SZAKDOLGOZAT

Nyilas Péter
Mérnökinformatikus hallgató részére

Közlekedési objektumok detekcióját megvalósító neurális hálós modell tanítása és magyarázata

A közlekedési objektumok, mint például autók, gyalogosok, kamionok, motorkerékpárok, biciklik és egyéb dinamikus közlekedési szereplők felismerése kritikus fontosságú feladat az önvezető, illetve a vezetést támogató rendszerek, valamint közlekedési monitoring rendszerek fejlesztése során. A neurális hálókon alapuló objektumdetekciós módszerek lehetővé teszik, hogy ezek az objektumok valós időben és nagy pontossággal detektálhatók legyenek.

A szakdolgozat célja egy olyan mély neurális hálózat kialakítása és tanítása, amely képes a közlekedési objektumok automatikus felismerésére és azonosítására képek alapján. Emellett az alkalmazás biztonságkritikus jellege miatt a hallgatónak meg kell vizsgálnia a modell magyarázhatósági aspektusait is, különös tekintettel a modell interpretálhatóságára.

A hallgató feladatának a következőkre kell kiterjednie:

- Végezzen irodalomkutatást a konvolúciós mély neurális hálók és a magyarázó technikák témájában!
- Végezzen adatgyűjtést, és készítse elő az adatokat a háló tanításához és validálásához!
- Végezze el a választott objektumdetekcióra használható neurális háló tanítását és értékelje ki annak teljesítményét!
- Alkalmazza a megismert modelfüggő és modelfüggetlen magyarázó módszereket a modell döntéseinek elemzésére!
- Alkalmazzon legalább két magyarázó módszert, és hasonlítsa össze azok hatékonyságát és eredményeit!

Tanszéki konzulens: Dr. Hullám Gábor, docens

Budapest, 2024.09.26.

……..
Dr. Dabóczi Tamás
tanszékvezető, egyetemi tanár

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Mesterséges Intelligencia és Rendszertervezés Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Artificial Intelligence and Systems Engineering

Training and Interpretation of a Neural Network Model for Traffic Object Detection

Bachelor's Thesis

Author
Péter Nyilas

Advisor
dr. Gábor Hullám

October 26, 2024

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Nyilas Péter*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2024. október 26.

Nyilas Péter
hallgató

Kivonat

Ez a dokumentum egy mesterséges intelligencia alapú rendszer fejlesztését mutatja be, amely képes automatikusan felismerni és osztályozni különböző objektumokat képeken, illetve foglalkozik a neurális hálózat interpretálásával. A rendszer a YOLOv8 modelt használja, amely egy mély neurális hálózat alapú objektumfelismerő algoritmus. A dolgozat részletesen tárgyalja a modell betanításának és interpretálásának folyamatát, valamint az elért eredményeket és levonja a következtetéseket.

Abstract

This paper describes the development of an artificial intelligence-based system that can recognise and classify different objects in images and interprets the neural network. The system uses the YOLOv8 model, which is a deep neural network based object recognition algorithm. The paper discusses in detail the process of training and interpreting the model and the results obtained and draws conclusions.

1 Introduction

The accurate detection of objects that are relevant to vehicle control is a critical task in the field of automotive and transportation systems. It is a particularly important aspect for the development of highly sophisticated autonomous driving systems, where numerous scenarios can arise in with different types of objects to detect and react to. These objects can be either static (not moving), or dynamic (capable of moving). The object detection is implemented trough the help of sensors using different physical phenomenons, such as acoustic or electromagnetic wave. These sensors are mounted onto the chassis of the autonomous vehicle.

One of the most widely used solution is the use of camera sensors as they are able to provide rich visual data that can be processed rapidly and can give the most information by itself. The main goal of this project was to develop a software capable of detecting dynamic traffic objects, while staying robust and reliable in various conditions. A fitting solution for this task was to utilise deep learning models, especially Convolutional Neural Networks, CNNs for short.

Nevertheless, the most significant limitation of these models is that they are frequently regarded as black boxes, lacking a straightforward correlation between input and output. As we might want to use these systems in a safety critical applications, it is important to avoid unpredictable behaviour or mistrust from the stakeholders. High level of explainability should be the bases for trust and confidence between the system and its stakeholders. In accordance with this, the artificial intelligence-functional safety and AI systems ISO standard[6] devotes a chapter(8.3) to Degree of transparency and explainability. These needs led to the development of numerous model interpretation techniques, as evidenced by the work of a number of different researchers such as Yu Liang [9]. These techniques vary in their approach, but they are all aiming to facilitate an insight into the decision-making process of the model. In a unified manner, these methods are called eXplainable Artificial Intelligence methods, XAI for short.

