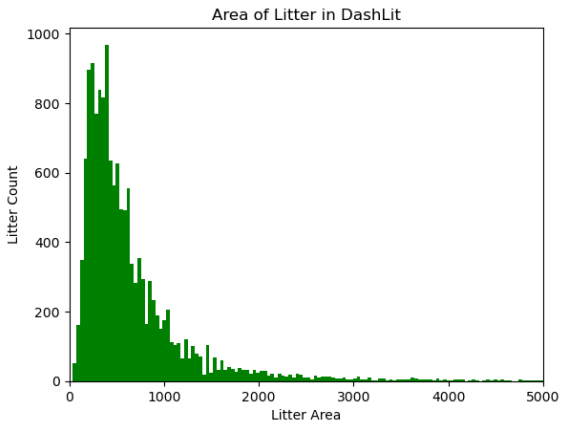


Figure 3.22: Histogram of DashLit litter size by area.

There was also random rotation of TACO litter , in that it had an equal chance of being placed in one of four directions. After pasting, no blending occurred for these trials. The amount of total training data in these new synthesised sets is seen in table 3.6. Note that unlike the baseline, not all litter instances in the count may be unique. Original instances present in the frames before placement are kept, in the hope that any overlap with artificially placed items may boost crowded model performance as in Jiangfan Deng et al. (2023).

Table 3.4: Table showing the training data in each generated dataset. On the left of each item (|) is the image count, and on the right is the instance count (litter samples).

| **Data** | **Train** |
|---|---|
| Random Placement | 10773\|33250 |
| Training Point Placement | 10773\|33013 |
| ROI (SAM) Placement | 10773\|33083 |

### Generated Stylised Datasets

A similar strategy for pasting occured with the addition of style transfer to the TACO samples prior to pasting. Firstly, the AdaIN model had to be trained with the style (DashLit) and content (TACO) images. The full 14644 litter instances from the Dashlit dataset were used with 4784 instances from TACO. These instances were

cropped out of their respective images by their bounding box, resized to 96x96 and saved prior to processing for AdaIN. Training, validation and test images were split in the ratio 80%; 10%; 10% respectively. The AdaIN model was compiled with its mean squared error (MSE) loss function and an Adam optimizer. Style weight was set to 4. Steps per epoch were set to 50. The model was trained until the loss converged, which was at 30 epochs (figure 3.23). A callback was defined such that style transfer quality could be examined after every epoch (3.24).

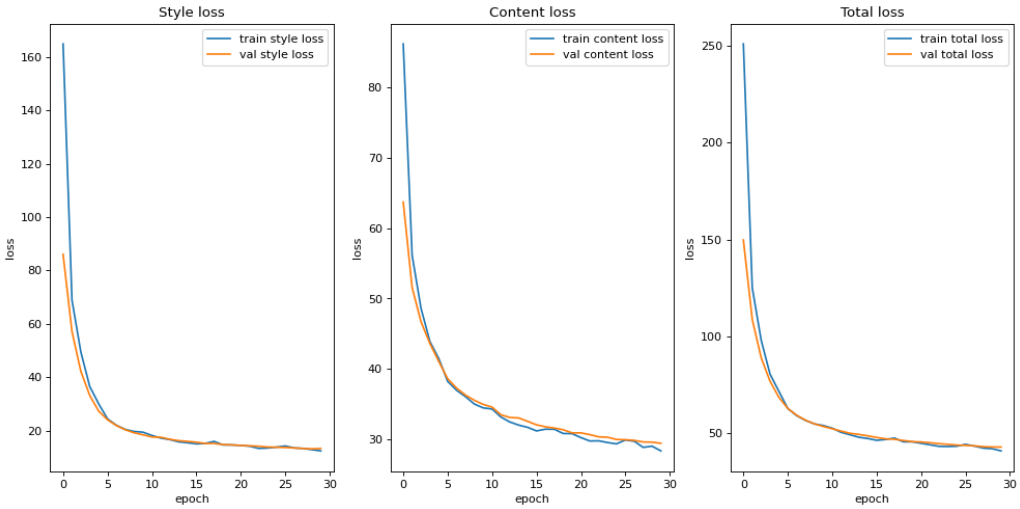


Figure 3.23: Style, content and total loss graphs. Loss converges at 30 epochs, with training and validation loss graphs maintaining a close trajectory.

Batches of style-transferred litter samples were visually assessed and deemed similar to native samples in the DashLit dataset. After training, the checkpoint was saved to be brought into the synthesis tool. Litter was pasted in the same manner as the non-stylised datasets, except a random style from samples in 1000 DashLit images was transferred to each. The variation in transferring different DashLit target styles to the same TACO instance can be seen in figure 3.25. Litter could be cut-out using the original segmentation mask as reference.

### Generated External-Frame Datasets

The experiment involving the external dashcam frame data (supervisely-roads) added 917 images, to which the optimum pasting solution would be added as discovered by running the previous experiments. The best solution is discussed in results (training point placement). The addition of blending was also included in these experiments.

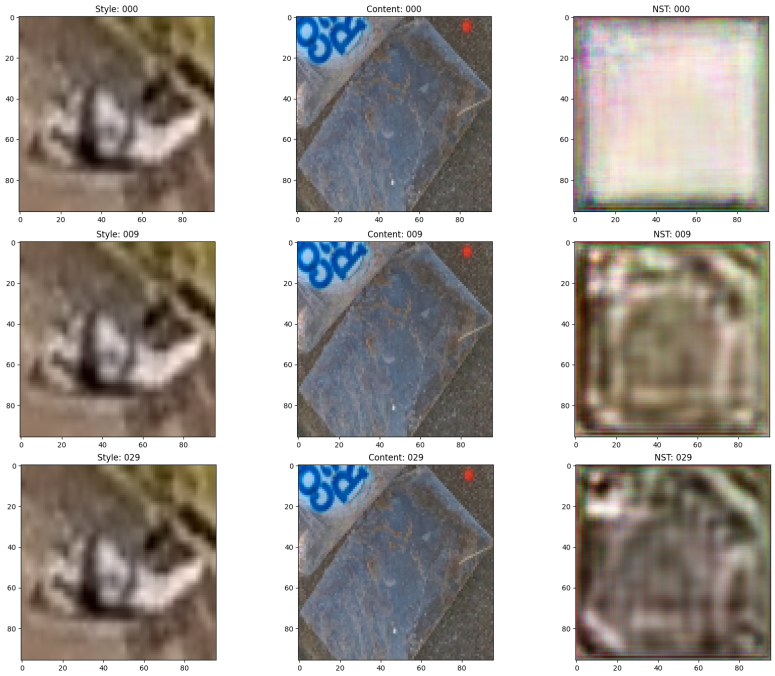


Figure 3.24: Showing style transfer training progress (NST image sample) through callback on epoch 1,10 and 30.

Table 3.5: Table showing the training data in each generated style-transferred dataset. On the left of each item (|) is the image count, and on the right is the instance count (litter samples).

| **Data** | **Train** |
|---|---|
| Style Random Placement | 10773\|33135 |
| Style Training Point Placement | 10773\|33095 |
| Style ROI (SAM) Placement | 10773\|32959 |

Table 3.6: Table showing the training data in each generated dataset using the external supervisely-roads and the optimum pasting solution. On the left of each item (|) is the image count, and on the right is the instance count (litter samples).

| **Data** | **Train** |
|---|---|
| External-dataset synthesised | 8907\|18824 |
| External-dataset synthesised w/ blending | 8895\|18684 |

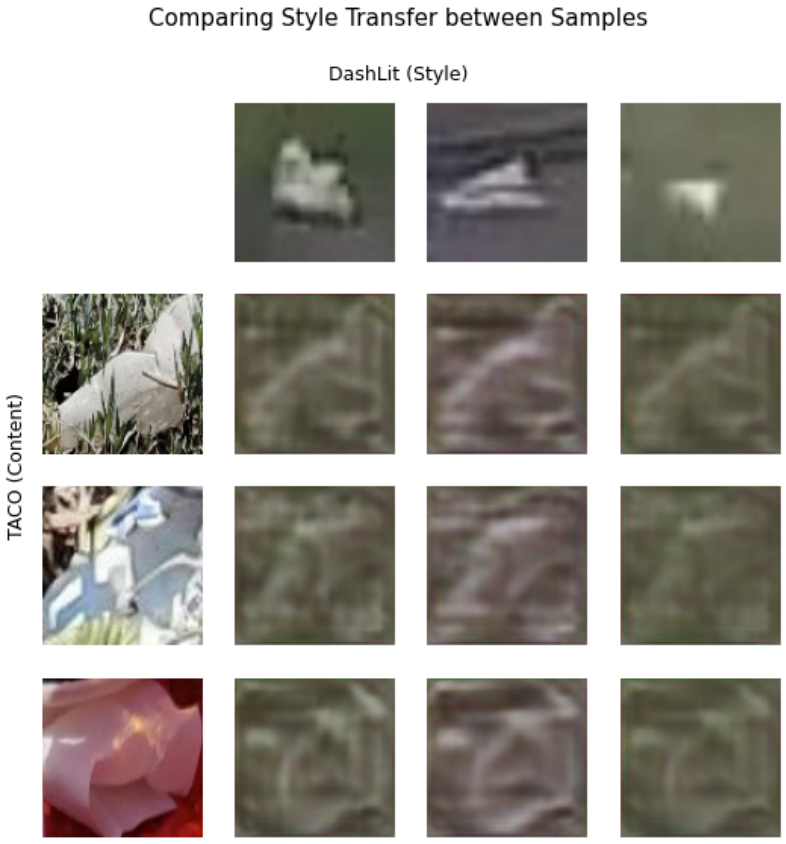


Figure 3.25: Variation in TACO samples with random style transfered from the DashLit dataset.

# Chapter 4

# Software Development

## 4.1 Software Development Methods

The solution created for this project was a collection of tools that come together to create a dataset creation program. Alongside this were a number of miscellaneous scripts for dataset management, model training and research insight.

As the main software solution- a dataset generator- had a defined output, and for the cases of the investigative research, only had one user- a waterfall method of development was decided on (also due to possible absence of continuous design & development post-thesis). A diagram showing this development flow can be seen in figure 4.1.

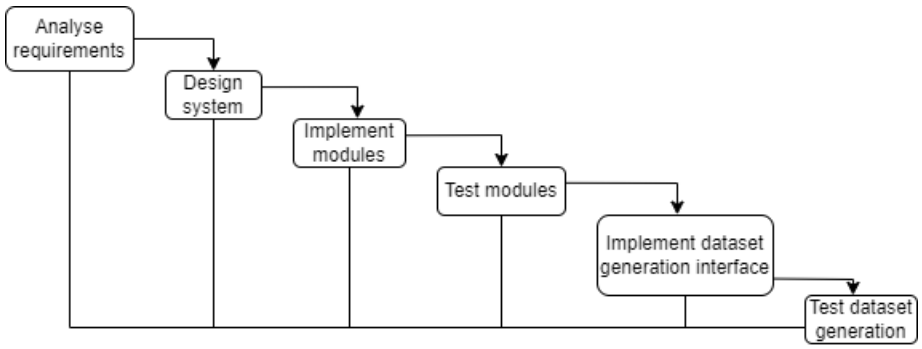


Figure 4.1: Flow chart displaying adaptation of Waterfall software development methodology.