The complexity of interpreting models in these domains arises from two key factors: the intricate nature of the data and the sheer size of models used. The aforementioned complexity makes it challenging to trace the decision-making process, which unfolds across numerous neurons and layers, presents a significant challenge. Nevertheless, it is essential for the identification of potential biases, the improvement of model performance, the fostering the trust of users and regulatory bodies. These factors are particularly important in the context of traffic object detection, where the consequences of model errors can be severe. This has heightened the importance of XAI solutions, with the aim of ensuring transparency for end users and developers alike.

As previously discussed in relation to other challenges facing the development of AI systems by Arun Das and Paul Rad in their article "Opportunities and Challenges in the Development of AI Systems" [3] The General Data Protection Regulation (GDPR) now requires that decisions made by automated systems be explainable to the user. Although the GDPR does not explicitly mention XAI and mainly focuses on data privacy, it seems probable that in the future, there will be a greater expectation of algorithmic transparency and clarification of AI systems.

It is therefore essential to enhance the interpretability of this project's model through the use of the aforementioned XAI methods such as SHAP, LIME and EigenCAM in order to maintain safety and ethical standards, given the nature of the application. Based on that the secondary objective of this project was to try and implement these interpretation methods and examine their effectiveness in the context of traffic object detection and try to instate a more transparent and reliable model.

1.1 Refinement of the task

The principal objective of this project is to develop a software solution capable of detecting traffic objects in real time. This will entail an investigation into the fundamental principles of deep learning models for object detection, with a particular emphasis on the construction of a model based on the YOLOv8 architectural framework. The model will be trained on a dataset comprising images of urban environments, each of which has been annotated with traffic objects. Following training, the model's performance will be evaluated based on its capacity to accurately detect traffic objects in real scenarios.

In addition to object detection, the secondary objective of the project is to enhance the interpretability of the model through the utilisation of Explainable AI (XAI) methodologies. The selection and implementation of both model-agnostic and model-specific interpretation techniques will be undertaken (they will be discussed and explained later), with a particular focus on understanding their underlying principles and applying them to the traffic detection model. The effectiveness of these interpretation techniques will be evaluated based on their ability to provide insights into the rationale behind the model's predictions. This will ensure that the system is both accurate and transparent in its decision-making processes. The objective of this project is to develop a deep learning model for traffic object detection and to apply XAI methods to enhance the model's interpretability.

The two tasks were approached as discrete entities, and thus their explanations were presented separately. Initially, the object detection and the neural network's

fulfilment of the task and its evaluation were discussed. Subsequently, the model's interpretation methods and their outcomes were evaluated.

The choice of dataset, model and interpretation methods was made with these goals in mind, with the intention of optimizing the model for the task in question, in line with the findings of the literature and previous studies.

1.2 Structure

The main parts of my thesis are as follows:

- Object detection component:
  - Literature study about image processing Deep Convolutional Neural Networks.
  - Yolo architecture and general overview on training and infrastructure.
  - Cityscapes dataset and data processing and representation.
  - Evaluation of model performance.
- Interpretation methods
  - Literature review about model-agnostic and model-dependent interpretation methods.
  - Implementation, use and evaluation of EigenCAM, LIME and SHAP.
  - To analyze the results and draw conclusions on the effectiveness of the model and the XAI solutions for traffic object detection.

2 Image Processing Deep Neural Networks

2.1 Overview of Convolutional Neural Networks

CNNs constitute a class of deep learning models that are commonly used for computer vision tasks including image classification, object detection, and semantic or instance segmentation. Given the ability of convolutional neural networks to learn spatial hierarchies of features and their various types of data representations (bounding box, segmentation mask, etc.), they are made ideal for the task of traffic object detection. Many recent advances in computer vision have been made possible by CNNs, including the development of models such as Yolo, Faster R-CNN, and SSD. A dozen of these models have been developed, each with its own unique architecture

and capabilities. However, they all share the same fundamental principles gathered recently by Jiuxiang Gu in his 2018 work, "Recent advances in convolutional neural networks" [5].