## 4.2 Software Design Documentation

The dataset generation toolbox was brainstormed in figure 4.2. The solution was broken down into elements.

The first element was litter dataset loading. Loading of the TACO dataset would involve cropping each sample in each supplied image by its bounding box and cropping its relevant mask in the same fashion. Loading DashLit litter samples would only be necessary in the case of style transfer, and would save samples in frame images cropped by their bounding boxes.

The second element would be the optional style transfer part of the solution. The AdaIN model would need to be defined in one file, and the code for loading and compiling it with a saved weights checkpoint in another.

The third element would be the contextual placement options. These would include a code file that deals with segmentation context (SAM, the best tested model), a code file that manages loaded training points, and a file that supplies pasting functions.

Finally, a file with the creation functionality would be the main file for generation. This would allow complete abstraction from the other files, in that the options for litter placement and the relevant data would be given and the output would be the synthesised datasets. However, the other files may be used elsewhere for data processing if needed and so can be used on their own also.

Once python was decided on as the programming language, a UML-like diagram displaying the end solution in further detail, including file, class, field(variable) and method(function) names was created as seen in figure 4.3.

## 4.3 Toolsets and Machine Environments

A complete breakdown of programming tools and their versions is displayed in table 4.1 and GPU environments in 4.2

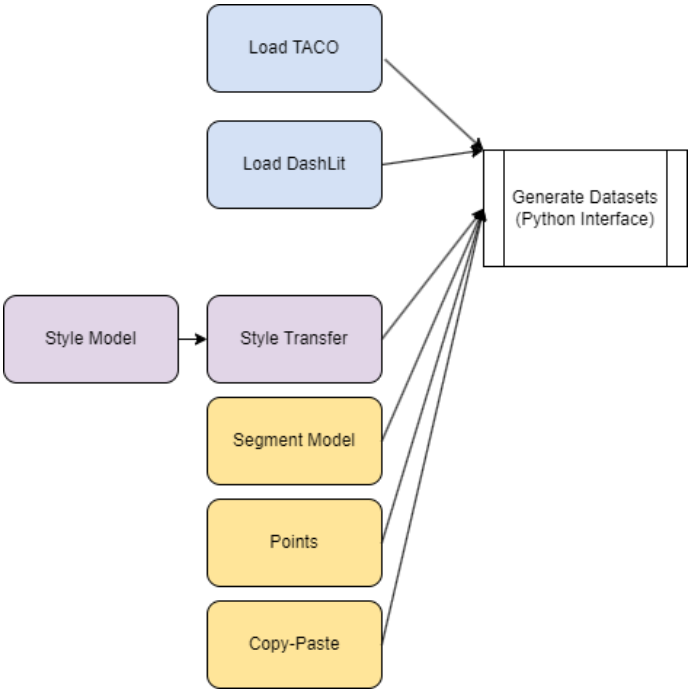


Figure 4.2: Drafted dataset synthesis toolbox application structure.

### 4.3.1 Programming Language & Libraries

The python (Van Rossum and Drake Jr, 1995) library was chosen as the programming framework due to its concise operating system management 'os' and 'random' native libraries, and heavy use in the field of machine learning, supplying required libraries for model training and inference.

The integrated development environments (IDEs) used for python development were Visual Studio Code (VS-Code) (Microsoft, 2023) and Jupyter Notebook (Kluyver et al., 2016). Both environments are free and open source. VSCode had an integrated version control management system, where code changes could be seamlessly committed to the remote GitHub repository (GitHub, 2023), an industry standard. It also contains Python syntax highlighting and suggestions, aswell as break-point functionality for debugging. Both VSCode and Jupyter have ssh protocol functionality (the former using an extension) in order to configure remote files on GPU cluster servers. VSCode was used to develop the dataset synthesis toolbox and write dataset processing scripts, while Jupyter was used to train models and test modular com-

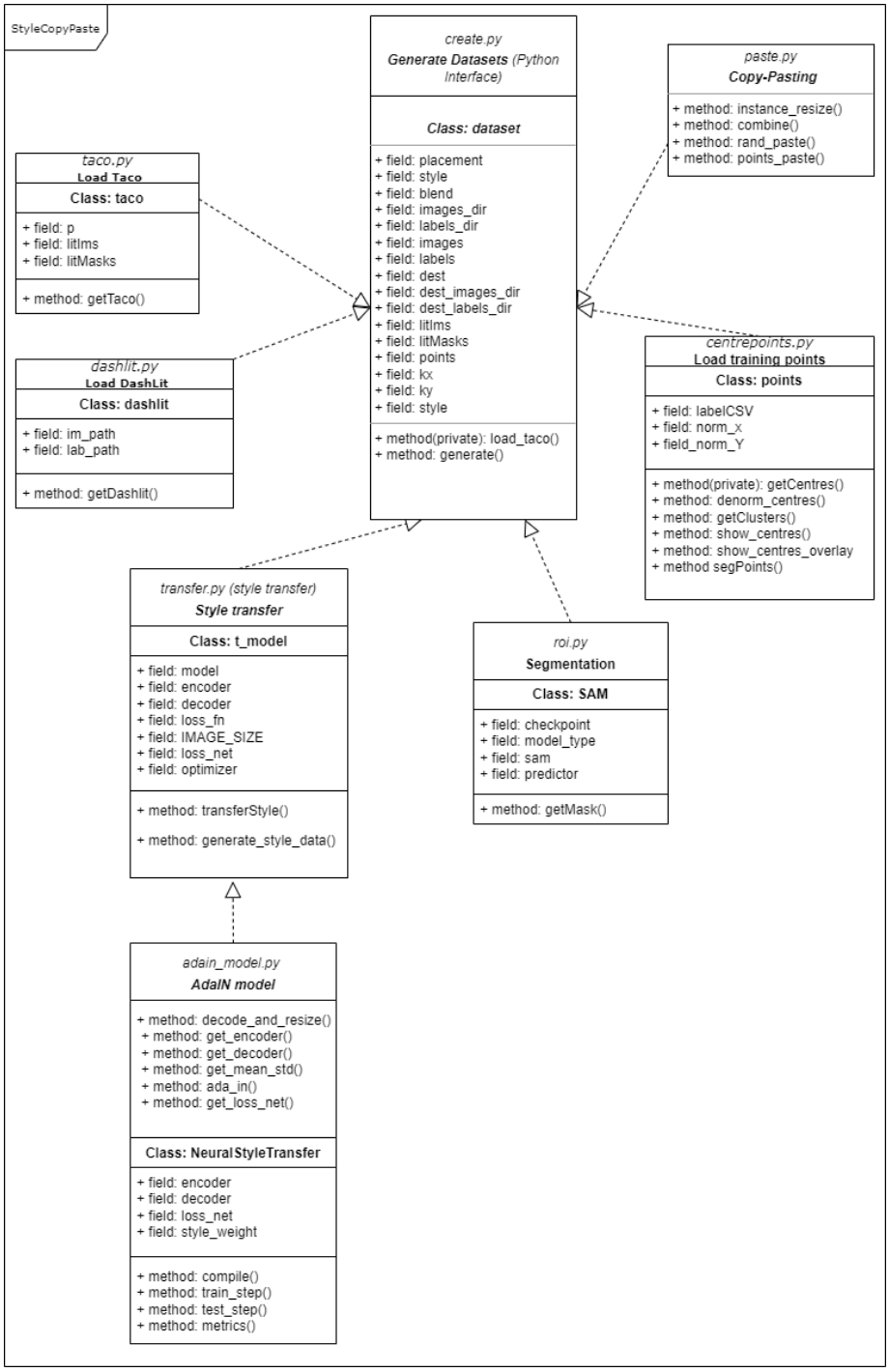


Figure 4.3: UML-like graph showing full dataset synthesis toolbox application structure.

ponents of the program. Jupyter is sufficient for this due to being able to run and output code elements in cells, also allowing for inline image and figure displaying.

Data manipulation libraries were required to work with image and comma seperated value (csv) data. NumPy (Harris et al., 2020) and Pandas (The pandas development

team, 2020) were used for these uses respectively. NumPy arrays are more memory efficient than Python lists due to being stored in a contiguous block of memory and advanced indexing and slicing allows managing images and data matrices of many dimensions easily. The library comes with a suite of mathematical functions to use on these arrays, as well as being able to broadcast custom functions over them. Pandas allowed for easy reading of csv files which store annotation information, and individual columns and rows can be selected naturally using its dataframe structures. Pandas also works well with NumPy as structures can be converted from arrays to dataframes with ease.

The Pycocotools tool is a python API created to manage the COCO dataset (Lin, Maire et al., 2014). It can also be used to manage datasets in COCO's json annotation structure, and thereby it was used to manage and extract information from the TACO dataset, being in the same format.

OpenCV (Bradski, 2000) was used for image processing. OpenCV has extensive image processing tools, such as image reading, colourspace conversion and morphological operations. It housed tools necessary for the copy-paste strategy, as well as comparing segmentation strategies.

Scikit-Learn (Pedregosa et al., 2011) was used in reducing training points using K-Means Clustering due to its neat algorithm abstraction in Python.

Matplotlib (Hunter, 2007) was used for displaying figures for data analysis and black box testing functionalities through visual output. The library allows for displaying images comprehensively in grids using axis logic, and can handle plotting numerical data with much labeling control, housing many graph types.

Two machine learning libraries used were Keras (Chollet et al., 2015) , a high-level API for TensorFlow (Martín Abadi et al., 2015), and PyTorch (Paszke et al., 2019). Keras allowed for defining the AdaIN model due to its modularity and abstraction from unnecessary TensorFlow configuration. PyTorch was required as it was used by both the Segment-Anything model (SAM) (Kirillov et al., 2023) and YOLOv8 model (Jocher, Chaurasia and Qiu, 2023) development frameworks.

Table 4.1: Table showing the programming toolsets & versions.

| Package | Version |
|---|---|
| **Python** | **3.9.16** |
| **Jupyter-Notebook** | **6.5.2** |
| NumPy | 1.23.4 |
| Pandas | 1.4.4 |
| OpenCV | 4.8.0.74 |
| Scikit-Learn | 1.0.2 |
| Matplotlib | 3.6.2 |
| Tensorflow (Keras) | 2.10.0 |
| PyTorch | 1.12.0 |
| Ultralytics (YOLOv8) | 8.0.81 |
| Segment-Anything | 1.0 |
| Pycocotools | 2.0.6 |
| Tqdm | 4.64.1 |

For monitoring YOLOv8 training, including performance metrics and graph output, TensorBoard (Martín Abadi et al., 2015) was used locally while 'Weights & Biases' (wandb) (Biewald, 2020) was used for remote monitoring. Both solutions house concise graph auto-creation allowing comparisons of loss and precision graphs between experiments, aswell as the option to download graph data as a CSV file for reformatting.

The Tqdm library was used for its loading-bar functionality in Python. It's implementation wraps around iterative statements, and allowed for a loading bar giving the progress of the status of dataset generation. It also subsequently acted as a debugging tool.

### 4.3.2 Computing Environments

While dataset generation mostly occurred using a modern CPU capable of running Python, training machine learning algorithms on GPUs was necessary due to time efficiency.

An Nvidia-GPU server (NVIDIA, Vingelmann and Fitzek, 2022) running Ubuntu 18.04.6 (Petersen, 2018) housed Titan-V graphical processing units supplied by LoVE (University of Lincoln, 2023). These were used to run YOLOv8 detection experiments limitlessly. There was an issue with the amount of storage on the servers at

Table 4.2: Table showing the GPU environments.

| GPU | Cuda | Usage |
|---|---|---|
| V100 | 12 | Generating datasets using segment-anything model |
| Titan-V | 11.2 | Running YOLOv8 object detection experiments |

the time of experiment however, and so dataset creation could not happen on-device. Whereas most dataset creation could occur on a CPU, if the Segment-Anything model is used, it required GPU use as it reduced synthetic image creation from 3 minutes to 2-3 seconds.

To rectify the storage issue, Google Colab (Bisong, 2019) was used, linked to a Google Drive with which the created datasets could be deposited. Colab pro was subscribed to for a month to allow for the use of their supplied V100 GPUs. Both the LoVE server and Colab housed these different GPUs along with different CUDA versions, but these were pre-installed (v11.2 and v12 respectively).

To maintain an environment on the LoVE server, Docker (Merkel, 2014) was used, allowing a YOLOv8 image to be run as a container for seamless detection model training. The relevant training set was stored on the platform, with the directory mounted to the container. Ports were forwarded such that training could be supervised on Jupyter, run graphs could be viewed locally on TensorBoard, and VSCode could be used to edit repository code. Datasets were transferred from a local PC using FileZilla. Unfortunately, the forfeit of using Colab was that dataset creation had to be supervised, as runtimes are not persistent due to enforced time-outs - the con to essentially "renting" a GPU and runtime.

## 4.4 Implementation

The implementation will be discussed as per the final program design in figure 4.3.

### 4.4.1 Dataset Generation

#### Loading TACO

Loading of TACO litter to be pasted occurred in *taco.py* which contained the class **taco**. The object can be instantiated by providing a path to the TACO data directory. The class comes with litIms and litMasks as empty list fields, to be filled when using the contained function *getTaco()*.

The function *getTaco()* takes a list of image ids, such that certain images from the TACO dataset can be loaded. PyCocoTools' *COCO()* function is used to read the taco annotations file from the data path given. Images are processed by id in a loop. Within this loop, the litter samples and their masks are cropped according to their bounding box. Cropped masks are saved as-is, and cropped litter images are "cut out" using their mask using the OpenCV bitwise operation and saved. Both are stored in memory to the litIms and litMasks variables.

#### Loading DashLit

Loading of the DashLit data if needed for style transfer occurs within *dashlit.py* which contained the class **dashlit**. An object can be instantiated with the path to the images and path to annotated labels in YOLO format. The class contains a list field litIms which will be filled by the contained function *getDashLit()*.

The function *getDashLit()* takes slice indices (a start and end to a relevant amount of dashcam frames). These are looped through, reading the image name and noting the relevant label name by changing the ".jpg" to ".txt" - as allowed by the YOLO format. By processing annotation lines in the label file, litter samples can be cropped from the frames by converting the yolo box centres into a format that can be used for index slicing.

For conversion, it is important to note how YOLO annotations are formatted. For each annotated image, there is a relevant text file. For each annotated instance in this image, the file has a line. The line is formatted as:

*<class id> <x centre> <y centre> <width> <height>*

...where *class id* is always '0' (one litter class) and the other variables relate to the bounding box centre point, width and height. These values are normalised according to the height and width of the relevant image.

For conversion to slice indexes, it is to be noted the syntax for NumPy 3D slicing (images have width, height and channel dimensions).

*imageArr[row1:row2, col1:col2, :]*

In images, "row" values correspond to the "y" height axis while "column" values correspond to the "x" width axis.

For conversion, therefore, bounding box information is multiplied by the relevant image height or width to de-normalise. Then, xmin and ymin (top-left bounding box point) are then inferred by subtracting half of the bounding box width from x-centre and half the bounding box height from y-centre. These give column 1 and row 1 values. The end point for each dimension slice is inferred by adding the width or height to these figures, getting the other side of the bounding box.

The cropped litter instances are then saved in memory to the litIms variable.

### AdaIN Model

The AdaIn model for style transfer was defined within *adain_model.py*. It contains functions for decoding and resizing to the model's format, defining the encoder (loading VGG19 from the Keras applications API), defining the AdaIN module (by computing mean and variance), defining the decoder, and loss. The model itself can be compiled via the **NeuralStyleTransfer** class, which concatenates all the network's modules.

### Style Transfer

The *adain_model.py* model definition is abstracted in the file for style transfer, *transfer.py*. A model can be instantiated as an object under the containing class **t_model**, supplied with the image size for transfer and weights checkpoint that was trained according to the specific content/style datasets. The containing function *transferStyle()* takes the style image and content images, resizes them appropriately,

and feeds them into the model to return the reconstructed neural style transfer (NST) image. The function *generate_style_data()* is more higher-level, using the *transferStyle()* function to return the NST image 'cut out' by a returned transformed mask (from the original TACO mask) using OpenCV's 'bitwise AND' functionality. It deals with the relevant sizing, dimension and tensor conversions.

### Loading Training Points

Loading of the litter bounding box centres "training points" occured within *centre-points.py* which contained the class **points**. A points object can be instantiated by supplying a path to a csv file containing annotation data in the form:

*[xmin] [ymin] [xmax] [ymax] [width] [height]*

...where *height* and *width* refers to image dimensions, not bounding box dimensions. These values are read into a Pandas dataframe, then converted into NumPy arrays. Bounding box centres are computed using the private *___getCentres___()* function on object instantation. Normalised 'x' and 'y' centre values can be computed before being saved as variables using the following equations:

$$norm(xcen) = \left(\frac{xmin + xmax}{2}\right) * \left(\frac{1}{width}\right) \tag{4.1}$$

$$norm(ycen) = \left(\frac{ymin + ymax}{2}\right) * \left(\frac{1}{height}\right) \tag{4.2}$$

The *denorm_centres()* function reverses normalisation by multiplying points by the inputed images dimensions. This allows for points to be placed in relevant location to the frame if they are supplied from a dataset of various size images.

*getClusters()* uses Scikit-Learn's K-Means Clustering implementation to return cluster centres from the object's own computed bounding box centres.

The *show_centres()* and *show_centres_overlay()* functions are used for displaying points in a figure; the former as a heatmap and the latter for overlaying onto an image.

*segPoints()* takes a mask generated using a segmentation method and converts it into arrays of x and y points using NumPy querying.

### Segmentation

Segmentation used the SAM model occurs via the **SAM** class in *roi.py*. This model class can be instantiated with the checkpoint of the trained Segment-Anything-Model supplied by its creators on GitHub, and can optionally be used with a GPU by setting the parameter "device = cuda". Most of the logic is abstracted behind the installed SAM package, but the function *getMask()* sets an image to the predictor, and uses points arrays 'pointsx' and 'pointsy' as a prompt for returning the best mask. It was found that SAM may return some very small noise in its mask, and these items were removed using morphological erosion of a 5*5 ellipse over 5 iterations.

### Copy-Pasting

Pasting logic was held within the *paste.py* file. *instance_resize()* was used to resize litter instances and could be used to resize by height or width and keep aspect ratio. It used OpenCV's resizing capabilities, with area relation as the strategy due to it maintaining visuals when shrinking items. Nearest neighbour or bilinear methods may introduce detail loss.

The *combine()* function copied litter instances onto the relevant area in the synthesised frame, and was the intergral function in all contextual placements. It initially took the litter image and its mask aswell as the centrepoint of placement (separate x and y values) as parameters.

The foreground is considered the litter instance and background is computed using the centre-point of placement. The background is calculated as the bounding box region produced when the litter is placed:

*background = imageArray[y-litter_height:y, x:x+litter_width]*

...where $x$ and $y$ are litter centres. The foreground and background are split into their relevant colour channels. The litter mask was inverted and used as a guide for cutting. The combination of foreground and background occurred through OpenCV's

"bitwise AND" functionality per colour channel. These channels are merged back into a new image. The bounded merged section is then "pasted" back onto the full frame using slice indexing.


Figure 4.4: Intermediaries of pasting a litter sample in the top-left of a frame (sky).

Combining foreground and backgrounds became less trivial to program when random rotation was introduced. Both the supplied mask and litter image had to be rotated, and then the height and width of the litter instance had to be re-assigned to the new rotated instance.

Later, blending functionality was added. OpenCV's *seamlessClone()* functionality was used to implement the Poisson method. It could be enabled in a similar fashion to random rotation - by using a boolean flag.

The function *rand_paste()* used the *combine()* function to paste the litter samples in random points. It takes an array of TACO images and masks as a parameter, as well as the background frame in question. The largest litter instance diameter is resized to a random size between 20 and 50 pixels, maintaining aspect ratio. Random x and y points were selected within the region of the image space, with a safety mechanism that did not allow pastes to pass image edges. The safety mechanism always subtracted the largest dimension of the litter from the random x and y value generated. This in turn means that pastes did not include edge cases where litter

would be occluded. The function returned coordinate and dimension info of the paste if the litter is rotated, as this may have changed.

*points_paste()* works in a similar fashion to random paste, except it pastes along supplied x and y points. A random pair of x & y points is selected. The points are assumed to be bounding box (or segmentation mask point) centres.

#### Dataset Generation

*create.py* is the interface which supplies functions that abstract the other files, and is used for verbose dataset synthesis using a little amount of code. Datasets are to be generated by instantiating the contained class **dataset** by supplying the object with the source directory (of YOLO images and labels), the destination directory (to save synthesised images and labels), and options. The options include the type of placement as a string ("random", "points" or "seg"), supplying the "style" parameter with a checkpoint if transfering, supplying the "pts" parameter with a path to training points csv file if using, and setting "blend" to **true** if implementing Poisson blending.