The designation of the class is derived from the utilisation of convolutional layers, which apply filters to the input data in order to extract features. Additionally, convolutional neural networks comprise other layers, including pooling layers, fully connected layers, and so forth.

These models are employed in a multitude of applications, including the perception component in autonomous vehicles, surveillance systems, and medical imaging. In view of the fact that these applications are in alignment with the project's stated objectives, and given that the dataset in question is derived from a number of different urban traffic environments, it has been decided that these types of neural networks should be employed for the specific task of traffic object detection

The subsequent paragraphs will provide an overview of the essential components of convolutional neural networks.

Convolutional layers represent the fundamental building blocks of convolutional neural networks. Convolution operations are applied to the input data using filters (or in other word kernels) that process the input image. This process enables the model to identify local patterns such as edges, textures, and shapes. Each convolutional layer generates a set of feature maps, which indicate the presence of different types of these aforementioned features in the input. These feature maps are then passed to the subsequent layers for further processing, such operations as pooling or classification, to refine the information coded into the image. During the training phase, the parameters of the kernels are learned (their values are adjusted), through backpropagation, allowing the network to enhance feature extraction for specific tasks.

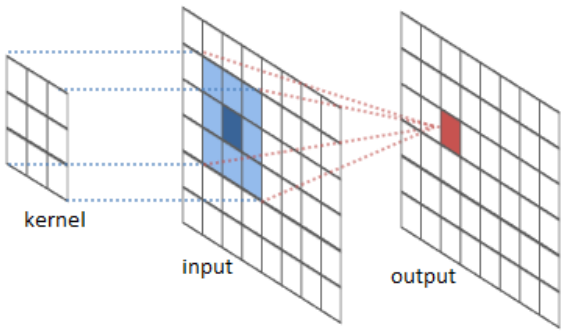


Figure 1: Convolutional layer operation [16]

Fully connected layers is used to describe a type of neural network in which each neuron is connected to every other neuron in the network's preceding layer, resulting in a dense connectivity pattern. This connectivity pattern is also referred to as a "dense layer" due to its dense connections on the semantic figure. They are typically located at the conclusion (HEAD) of convolutional neural network (CNN) architectures, where the final output is calculated. In these layers, as one neuron receives data from every neuron in the previous layer, each connection characterised by a specific trainable weight and each neuron possesses a unique trainable bias. This structure enables the model to integrate information from all features and make final predictions. Fully connected layers in image processing or object detection tasks are computing the output probabilities for each class based on the features extracted by the preceding layers. Regularization techniques, such as dropout and batch normalisation, are frequently employed in these layers to prevent overfitting.

Batch normalisation is a process that normalises the inputs of each layer in a neural network, ensuring that their mean is zero and their standard deviation is one. During training this normalisation is run on every mini-batch (small, randomly selected subset of training samples) of the data.

Dropout is a technique which involves the selecting of a random subset of neurons from the model's chosen layer, then ignoring them during training (dropping them out). This results in their exclusion from the whole learning process. The advantage of this process is that it helps the model to be resilient and less prone to overfitting, by forcing it to learn multiple independent representation of the data.

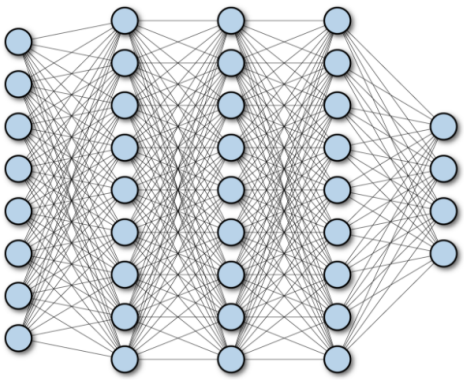


Figure 2: Fully Connected Layer semantics [16]

2.1.1 Sampling layers

Sampling layers are utilised to reduce the spatial dimensions of input data while maintaining its essential characteristics. This process encompasses techniques such

as sub-sampling, strided convolutions, and global pooling. The following section provides an in-depth examination of several frequently employed sampling methods.

In the following paragraphs I will discuss some of the most important sampling layers.

Max Pooling is one of the most prevalent sub-sampling techniques employed in convolutional neural networks. In this approach, a sliding window processes the input feature map, and for each window, the maximum value is selected. This operation reduces the dimensionality of the feature map, preserving the most notable features (such as edges or textures) while discarding those of lesser relevance. Max pooling is particularly effective in object detection tasks where maintaining the strongest activations is of paramount importance, as evidenced by the findings of GU. [5].