Once the **dataset** class is instantiated as an object, the contained function *generate()* can be called by supplying it with a path to the litter sample dataset (TACO). The dataset is then generated relevant to the optional flags supplied at instantiation. Error handling skips image creation if there is an error with one of the litter samples (if it is unsupplied as 'NoneType' according to the relevant id for example).

An auto-annotation feature occurs that formats the new litter placement location's normalised centre, width and height. It prints this as a new line onto a copy of the original YOLO label file for that frame, then saves in an adjacent directory to the new image in YOLO format.

### 4.4.2 Miscellanous Scripts

Scripts were created to pre-process files, train models and test functionalities.

The TACO dataset had many classes and was in COCO format. To convert (for use in the DashLit-TACO baseline), annotations were loaded using the Pycocotools

API, normalised for YOLO, then printed to text file labels. Classes were all reset to '0' (single litter class).

Training both AdaIN and YOLOv8 detection models happened via ipython notebooks on the Jupyter Notebook platform. This allowed for displaying various configurations per code block, as well as examining progress mid-training. YOLOv8 in particular was provided via the Ultralytics library, a highly verbose detection network configuration API, with relevant abstract supplied functions suitable for notebook runs.

After dataset generation, instance and image counts could be calculated using the produced yolov8 label files - simply counting the files returned image counts, and counting the lines per file returned litter instance numbers.

## 4.5 Testing

Testing the main generator took place using the black box strategy, such that both generated images and their relevant labels were adequately produced.

For each module; pasting, segmentation and style transfer, generated frames were displayed indefinitely until deemed satisfactory.

For analysing the labels, generated image and label paths were loaded, and bounding boxes were visualized after conversion from YOLO to a format suitable for OpenCV's 'rectangle' display module. The YOLOv8 Ultralytics training procedure also displays bounding-box visualizations on training images as a debugging feature.

# Chapter 5

# Results

This chapter will run through the results of the investigation into the impact of context and style transfer on copy-paste image generation for dashcam litter detection. Experimental results are split into that of the baseline detection results, results from changing contextual placement, results from implementing style transfer, and results from using external data frames for generation. Results are presented visually in terms of image generation output, distribution focal loss (DFL) graphs are displayed for insight into detection model training, and detection performance measures are displayed as explained in 3.2.8. A full detection performance comparison table can be seen in 5.5. As previously mentioned, precision and recall are calculated at a 0.3 IoU threshold.

## 5.1 Baseline detection

Table 5.1 contains baseline detection results on the test set of the DashLit and DashLit-TACO (simple combination) datasets when the training sets are used for training YOLOv8L. Training and validation DFL loss graphs are seen in figure 5.1.

Table 5.1: Table showing baseline detection results.

| **Dataset** | **Recall** | **Precision** | **AP @ 0.5** | **AP @ .5-.95** |
|---|---|---|---|---|
| DashLit | 0.708 | 0.759 | 0.767 | 0.323 |
| DashLit-TACO | 0.694 | 0.78 | 0.76 | 0.313 |

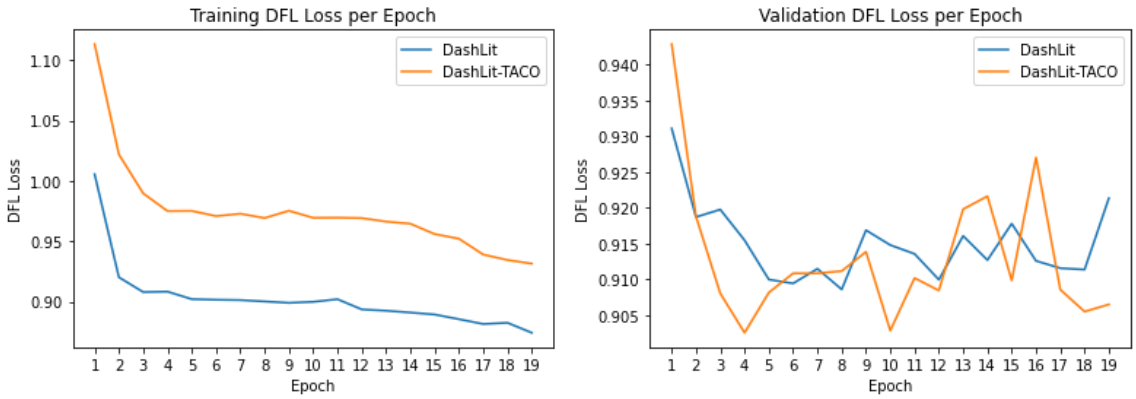


Figure 5.1: Graph showing DFL loss per epoch of training YOLOv8L on baseline datasets.

## 5.2 Pasting Context

Table 5.2 contains context placement experimental detection results on the test set of the DashLit dataset when the generated training sets are used for training YOLOv8L. Training and validation DFL loss graphs are seen in figure 5.3. Cropped samples from the context-weary images generated from the developed toolset are shown in figure 5.2.

(a) Random placement gen.

(b) Training-point placement gen.

(c) ROI placement gen.

Figure 5.2: Comparing cropped output of the context-querying generated datasets.

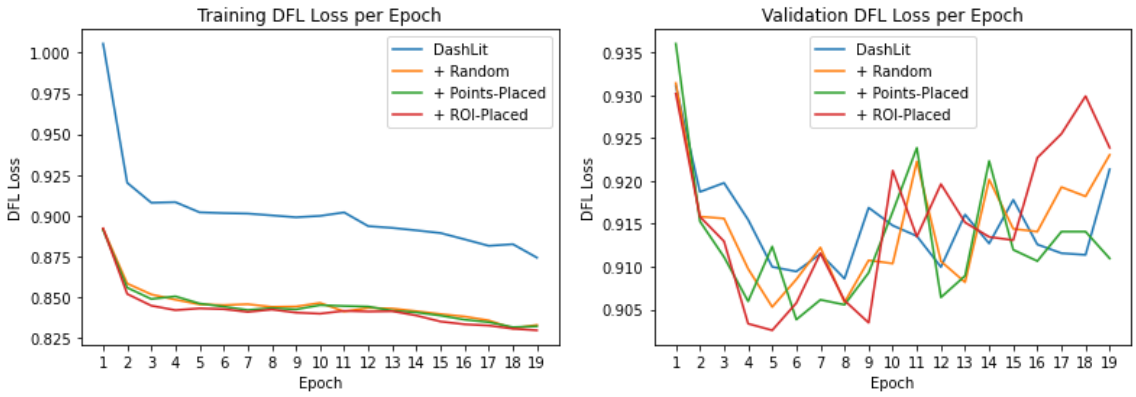


Figure 5.3: Graph showing DFL loss per epoch of training YOLOv8L on context-generated datasets.

Table 5.2: Table showing data generation detection results under different contextual placements.

| Dataset | Recall | Precision | AP @ 0.5 | AP @ .5-.95 |
|---|---|---|---|---|
| Random | 0.705 | 0.751 | 0.751 | 0.311 |
| Points-Placement | 0.698 | 0.757 | 0.759 | 0.317 |
| ROI (SAM) Placement | 0.703 | 0.748 | 0.752 | 0.31 |

## 5.3 Style Transfer

Table 5.3 contains style-context placement experimental detection results on the test set of the DashLit dataset when the generated training sets are used for training YOLOv8L. Training and validation DFL loss graphs are seen in figure 5.5. Cropped samples from the style-transferred context-weary images generated from the developed toolset are shown in figure 5.4.

Table 5.3: Table showing data generation detection results under different contextual, style transferred placements.

| Dataset | Recall | Precision | AP @ 0.5 | AP @ .5-.95 |
|---|---|---|---|---|
| Random [NST] | 0.688 | 0.749 | 0.747 | 0.309 |
| Points-Placement [NST] | 0.741 | 0.747 | 0.756 | 0.311 |
| ROI (SAM) Placement [NST] | 0.697 | 0.751 | 0.748 | 0.301 |


(a) Random placement gen.

(b) Training-point placement gen.

(c) ROI placement gen.

Figure 5.4: Comparing cropped output of the context-style-querying generated datasets.

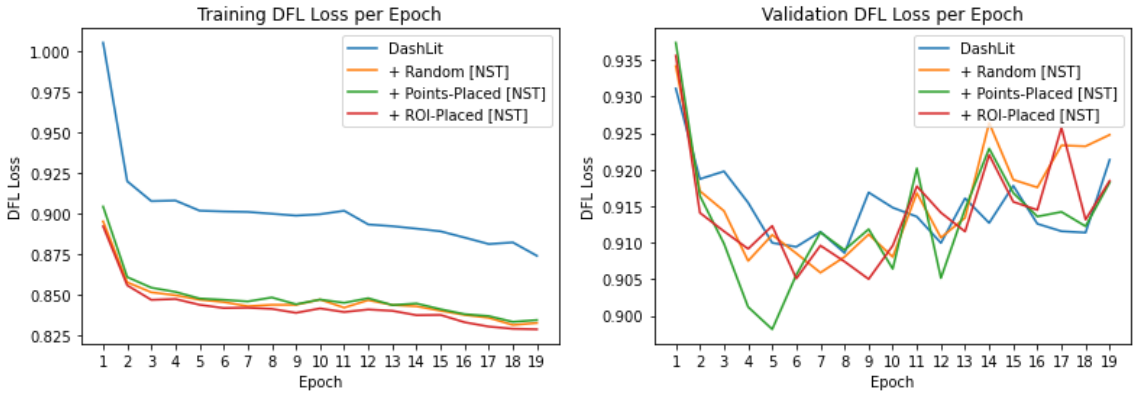


Figure 5.5: Graph showing DFL loss per epoch of training YOLOv8L on neutal style transfer generated datasets.

## 5.4 External data & Blending

Table 5.4 contains detection results on the test set of the DashLit dataset when the external-frame generated training sets are used for training YOLOv8L. Training and validation DFL loss graphs are seen in figure 5.7. Cropped samples from the

blend/non-blend images generated from the developed toolset are shown in figure 5.6.

(a) External-frame placement gen. (b) External-frame blending placement gen.

Figure 5.6: Comparing cropped output of the external frame/ blend-querying generated datasets.