Average Pooling , similarly to maximum pooling, average pooling utilises a sliding window; however, rather than calculating the maximum value, it computes the average. This results in a more gradual representation of the feature map and is frequently employed when the objective is to generalise features to a greater extent than preserving sharp edges [5].

Global Pooling methods, such as global max pooling and global average pooling, condense the entire feature map into a single value for each channel, based on the method chosen, eg. if the global max pooling is chosen, entire channel is condensed into the maximal value of the channel. This technique is commonly employed at the conclusion of a convolutional neural network, particularly in image classification tasks, where the spatial location of features is of lesser consequence than their mere presence. Global pooling mitigates the risk of overfitting by producing a compact representation of the input data, as evidenced by Ruby (2020). [5].

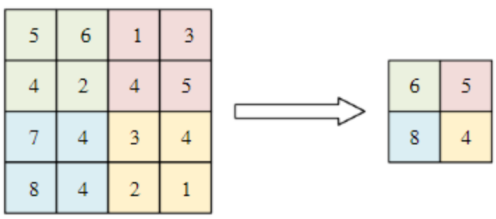
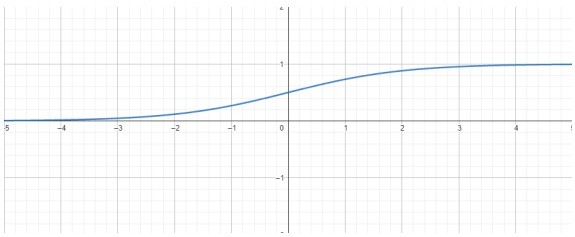


Figure 3: Pooling layer operation [16]

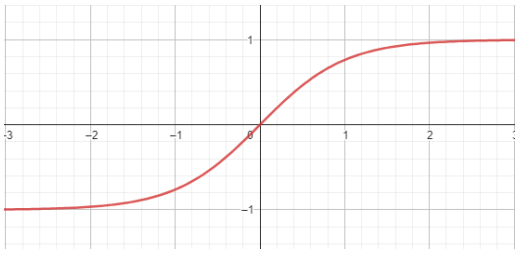
2.2 Activation Functions

Activation functions are employed in convolutional neural networks and other neural networks to introduce non-linearity into the model, thereby enabling it to learn complex patterns and relationships within the data by determining the output of each neuron based on its input. After calculating the weighted sum of the inputs to a neuron, the activation function transforms this sum into the output. These functions allows the model to represent a wide range of functions. The output of the activation function serves as the input to the next layer.

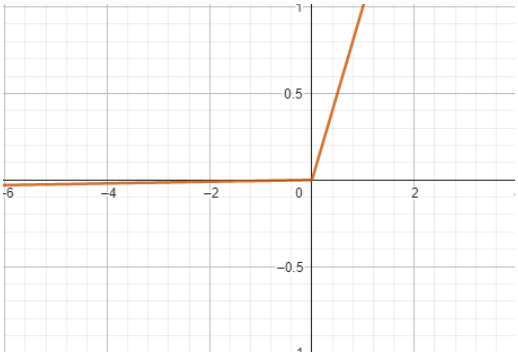
In light of the work conducted by Siddharth Sharma and Simone Sharma regarding activation functions [15] the most prevalent activation functions are ReLU, Sigmoid, and Tanh (Hyperbolic tangent). Although alternative activation functions, such as BSF and ELU, could be employed, they are not utilised by the examined model (Yolov8). Consequently, a detailed discussion of these functions will not be provided. The following paragraphs will discuss the ReLU, Sigmoid, and Tanh activation functions based on the work of Siddharth and Simone Sharma [15].



(a) Sigmoid activation function



(b) Tanh activation function



(c) ReLU activation function

Figure 4: Activation functions: Sigmoid,Tanh and ReLU

ReLU: the Rectified Linear Unit (ReLU) is one of the most common and easy to understand activation functions in convolutional (or any other) neural networks. It is defined as $f(x) = \max(0, x)$ in the works of Siddharth Sharma [15]. This function introduces non-linearity while maintaining a simple and efficient computation. The ReLU function helps mitigate the vanishing gradient issue, allowing models to learn

faster and perform better. However, be susceptible to the phenomenon known as the "dying ReLU" problem, where neurons become inactive and only output zeros, particularly during the training of deep networks.