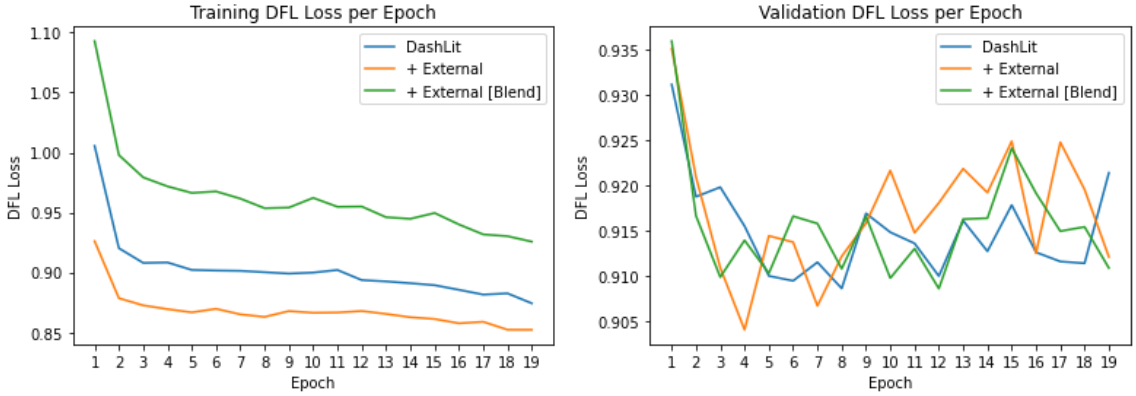


Figure 5.7: Graph showing DFL loss per epoch of training YOLOv8L on external-frame generated datasets.

Table 5.4: Table showing training-point data generation detection results using frames from another dashcam dataset and Poisson blending (no style transfer).

| **Dataset** | **Recall** | **Precision** | **AP @ 0.5** | **AP @ .5-.95** |
|---|---|---|---|---|
| DashLit-ExternalGen | 0.704 | 0.766 | 0.761 | 0.317 |
| DashLit-ExternalGen [Blend] | 0.707 | 0.764 | 0.767 | 0.326 |

## 5.5 Full Results

Table 5.5: Table showing all detection performance results, with the highest metrics highlighted in **bold**.

| **Dataset** | **Recall** | **Precision** | **AP @ 0.5** | **AP @ .5-.95** |
|---|---|---|---|---|
| DashLit | 0.708 | 0.759 | **0.767** | 0.323 |
| DashLit-TACO | 0.694 | **0.78** | 0.76 | 0.313 |
| Random | 0.705 | 0.751 | 0.751 | 0.311 |
| Points-Placement | 0.698 | 0.757 | 0.759 | 0.317 |
| ROI (SAM) Placement | 0.703 | 0.748 | 0.752 | 0.31 |
| Random [NST] | 0.688 | 0.749 | 0.747 | 0.309 |
| Points-Placement [NST] | **0.741** | 0.747 | 0.756 | 0.311 |
| ROI (SAM) Placement [NST] | 0.697 | 0.751 | 0.748 | 0.301 |
| DashLit-ExternalGen | 0.704 | 0.766 | 0.761 | 0.317 |
| DashLit-ExternalGen [Blend] | 0.707 | 0.764 | **0.767** | **0.326** |

# Chapter 6

# Discussion and Evaluation

## 6.1 Evaluation of Results

### 6.1.1 Generated Output

The composited images produced by the developed synthesis solution are subject to commentary.

Random placement of litter items is distinct. This form of image production does not discriminate on where litter is placed - in the sky, on the dashboard, and items appear visually out of place. It may be difficult to distinguish between the "region of interest" SAM-mask placed litter items and those placed according to training litter locations. Training-point placed litter has a pattern that exists along the edges of the road, sometimes stretching past curbsides. ROI-placed litter is less selective by location, however appears just as realistic, stretching further into road centres. Both of these methods of placement seem visually feasible locations for litter to end up naturally.

Style-transferred litter instances may be more difficult to spot in images with the naked eye, sharing similar colours and intensities with grass and curbsides. When they appear within verges they more closely resemble litter items seen within the organic DashLit dataset, with its frame compression different to that of the unprocessed TACO images. When these style-transferred pieces are placed in the sky, dashboard (i.e. in the random-context samples) and on road centres, they now seem visually out-of-place, as if the item has an unnecessary camouflage effect.

The road environment in the external "supervisely-road" generated sets is of contrast to DashLit data. These foreign images have a lower light intensity and seem more compressed, which makes placed litter instances stand out even further. The Poisson seamless blending strategy appears to partially remedy this, as placed objects seem to appear natively in some backgrounds, especially grass-sides. However if the litter item has colourful high-resolution packaging, there seems to be a "halo" effect, where the boundary is overtly blurred.

### 6.1.2 Detection Performance

On viewing the detection results of DashLit and DashLit-TACO, it appears that the simple addition of the 1500 TACO images in training mostly degrades performance. Their inclusion reduces recall at a 0.3 IOU threshold, and reduces AP at both higher thresholds. There is in precision at 0.3 IOU from 0.759 to 0.78 however, possibly indicating that introducing these varied backgrounds and diverse litter samples aids the model to detect against false positives at lower thresholds.

No contextual placement strategy improves performance over the baseline outcomes. However, points-placed litter images give rise to the highest precision and average precision values of all context experiments. This may indicate that test data has a very similar litter location distribution to the training litter. Interestingly, random placement and ROI placement give rise to slightly higher recall values over points placement, possibly training the model to look for litter with different background appearance (while reducing precision, possibly by widening the search space for litter location predictions).

Transferring style to placed instances had no effect on increasing detection precision or average precision over either the base context experiments or the baselines. However, transferring style has a large increase in detection recall when applied to points-based placement, from 0.708 in the DashLit baseline to 0.741. This could indicate that style transfer is a highly beneficial method to increase recall when placing foreign litter objects, but must comply to objects in very similar locations from which the style was "learned".

Using unseen frames from another dashcam dataset seems to increase precision at a low IoU threshold while having little change from the recall and average precision baseline. The blending experiment using this data is the only such that matches the average precision at a 0.5 IOU in the baseline. Blending litter in external datasets has a slight average precision increase at high IOU thresholds, and also increases precision over the DashLit baseline at the low 0.3 threshold. This could indicate that introducing litter placements in unseen data could boost tracking accuracy, as precision is increased over all examined thresholds without negating recall.

### 6.1.3 Loss Graphs

On viewing the training and validation DFL loss graphs, a pattern emerges. Training loss converges steadily while validation loss converges more noisily. This may indicate that more quality data may be required. In the training loss graphs for the context and style experiments, loss appears to be much lower than baseline, while is similar to baseline in the validation graphs. This could indicate that overfitting is occuring, and may be due to the extra synthesised data being added to the original data for training - where frame backgrounds are duplicated. This does not seem to be the case with the external frame synthesised datasets, whereby the blended training losses are higher than the baselines, yet the validation converges similarly. This could indicate that introducing these different backgrounds (from the same dashcam-oriented domain) can help the model generalise, which is backed up by the stable recall and average precision over various thresholds.

## 6.2 Reflection

Some improvements to this project could be made, due to both the time limitations of the MSc thesis period and the continuous development of my research skills.

### 6.2.1 Amendments to Research

While results from these experiments have given insight into data synthesis in the domain of litter detection, there can be improvement in some aspects of research.

I would suggest introducing the blend methods to all contextual and style based synthesis methods, to examine whether it would improve detection performance as in the external dataset frame research. It may also be necessary to find a better method of verge segmentation using supervised deep learning methods, which may give rise to more quality synthesised datasets for the "region of interest" contextual composition strategy. However, as the points-placement strategy seemed to be quite efficient, it may be a case of Occam's razor, whereby the simple solution indicated by the data itself may be the most beneficial. Furthermore, a more dynamic strategy to use only the most realistic points out of the training data for each specific frame may provide further insight.

Concerning litter sizing, although a strategy mimicked sizing seen in the base DashLit dataset, perhaps a smarter approach could increase dataset quality. A proposed solution could be sizing based on how far the litter is deemed to be away from the capturing vehicle, introducing a size interpolation mechanic as the artificial litter appears near.

With regards to dataset training, perhaps it would be useful to experiment with training the detector on *only* synthesised frames, such that organic ones with the same background do not remain in training. This could either reduce overfitting, or negate the possible overcrowded-learning that training with additional synth frames would provide.

On revision, although style transfer quality was partially assessed through the downstream detection task, it may be holistic to determine the transfer performance prior by less subjective means than simple visual inspection by the researcher. Briakou et al. (2021) reviews the use of human judgement of performance by groups of study participants, as they deemed it the most popular gauge of style transfer quality (69 out of their 97 reviewed papers). While this paper focused on style transfer in terms of language, the judgement factors could be transferable into the domain of vision; "style" meaning the creativity involved in output, "meaning" quantifying the content preservation factor and "fluency" meaning how natural or true-to-life output looks. Vision-specific human studies involve simple style ratings from a numerical range (Z.

Wang et al., 2021). H. Chen et al. (2023) quantifies quality for visual style transfer as content preservation (CP), style resemblance (SR), and overall vision (OV), mimicking the factors seen in quantifying language style transfer. They can then rank NST algorithms by their subjective scores using the Bradley-Terry model. They go a step further and also analyze objective metrics such as examining "Gram-Based" features for style (texture similarity) and "Difference of Gaussian" (DoG) response features for content (edge similarity). Perhaps examining these metrics in terms of artificial litter production quality could improve the quality of the research's scientific methodology.

### 6.2.2 Amendments to Software

The software solution (dataset generator) could be improved in its implementation. Firstly, TACO instances (and DashLit if implementing style transfer) were loaded in full and saved in memory before placement. This meant that high RAM was required to perform data synthesis. It meant that when using SAM for synthesis in Google Colab, the GPU had to be purchased with high memory. Perhaps a method of loading litter instances by TACO image path per image may be more efficient for reducing memory resources, with higher time complexity per image creation as forfeit.

Another option could be pasting litter items in frames during training time, allowing different locations of placement at every epoch, similar to how Jocher, Chaurasia and Qiu (2023) implement online augmentation for Yolov8. This would replace this work's offline method, however would make the synthesized images more difficult to study.

Loading training point data is carried out through loading an annotation csv file. It may improve usability to extend the option to read annotations in the format of the supplied native dataset also, such as the YOLO, COCO, or Pascal VOC format. These options may also be useful for synthesised label output to extend usability to other projects.

Litter placement also does not consider obscuring edge cases- simply disallowing

placements which would "fall off" the image. This could be remedied by slicing the litter sample relevantly in accordance to edge overlap. This artificially-obscured information may even benefit the model in terms of learning edge cases.

There were multiple storage issues when dealing with dataset generation. The lack of memory on the LoVE GPU server meant that only one dataset at a time could be hosted for training the detector, and therefore would not be of use for SAM dataset synthesis (which motivated moving to Colab). While Colab remedied the storage issue with access to a mounted Google Drive, a less long term costly method would be locating a local PC with reliable GPU, which would also allow for a persistent runtime. There was also an issue with uploading and extracting large datasets in Google Drive, as it stopped loading original DashLit frames needed for synthesis after a certain amount of time. This led to the experiment adding much less synthesised data per run than initially intended. Luckily, the reduced sets still produced insight.

# Chapter 7

# Conclusions

## 7.1 Final Verdict

A software solution was created which can accept different contextual composition options and paste litter instances from source datasets to a target dataset with optional style transfer and blending. Datasets were created using these options to assess for the most feasible synthetic data which would boost the training of a dashcam litter detection solution. A number of discoveries were found.

Simply adding any-context annotated litter data to detection training boosts low-threshold precision whilst slightly reducing recall and average precision. Target-style-transferred litter placed in frame locations seen in training data boosts recall substantially while slightly reducing low-threshold precision and average precision. Segmenting verge regions with a low mask generation performance is not beneficial as a contextual litter placement strategy. Placing litter items in unseen external data frames and blending them into the environment increases precision at a low IOU threshold and increases average precision at high average thresholds, while having negligible negative effects on other metrics.

With regards to the main research hypothesis; *training a neural network with external litter dataset items pasted within original dashcam frames boosts detection performance*, this has partially been supported, in that training-litter placement with style transfer increases recall (to identify more items) and the addition of litter-blended foreign dataset frames boosts average precision over a range of thresholds (to reduce false positives and improve the precision-recall trade-off). The take-away is that

such detection models benefit from varied data that exists from the same domain (road-viewing camera), and style-transfer is only useful in such composition if the pasted pieces are in the exact locations seen in all possible data samples.

## 7.2 Future Work

Future work could address the issues faced in 6.2, but there are also numerous possible research extensions to gather more insight into dataset synthesis for dashcam litter detection.

Deep learning object detection solutions often merit training many samples. Increasing the amount of data in training could positively impact the detection performance. This could occur through further synthesis. During the annotation and processing of dashcam videos, a high amount of frames which do not contain litter are often discarded. These could be used to synthesise more litter frames such that the algorithm learns from more diverse background scenarios. There also exists many open-source litter datasets as discussed in 2.1.1. Further data variety could be introduced by including more of these litter samples, such as the extended version of the TACO dataset. There is also hope that this work furthers resolution of the litter detection problem across domains, such that other datasets could be synthesised to mimic the context of other locations (such as beaches for example).

While the benefits of style transfer for domain adaptation have been shown in this experiment, such research could be extended into that of generative adversarial networks. A generator network could control placement and integration of litter such that the produced images appear "real" to the discriminator network, returning a potential viable new dataset. The addition of random noise could aid diversity in litter and background variation. However it must be known that this possible line of work departs from simply using litter datasets currently available as-is by introducing more 'artificial' aspects of synthesis, and could potentially decrease the amount of transparency in image production.

Advances in detection performance research greatly benefits that of related two-stage

tracking. Two-stage tracking works by first detecting instances by means such as a trained deep learning model then relating same-object-detections between frames as "tracks". The track relation could be carried out by the likes of a Kalman filter with a constant velocity model. Difficulty ensues when detections are missed on some frames, which may be remedied by looking for detections at a lower confidence, perhaps at different intersection over union thresholds. The fact that adding generated litter frames from foreign domain-specific datasets into training increases precision and average precision over multiple thresholds therefore would increase such tracking accuracy. Further work needs to be done however in tackling the tracking complexities in the change of vehicle speed and sway, but successful tracks could allow for efficient litter mapping strategies.

This work also makes way for litter collection strategies using artificial intelligence along with robots. This could reduce the risk to life of human-collection on such busy roadsides. It is hopeful that the outcomes of these related works reach that of safer, cleaner roadsides which remain free of such pollution in the near future.

# References


'Let's Do it' Foundation (2016). *Wade-Ai*. URL: https://github.com/letsdoitworld/wade-ai (cit. on p. 8).

Aboah, Armstrong et al. (2023). *Real-Time Multi-Class Helmet Violation Detection Using Few-Shot Data Sampling Technique and YOLOv8*. URL: https://openaccess.thecvf.com/content/CVPR2023W/AICity/html/Aboah_Real-Time_Multi-Class_Helmet_Violation_Detection_Using_Few-Shot_Data_Sampling_Technique_CVPRW_2023_paper.html (cit. on p. 30).

Adamová, Veronika (Nov. 2020). 'DASHCAM AS A DEVICE TO INCREASE THE ROAD SAFETY LEVEL'. In: *Proceedings of CBU in Natural Sciences and ICT* 1, pp. 1–5. DOI: https://doi.org/10.12955/pns.v1.113 (cit. on p. 3).

Almanzor, Elijah et al. (Dec. 2022). 'Autonomous detection and sorting of litter using deep learning and soft robotic grippers'. In: *Frontiers in Robotics and AI* 9. DOI: https://doi.org/10.3389/frobt.2022.1064853 (cit. on p. 10).

Andrei, Alexandru-Toma and Ovidiu Grigore (Mar. 2023). 'Mean shift clustering with bandwidth estimation and color extraction module used in forest segmentation'. In: *IEEE Xplore*, pp. 1–6. DOI: https://doi.org/10.1109/ATEE58038.2023.10108106. URL: https://ieeexplore.ieee.org/abstract/document/10108106 (cit. on p. 14).

Ashwini, A et al. (Apr. 2023). 'Automatic traffic sign board detection from camera images using deep learning and binarization search algorithm'. In: *IEEE Xplore*, pp. 1–5. DOI: https://doi.org/10.1109/RAEEUCCI57140.2023.10134376. URL: https://ieeexplore.ieee.org/abstract/document/10134376 (cit. on p. 14).

Bae, K et al. (2020). *Anti-Litter Surveillance based on Person Understanding via Multi-Task Learning*. URL: https://www.semanticscholar.org/paper/Anti-Litter-Surveillance-based-on-Person-via-Bae-Yun/a1f69376de7aa59dc7a5050c9d9a525 (cit. on p. 10).

Beucher, Serge and Marc Bilodeau (Aug. 2005). 'Road segmentation and obstacle detection by a fast watershed transformation'. In: *Proceedings of the Intelligent Vehicles '94 Symposium*. DOI: https://doi.org/10.1109/ivs.1994.639531 (cit. on p. 14).

Biewald, Lukas (2020). *Experiment Tracking with Weights and Biases*. Software available from wandb.com. URL: https://www.wandb.com/ (cit. on p. 49).

Bisong, Ekaba (2019). 'Google Colaboratory'. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7. URL: https://doi.org/10.1007/978-1-4842-4470-8_7 (cit. on p. 50).

Bradski, G. (2000). 'The OpenCV Library'. In: *Dr. Dobb's Journal of Software Tools* (cit. on pp. 24, 48).

Briakou, Eleftheria et al. (2021). *A Review of Human Evaluation for Style Transfer*, pp. 58–67. URL: https://aclanthology.org/2021.gem-1.6.pdf (cit. on p. 67).

Chaturvedi, Kunal et al. (Mar. 2023). 'SS-CPGAN: Self-Supervised Cut-and-Pasting Generative Adversarial Network for Object Segmentation'. In: *Sensors* 23.7, pp. 3649–3649. DOI: https://doi.org/10.3390/s23073649. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10098536/ (cit. on p. 13).

Chen, Hangwei et al. (July 2023). 'Quality Evaluation of Arbitrary Style Transfer: Subjective Study and Objective Metric'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 33.7, pp. 3055–3070. DOI: https://doi.org/10.1109/tcsvt.2022.3231041. URL: https://arxiv.org/pdf/2208.00623.pdf (cit. on p. 68).

Chen, Long Qing et al. (June 2011). 'Traffic sign detection and recognition for intelligent vehicle'. In: *IEEE Intelligent Vehicles Symposium*. DOI: https://doi.org/10.1109/ivs.2011.5940543 (cit. on pp. 14, 23).

Cheng, Gong et al. (2023). 'Towards Large-Scale Small Object Detection: Survey and Benchmarks'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20. DOI: https://doi.org/10.1109/TPAMI.2023.3290594. URL: https://arxiv.org/abs/2207.14096 (cit. on p. 6).

Chollet, François et al. (2015). *Keras*. https://keras.io (cit. on p. 48).

Córdova, Manuel et al. (Jan. 2022). 'Litter Detection with Deep Learning: A Comparative Study'. In: *Sensors* 22.2, p. 548. DOI: https://doi.org/10.3390/s22020548. URL: https://www.mdpi.com/1424-8220/22/2/548/htm (cit. on pp. 9, 10, 29).

Creswell, Antonia et al. (Jan. 2018). 'Generative Adversarial Networks: An Overview'. In: *IEEE Signal Processing Magazine* 35.1, pp. 53–65. DOI: https://doi.org/10.1109/msp.2017.2765202 (cit. on p. 5).

Deng, Jia et al. (June 2009). 'ImageNet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. DOI: https://doi.org/10.1109/cvpr.2009.5206848 (cit. on p. 16).

Deng, Jiangfan et al. (June 2023). 'Improving Crowded Object Detection via Copy-Paste'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.1,

pp. 497–505. DOI: https://doi.org/10.1609/aaai.v37i1.25124. URL: https://ojs.aaai.org/index.php/AAAI/article/view/25124 (cit. on pp. 13, 40).

Dhanachandra, Nameirakpam, Khumanthem Manglem and Yambem Jina Chanu (2015). 'Image segmentation using k -means clustering algorithm and subtractive clustering algorithm'. In: *Procedia Computer Science* 54, pp. 764–771. DOI: https://doi.org/10.1016/j.procs.2015.06.090 (cit. on p. 14).

Durai, D Binny Jeba and T Jaya (June 2023). 'Automatic severity grade classification of diabetic retinopathy using deformable ladder Bi attention U-net and deep adaptive CNN'. In: *Medical Biological Engineering Computing*. DOI: https://doi.org/10.1007/s11517-023-02860-9 (cit. on p. 15).

Dutta, Abhishek and Andrew Zisserman (2019). 'The VIA Annotation Software for Images, Audio and Video'. In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: ACM. ISBN: 978-1-4503-6889-6/19/10. DOI: 10.1145/3343031.3350535. URL: https://doi.org/10.1145/3343031.3350535 (cit. on p. 20).

Dvornik, Nikita, Julien Mairal and Cordelia Schmid (July 2018). *Modeling Visual Context is Key to Augmenting Object Detection Datasets*. DOI: https://doi.org/10.48550/arXiv.1807.07428. URL: https://arxiv.org/abs/1807.07428 (cit. on pp. 13, 22).

Dwibedi, Debidatta, Ishan Misra and Martial Hebert (Aug. 2017). *Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection*. DOI: https://doi.org/10.48550/arXiv.1708.01642. URL: https://arxiv.org/abs/1708.01642 (cit. on pp. 11, 32).

Elzarka, Hazem, Steven Buchberger and Puneeth Sai Meduri (Mar. 2020). *Mitigating Storm Drainage System Impacts from Litter and Debris*. Ed. by University of Cincinnati. Dept. of Civil/Architectural Engineering and Construction Management. URL: https://rosap.ntl.bts.gov/view/dot/55978 (cit. on p. 2).