Sigmoid: the Sigmoid function translates input values to a range between 0 and 1, rendering it useful for binary classification problems. It is defined as $f(x) = \frac{1}{1+e^{-x}}$ in the works of Siddharth Sharma [15]. While the Sigmoid function provides smooth gradients, it is prone to the vanishing gradient problem, especially for large positive or negative input values. This can slow down the training of deep networks, which is why it is often replaced by other activation functions in hidden layers, though it still finds use in the output layer for binary classification tasks.

Tanh: the hyperbolic tangent (Tanh) function is another activation function that maps input values to a range between -1 and 1. It is defined as $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$in the works of Siddharth Sharma [15]. The Tanh function is zero-centered and centricaly symmetrical, facilitates the centring of the data and may result in accelerated convergence during training. Nevertheless, it continues to exhibit the vanishing gradient problem for large input values, though to a lesser extent than the Sigmoid function.

2.3 Introduction to YOLO

The YOLO (You Only Look Once) model is a relatively recent object detection system that is straightforward to utilise and comprehend, while also benefiting from a thriving community of users and developers. It is based on the YOLO algorithm, which is a real-time object detection algorithm developed by Joseph Redmon and Ali Farhadi in 2015[12].

Unlike traditional object detection methods that apply classifiers to various sections of an image, YOLO (You Only Look Once) approaches the problem as a single regression problem. The algorithm divides the input image into an $S \times S$ grid and predicts bounding boxes and class probabilities in parallel for each grid cell, enabling the model to detect multiple types and instances of objects in a single run. This kind of architecture not only enhances speed by processing the entire image at once but also improves detection accuracy by reducing the number of false positives. By minimizing redundant detections and effectively capturing the spatial context of objects, YOLO allows for real-time applications, given its fast inference speeds, making it particularly effective in scenarios such as video surveillance, autonomous driving, and robotics.

The YOLO model has evolved through multiple versions, with improvements in both performance and varying capability. Subsequent versions of the model, including Yolov2, Yolov3, and the latest Yolov5 and Yolov7, as well as Yolov8 (with Yolov10 and 11 forthcoming), have introduced advancements in network architecture, feature extraction, and training techniques. These developments have rendered YOLO a suitable candidate for a noumber of applications, including autonomous vehicles, as evidenced by my own observations during my work in the industry, surveillance systems, and real-time video analysis.

A further noteworthy attribute of the Yolo-type neural networks is their scalability and versatility in terms of model architecture. These models are available in a range of sizes (*nano, small, medium, large, extralarge*) and with a variety of detection types (*semantic segmentation, bounding boxes, oriented bounding boxes, instance segmentation*), which can be deployed in diverse scenarios.

Model	size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.2	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 5: Different sizes of the Yolov8 [17]

2.4 Model architecture

This network, like many others in the CNN family, has a distinctive architectural configuration that enables the execution of intricate tasks such as object detection and classification. The system is constituted of three distinct and discrete components, each comprising a unique set of layers that perform specific and separate functions. The following paragraphs aim to provide an overview of these elements and their contribution to the model's detection.

Backbone: the Yolov8 model is based on convolutional neural networks that have been specifically designed to capture essential features from input images. It comprises multiple layers of convolutional operations (called Conv and a complex layer called C2F) that extract progressively more sophisticated representations. This