Environment Times (2022). *Efforts to tackle roadside litter's deadly effect on England's wildlife*. URL: https://www.environmenttimes.co.uk/news/item/1029-efforts-to-tackle-roadside-litter-s-deadly-effect-on-england-s-wildlife (cit. on p. 3).

Everingham, M. et al. (2010). *The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results*. URL: http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html (cit. on p. 14).

Fang, Hao-Shu et al. (Aug. 2019). *InstaBoost: Boosting Instance Segmentation via Probability Map Guided Copy-Pasting*. DOI: https://doi.org/10.48550/arXiv.1908.07801. URL: https://arxiv.org/abs/1908.07801 (cit. on pp. 13, 22).

Fey, Day et al. (2022). *Trash AI*. URL: https://www.trashai.org/ (cit. on p. 8).

Fulton, Michael et al. (Sept. 2018). *Robotic Detection of Marine Litter Using Deep Visual Detection Models*. DOI: https://doi.org/10.48550/arXiv.1804.01079. URL: https://arxiv.org/abs/1804.01079 (cit. on p. 8).

Gatys, Leon A, Alexander S Ecker and Matthias Bethge (2015). *A Neural Algorithm of Artistic Style*. URL: https://arxiv.org/abs/1508.06576 (cit. on pp. vi, 15, 16, 27, 29).

Georgakis, Georgios V et al. (Oct. 2016). 'Multiview RGB-D Dataset for Object Instance Detection'. In: *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*. DOI: https://doi.org/10.1109/3dv.2016.52 (cit. on p. 12).

Ghiasi, Golnaz et al. (June 2021). *Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation*. DOI: https://doi.org/10.48550/arXiv.2012.07177. URL: https://arxiv.org/abs/2012.07177 (cit. on pp. 13, 22).

Giakoumoglou, Nikolaos, Eleftheria Maria Pechlivani and Dimitrios Tzovaras (Oct. 2023). 'Generate-Paste-Blend-Detect: Synthetic dataset for object detection in the agriculture domain'. In: *Smart Agricultural Technology* 5, p. 100258. DOI: https://doi.org/10.1016/j.atech.2023.100258. URL: https://www.sciencedirect.com/science/article/pii/S2772375523000886 (cit. on p. 13).

Girshick, Ross et al. (June 2014). 'Rich feature hierarchies for accurate object detection and semantic segmentation'. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587. DOI: https://doi.org/10.1109/cvpr.2014.81. URL: https://dl.acm.org/citation.cfm?id=2679851 (cit. on p. 14).

GitHub (2023). *GitHub*. URL: https://github.com/ (cit. on p. 46).

Google (2023). *Google Translate*. URL: https://translate.google.co.uk/?sl=auto&tl=en&op=images (cit. on p. 21).

GOV.UK (2022). *Driving down roadside litter levels in national spring clean*. URL: https://www.gov.uk/government/news/driving-down-roadside-litter-levels-in-national-spring-clean (cit. on p. 3).

– (2023). *Report a litter problem*. URL: https://www.gov.uk/report-litter (cit. on p. 3).

GOV.UK & HM Treasury (2023). *Public sector expenditure on national and local roads in the United Kingdom from 2009/10 to 2022/23 (in billion GBP)*. URL: https://www.statista.com/statistics/298667/united-kingdom-uk-public-sector-expenditure-national-roads/#:~:text=Public%20sector%20spending%20on%20roads (cit. on p. 2).

Harris, Charles R. et al. (Sept. 2020). 'Array programming with NumPy'. In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 47).

Hawkins, Gay. (Jan. 2006). 'Ethics of Waste: How We Relate to Rubbish'. In: LANHAM: ROWMAN LITTLEFIELD. URL: https://eds.s.ebscohost.com/eds/detail/detail?vid=0&sid=a3991070-c2b0-49e6-aa3e-cf540c1cb272%40redis&bdata=JnNpdGU9ZWRzLWxpdmUmc2NvcGU9c2l0ZQ%3d%3d#AN=edsgob.11080533&db=edsgob (cit. on p. 3).

He, Kaiming, Xinlei Chen et al. (Dec. 2021). 'Masked Autoencoders Are Scalable Vision Learners'. In: *arXiv:2111.06377 [cs]*. URL: https://arxiv.org/abs/2111.06377 (cit. on p. 26).

He, Kaiming, Georgia Gkioxari et al. (2017). *Mask r-cnn*. URL: https://arxiv.org/abs/1703.06870 (cit. on p. 14).

Higuchi, Keita et al. (Apr. 2023). 'Interactive Generation of Image Variations for Copy-Paste Data Augmentation'. In: *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* 181. DOI: https://doi.org/10.1145/3544549.3585856 (cit. on p. 13).

Hong, Jungseok, Michael Fulton and Junaed Sattar (July 2020). *TrashCan: A Semantically-Segmented Dataset towards Visual Detection of Marine Debris*. DOI: https://doi.org/10.48550/arXiv.2007.08097. URL: https://arxiv.org/abs/2007.08097 (cit. on p. 8).

Hossain, Md. Shamim et al. (June 2023). 'The segmentation of nuclei from histopathology images with synthetic data'. In: *Signal, Image and Video Processing*. DOI: https://doi.org/10.1007/s11760-023-02597-w (cit. on p. 15).

Hunter, J. D. (2007). 'Matplotlib: A 2D graphics environment'. In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on p. 48).

Jing, Yongcheng et al. (Nov. 2020). 'Neural Style Transfer: A Review'. In: *IEEE Transactions on Visualization and Computer Graphics* 26.11, pp. 3365–3385. DOI: https://doi.org/10.1109/TVCG.2019.2921336. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8732370 (cit. on p. 5).

Jocher, Glenn, Ayush Chaurasia and Jing Qiu (2023). *Ultralytics YOLOv8*. Version 8.0.0. URL: https://github.com/ultralytics/ultralytics (cit. on pp. 1, 15, 17, 18, 29, 48, 68).

Jocher, Glenn, Alex Stoken et al. (Oct. 2020). *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. Version v3.1. DOI: 10.5281/zenodo.4154370. URL: https://doi.org/10.5281/zenodo.4154370 (cit. on p. 29).

Kadish, David, Sebastian Risi and Anders Sundnes Løvlie (May 2021). *Improving Object Detection in Art Images Using Only Style Transfer*. DOI: https://doi.org/10.48550/arXiv.2102.06529. URL: https://arxiv.org/abs/2102.06529 (cit. on pp. 15, 28).

Karimi, Kasra and Ardeshir Faghri (May 2022). 'Study of Litter on Delaware Roadways'. In: *Current Urban Studies* 10.2, pp. 249–262. DOI: https://doi.org/10.4236/cus.2022.102015. URL: https://www.scirp.org/journal/paperinformation.aspx?paperid=117718 (cit. on p. 2).

Karras, Tero, Samuli Laine and Timo Aila (2018). *A Style-Based Generator Architecture for Generative Adversarial Networks*. URL: https://arxiv.org/abs/1812.04948 (cit. on p. 28).

Kirillov, Alexander et al. (Apr. 2023). 'Segment anything'. In: *arXiv (Cornell University)*. DOI: https://doi.org/10.48550/arxiv.2304.02643 (cit. on pp. vi, 15, 25, 26, 48).

Kirschner, Jeff, Dick Ayres and Sean Doherty (2012). *About Litterati*. URL: https://www.litterati.org/about (cit. on p. 9).

Kluyver, Thomas et al. (2016). 'Jupyter Notebooks - a publishing format for reproducible computational workflows'. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by Fernando Loizides and Birgit Scmidt. Netherlands: IOS Press, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/ (cit. on p. 46).

Kraft, Marek et al. (Mar. 2021). 'Autonomous, Onboard Vision-Based Trash and Litter Detection in Low Altitude Aerial Images Collected by an Unmanned Aerial Vehicle'. In: *Remote Sensing* 13.5, p. 965. DOI: https://doi.org/10.3390/rs13050965 (cit. on pp. 8, 10).

Lai, Kevin et al. (May 2011). 'A large-scale hierarchical multi-view RGB-D object dataset'. In: *International Conference on Robotics and Automation*. DOI: https://doi.org/10.1109/icra.2011.5980382 (cit. on p. 12).

Li, Da et al. (Oct. 2017). *Deeper, Broader and Artier Domain Generalization*. DOI: https://doi.org/10.48550/arXiv.1710.03077. URL: https://arxiv.org/abs/1710.03077 (cit. on p. 15).

Li, Yiting et al. (May 2023). 'A Modified YOLOv8 Detection Network for UAV Aerial Image Recognition'. In: *Drones* 7.5, p. 304. DOI: https://doi.org/10.3390/drones7050304. URL: https://www.mdpi.com/2504-446X/7/5/304 (cit. on p. 30).

Lin, Tsung-Yi, Piotr Dollár et al. (Apr. 2017). 'Feature Pyramid Networks for Object Detection'. In: *arXiv:1612.03144 [cs]*. URL: https://arxiv.org/abs/1612.03144 (cit. on p. 30).

Lin, Tsung-Yi, Michael Maire et al. (2014). 'Microsoft COCO: Common Objects in Context'. In: *Computer Vision – ECCV 2014* 8693, pp. 740–755. DOI: https://doi.org/10.1007/978-3-319-10602-1_48. URL: https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48 (cit. on pp. 9, 16, 30, 38, 48).

Lou, Haitong et al. (May 2023). 'DC-YOLOv8: Small-Size Object Detection Algorithm Based on Camera Sensor'. In: *Electronics* 12.10, pp. 2323–2323. DOI: https://doi.org/10.3390/electronics12102323 (cit. on p. 30).

Lynch, Seán (June 2018). 'OpenLitterMap.com – Open Data on Plastic Pollution with Blockchain Rewards (Littercoin)'. In: *Open Geospatial Data, Software and Standards* 3.1. DOI: https://doi.org/10.1186/s40965-018-0050-y (cit. on p. 9).

Mahony, Niall O' et al. (2020). 'Deep Learning vs. Traditional Computer Vision'. In: *arXiv:1910.13796 [cs]* 943. DOI: https://doi.org/10.1007/978-3-030-17795-9. URL: https://arxiv.org/abs/1910.13796 (cit. on p. 3).

Majchrowska, Sylwia et al. (Feb. 2022). 'Deep learning-based waste detection in natural and urban environments'. In: *Waste Management* 138, pp. 274–284. DOI: https://doi.org/10.1016/j.wasman.2021.12.001 (cit. on pp. 4, 9, 10).

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/ (cit. on pp. 48, 49).

Merkel, Dirk (Mar. 2014). 'Docker: Lightweight Linux Containers for Consistent Development and Deployment'. In: *Linux J.* 2014.239. ISSN: 1075-3583 (cit. on p. 50).

Microsoft (2023). *Visual Studio Code*. URL: https://code.visualstudio.com/ (cit. on p. 46).

Mortensen, Eric M and William A Barrett (Sept. 1995). 'Intelligent scissors for image composition'. In: *International Conference on Computer Graphics and Interactive Techniques*. DOI: https://doi.org/10.1145/218380.218442 (cit. on p. 21).

National Highways (2023). *National Highways - Summary reports*. URL: https://report.nationalhighways.co.uk/reports/National+Highways?zoom=8&lat=52.75855&lon=-1.1687 (cit. on p. 3).

Nowruzi, Farzan Erlik et al. (July 2019). *How much real data do we actually need: Analyzing object detection performance using synthetic and real data*. DOI: https://doi.org/10.48550/arXiv.1907.07061. URL: https://arxiv.org/abs/1907.07061 (cit. on p. 11).

NVIDIA, Péter Vingelmann and Frank H.P. Fitzek (2022). *CUDA, release: 12*. URL: https://developer.nvidia.com/cuda-toolkit (cit. on p. 49).

Paszke, Adam et al. (2019). 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *CoRR* abs/1912.01703. arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703 (cit. on p. 48).

Paulin, Goran and Marina Ivašić-Kos (Jan. 2023). 'Review and analysis of synthetic dataset generation methods and techniques for application in computer vision'.

In: *Artificial Intelligence Review*. DOI: https://doi.org/10.1007/s10462-022-10358-3 (cit. on pp. vi, 11, 12).

Pedregosa, F. et al. (2011). 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on p. 48).

Pérez, Patrick, Michel Gangnet and Andrew Blake (July 2003). 'Poisson image editing'. In: *ACM SIGGRAPH 2003 Papers*. DOI: https://doi.org/10.1145/1201775.882269 (cit. on pp. vi, 32, 33).

Perrett, Andrew et al. (June 2023). 'DeepVerge: Classification of roadside verge biodiversity and conservation potential'. In: *Computers, Environment and Urban Systems* 102, p. 101968. DOI: https://doi.org/10.1016/j.compenvurbsys.2023.101968. URL: https://www.sciencedirect.com/science/article/pii/S0198971523000315 (cit. on p. 2).

Petersen, Richard (2018). *Ubuntu 18.04 Lts Desktop: Applications and Administration*. surfing turtle press. ISBN: 1936280523 (cit. on p. 49).

Philippe, M. et al. (Feb. 2023). 'Fate of antimony contamination generated by road traffic – A focus on Sb geochemistry and speciation in stormwater ponds'. In: *Chemosphere* 313, p. 137368. DOI: https://doi.org/10.1016/j.chemosphere.2022.137368. URL: https://www.sciencedirect.com/science/article/pii/S0045653522038619#bib31 (cit. on p. 2).

Phillips, Benjamin B. et al. (Oct. 2021). 'Road verge extent and habitat composition across Great Britain'. In: *Landscape and Urban Planning* 214, p. 104159. DOI: https://doi.org/10.1016/j.landurbplan.2021.104159. URL: https://www.sciencedirect.com/science/article/pii/S0169204621001225 (cit. on p. 2).

Proença, Pedro F. and Pedro Simões (Mar. 2020). *TACO: Trash Annotations in Context for Litter Detection*. DOI: https://doi.org/10.48550/arXiv.2003.06975. URL: https://arxiv.org/abs/2003.06975 (cit. on pp. 1, 9, 16, 20).

Ramalingam, Balakrishnan et al. (Apr. 2021). 'Deep Learning Based Pavement Inspection Using Self-Reconfigurable Robot'. In: *Sensors* 21.8, p. 2595. DOI: https://doi.org/10.3390/s21082595 (cit. on p. 10).

RangeKing (2023). *Brief summary of YOLOv8 model structure*. URL: https://github.com/ultralytics/ultralytics/issues/189 (cit. on pp. vi, 30).

Redmon, Joseph et al. (2015). *You Only Look Once: Unified, Real-Time Object Detection*. URL: https://arxiv.org/abs/1506.02640 (cit. on pp. vi, 9, 16, 29, 30).

Ren, Shaoqing et al. (June 2017). 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6, pp. 1137–1149. DOI: https://doi.org/10.1109/tpami.2016.2577031. URL: https://ieeexplore.ieee.org/document/7485869 (cit. on pp. 10, 12, 16, 29).

Ronneberger, Olaf, Philipp Fischer and Thomas Brox (May 2015). *U-net: Convolutional networks for biomedical image segmentation*. URL: https://arxiv.org/abs/1505.04597 (cit. on p. 15).

Rozantsev, Artem, Vincent Lepetit and Pascal Fua (Aug. 2015). 'On rendering synthetic images for training an object detector'. In: *Computer Vision and Image Understanding* 137, pp. 24–37. DOI: https://doi.org/10.1016/j.cviu.2014.12.006 (cit. on p. 11).

Rummelhard, Lukas et al. (June 2017). 'Ground estimation and point cloud segmentation using SpatioTemporal Conditional Random Field'. In: *IEEE Xplore*, pp. 1105–1110. DOI: https://doi.org/10.1109/IVS.2017.7995861. URL: https://ieeexplore.ieee.org/abstract/document/7995861 (cit. on p. 14).

Sekachev, Boris et al. (Aug. 2020). *opencv/cvat: v1.1.0*. Version v1.1.0. DOI: 10.5281/zenodo.4009388. URL: https://doi.org/10.5281/zenodo.4009388 (cit. on pp. 5, 20, 21).

Serezhkin, A (2020). *Drinking Waste Classification*. URL: https://www.kaggle.com/arkadiyhacks/drinking-waste-classification (cit. on p. 8).

Shorten, Connor and Taghi M. Khoshgoftaar (July 2019). 'A survey on Image Data Augmentation for Deep Learning'. In: *Journal of Big Data* 6.1. DOI: https://doi.org/10.1186/s40537-019-0197-0 (cit. on pp. 4, 5).

Simonyan, Karen and Andrew Zisserman (Apr. 2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. URL: https://arxiv.org/abs/1409.1556 (cit. on p. 28).

Singh, Arjun et al. (Sept. 2014). 'BigBIRD: A large-scale 3D database of object instances'. In: *International Conference on Robotics and Automation*. DOI: https://doi.org/10.1109/icra.2014.6906903 (cit. on p. 12).

Supervisely, Anna et al. (May 2022). *Country Roads*. URL: https://github.com/supervisely-ecosystem/country-roads#download (cit. on p. 21).

Talaat, Fatma M and Hanaa ZainEldin (July 2023). 'An improved fire detection approach based on YOLO-v8 for smart cities'. In: *Neural Computing and Applications*. DOI: https://doi.org/10.1007/s00521-023-08809-1 (cit. on p. 30).

Tan, Mingxing, Ruoming Pang and Quoc V. Le (July 2020). 'EfficientDet: Scalable and Efficient Object Detection'. In: *arXiv:1911.09070 [cs, eess]*. URL: https://arxiv.org/abs/1911.09070 (cit. on p. 9).

The pandas development team (Feb. 2020). *pandas-dev/pandas: Pandas*. Version latest. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134 (cit. on p. 47).

Thung, Gary and M Yang (2016). *Trashnet*. URL: https://github.com/garythung/trashnet (cit. on p. 8).

Torralba, Antonio and Alexei A. Efros (June 2011). *Unbiased look at dataset bias*. DOI: https://doi.org/10.1109/CVPR.2011.5995347. URL: https://ieeexplore.ieee.org/document/5995347 (cit. on p. 15).

University of Lincoln (2023). *LoVE: Laboratory of Vision Engineering*. URL: https://www.visioneng.org.uk/ (cit. on pp. 4, 20, 39, 49).

Van Rossum, Guido and Fred L Drake Jr (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam (cit. on p. 46).

Veeravasarapu, V. S. R. et al. (Dec. 2015). *Model Validation for Vision Systems via Graphics Simulation*. DOI: https://doi.org/10.48550/arXiv.1512.01401. URL: https://arxiv.org/abs/1512.01401 (cit. on p. 11).

Venkateswara, Hemanth et al. (June 2017). *Deep Hashing Network for Unsupervised Domain Adaptation*. DOI: https://doi.org/10.48550/arXiv.1706.07522. URL: https://arxiv.org/abs/1706.07522 (cit. on p. 15).

Wang, Tao et al. (July 2020). 'A Multi-Level Approach to Waste Object Segmentation'. In: *Sensors* 20.14, p. 3816. DOI: https://doi.org/10.3390/s20143816 (cit. on p. 8).

Wang, Xu et al. (2023). 'CGA-UNet: Category-guide attention u-net for dental abnormality detection and segmentation from dental-maxillofacial images'. In: *IEEE Transactions on Instrumentation and Measurement*, pp. 1–1. DOI: https://doi.org/10.1109/TIM.2023.3288256. URL: https://ieeexplore.ieee.org/abstract/document/10158743 (cit. on p. 15).

Wang, Yue et al. (Aug. 2022). 'Feature-Based Style Randomization for Domain Generalization'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.8, pp. 5495–5509. DOI: https://doi.org/10.1109/TCSVT.2022.3152615. URL: https://ieeexplore.ieee.org/document/9716108 (cit. on pp. 15, 28).

Wang, Zhizhong et al. (June 2021). 'Evaluate and improve the quality of neural style transfer'. In: *Computer Vision and Image Understanding* 207, p. 103203. DOI: https://doi.org/10.1016/j.cviu.2021.103203 (cit. on p. 67).

Xu, Zhenbo et al. (2021). *Continuous Copy-Paste for One-Stage Multi-Object Tracking and Segmentation*. URL: https://openaccess.thecvf.com/content/ICCV2021/html/Xu_Continuous_Copy-Paste_for_One-Stage_Multi-Object_Tracking_and_Segmentation_ICCV_2021_paper.html (cit. on p. 13).

Zhang, Jinyou and Nagel H D (Oct. 1994). 'Texture-based segmentation of road images'. In: *Proceedings of the Intelligent Vehicles '94 Symposium*. DOI: https://doi.org/10.1109/ivs.1994.639516 (cit. on p. 14).

Zhou, Mingzhe (Dec. 2022). *Research Advanced in Deep Learning Object Detection*. DOI: https://doi.org/10.1109/TOCS56154.2022.10016018. URL: https://ieeexplore.ieee.org/document/10016018 (cit. on p. 3).

Zhou, Wei et al. (Apr. 2023). 'Automatic waste detection with few annotated samples: Improving waste management efficiency'. In: *Engineering Applications of Artificial Intelligence* 120, p. 105865. DOI: https://doi.org/10.1016/j.engappai.2023.105865 (cit. on pp. 9, 10).

Zhu, Yukun et al. (2015). *segDeepM: Exploiting Segmentation and Context in Deep Neural Networks for Object Detection*. URL: https://openaccess.thecvf.com/content_cvpr_2015/html/Zhu_segDeepM_Exploiting_Segmentation_2015_CVPR_paper.html (cit. on p. 14).