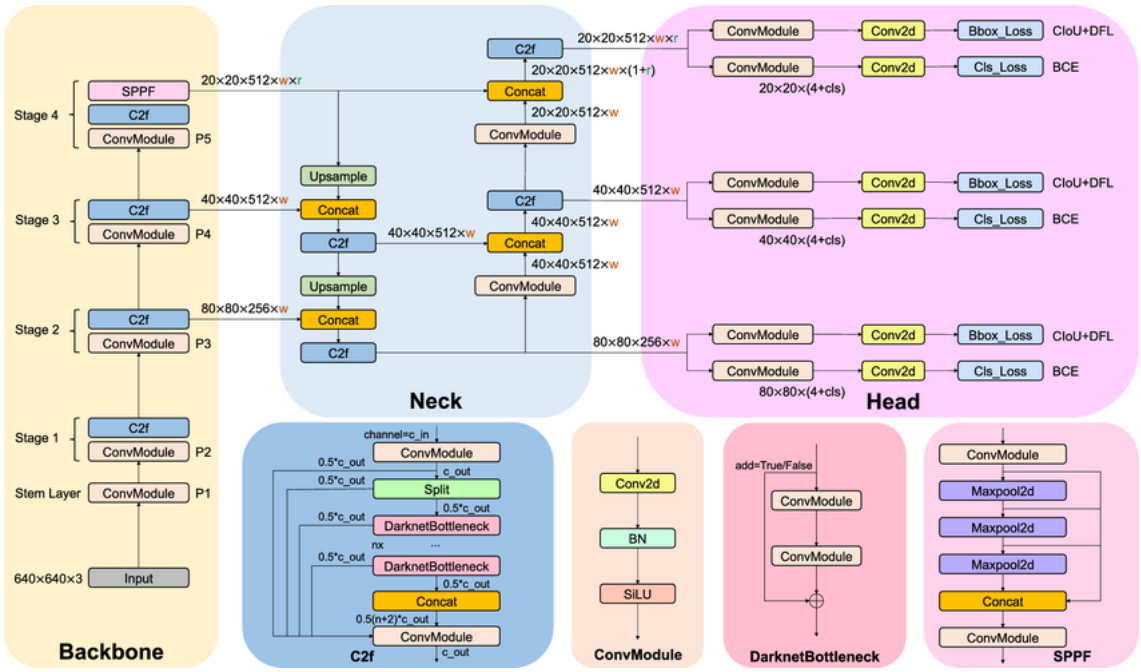


Figure 6: The architecture of the Yolov8 model [7]

structure emphasises both depth and computational efficiency, enabling the model to discern subtle details while maintaining rapid processing capabilities. Innovations such as skip connections and normalization techniques are employed to enhance the learning dynamics and improve the model's robustness to variations in input conditions.

Neck: in the Yolov8 architectural design, the neck serves as an intermediary between the backbone and the head. The primary function of the neck is to consolidate features from the various levels of the backbone, thereby enhancing the model's ability to detect objects across different scales. By employing strategies such as feature fusion or pyramid pooling, the neck effectively integrates both coarse and fine features, enabling the model to better handle overlapping objects and diverse scene contexts. This feature aggregation is crucial for optimising the detection performance, as it allows the model to harness a comprehensive range of information.

Head: the head of the Yolov8 model is responsible for generating the final outputs, which are based on the features that have been processed through the neck. The final stage of the Yolov8 model translates the aggregated feature maps into bounding box predictions and associated class scores for each detected object. This section typically employs a combination of convolutional and fully connected layers to refine these outputs, ensuring they are accurate and meaningful. Additionally, the head may implement techniques such as adaptive anchors or confidence scoring

to enhance the localisation and classification accuracy. By effectively synthesising the rich feature information, the head enables Yolov8 to achieve high-performance object detection suitable for a wide array of applications.

2.5 Training

The training process [12] for the Yolov8 model involves feeding its input with a large dataset of labelled images. The model learns to identify objects by minimizing the difference between its predictions and the actual labels trough multiple iterations: After a training cycle, if the default settings are kept, a validation function (so called "val") is run, to determine more information on the model's performance on pictures that are not present in the training set. The training process is iterative, with the model adjusting its parameters to improve its predictions over time. This process is computationally intensive and requires access to powerful hardware, the best fitting hardware for this purpose, that can be found in an ordinary PC, is the GPU.

The training process involves several key steps, including data preprocessing, model initialization, loss calculation, and parameter optimization. The model is trained using a technique called backpropagation, which involves adjusting the model's weights based on the error between its predictions and the ground truth labels.

The Yolov8 model uses a number of different loss functions, according to the original paper [12] , loss functions to measure the difference between its predictions and the actual labels in different ways:

- **CIoU (Complete Intersection over Union) loss**:

$$\text{CIoU} = 1 - \left(\text{IoU} - \frac{\rho^2(\mathbf{b}, \mathbf{b}^g)}{c^2} - \alpha v \right) \text{ [8]} \tag{1}$$

  where ρ is the Euclidean distance between the centre-points of the predicted box $\mathbf{b}$ and the ground truth box $\mathbf{b}^g$, c is the diagonal length of the smallest enclosing box covering the two boxes, α is a positive trade-off parameter that serves to balance the importance of the aspect ratio consistency term v, that measures the consistency of aspect ratios. It is more sensible to localisation accuracy.

- **DFL (Distribution Focal Loss)**:

$$\text{DFL} = -\sum_{i=1}^{N} p_i \log(q_i) \text{ [18]} \tag{2}$$

where p_i is the true probability distribution and q_i is the predicted probability distribution. It is sensible to the accuracy classification.

- **BCE (binary cross-entropy for classification loss)**:

$$\text{BCE} = -\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \text{ [13]} \tag{3}$$

where y_i is the true label and p_i is the predicted probability.

I opted for training the model on my local machine, using a single GPU, while it provided me with sufficient performance, the training time was significantly longer than it would have been on a more powerful cloud machine. To reduce the chance of overfitting, the training dataset is typically divided into training and validation sets, with the latter used to evaluate the model's performance on unseen data. The trainings batch size where set to 3, which is not common, but it was necessary to fit the model on the GPU, to optimise training time.

3 Object detection component implementation

3.1 Dataset and formats

My project started with choosing and downloading a suitable dataset containing data from urban traffic scenarios. I chose the Cityscapes dataset [2]. These data was not in the correct format for the Yolo model to use. The dataset contains semantic segmentations, but the vanilla Yolo architecture is working with bounding boxes. I needed to write a format conversion script, following the descriptions in subsection 4.2.

Cityscapes: the Cityscapes dataset is a large-scale dataset [2] used for training and evaluating object detection models. It contains high-resolution images of urban scenes, with detailed annotations for various objects such as cars, pedestrians, and traffic signs. The dataset is widely used in the field of computer vision for tasks such as semantic segmentation and object detection.

I chose the gtFine dataset, consisting precisely labelled segmentation masks. Based on their work where they published this dataset [2] titled "he cityscapes dataset for semantic urban scene understanding" the gtFine dataset consists of 5000 images, with a resolution of 1024×2048 pixels, and annotations for 30 classes of objects. It is divided into three subsets: training, validation, and test, with 2975, 500, and 1525 images, respectively. The validation and training sets contain annotations for

30 classes, while the test set doesn't include any annotations. The dataset is labelled using pixel-level segmentation masks, which provide detailed information about the location and shape of objects in the scene.

However for the purpose of this work, the annotations were converted into bounding box format, which is more suitable and straight forward for this object detection tasks. Also, the dataset was filtered to include only the classes that are relevant to the project.

3.2 Data management

The images and annotations are converted into a format that the Yolov8 model can process, which is a text file containing the image path and the coordinates of the bounding boxes in pixels for each object in the image. The format is straightforward: each line of the text file represents a single instance of an object. The first variable is the label_id, which denotes the object's class. This is followed by the two-point's four (x, y) type coordinates in pixel. This process uses a custom script that reads the annotations from the Cityscapes dataset and converts the labels into labels whose classes are filtered and transformed the relevant classes into the grouping I chose for this project.

The classes were grouped into five categories:

- **small vehicle** *(usually cars, which are for personal use)*,
- **large vehicle** *(busses, trucks and other large non personal vehicles)*,
- **two wheeler** *(bicycles and motorcycles)*,
- **On-rails** *(trains and trams though the smaller Fine dataset didn't include any in the training set)*
- and **person** *(pedestrian, and rider)*.

The categories were selected on the basis that the model's confusion is more readily managed when objects that are more similar in appearance are grouped together, thereby facilitating the model's ability to distinguish between them.

The script also converts the semantic segmentation masks into bounding boxes, trough finding the most extreme points and of the mask, create a bounding box around them. This converted output is then saved into a text file, which is in the format used as input for the Yolov8 model.

The final structure of the dataset with metadata (descriptors and lists of labels and pictures) is as follows:

- **Root (gtFine)**: The root directory of the Cityscapes dataset, which contains the images and annotations.
  - **image**: Folder containing the images in the dataset, broken down into training, validation, and test sets and further divided into subfolders based on the city where the images were captured.
  - **labels**: The class label for each object in the image, with tha sam subfolder structure as the images.
  - **train.txt**: The training set, which contains the paths to the training images and their corresponding annotations.
  - **val.txt**: The validation set, which contains the paths to the validation images and their corresponding annotations.
  - **test.txt**: The test set, which contains the paths to the test images and their corresponding annotations.
- **Descriptors**: The folder containing a list of class labels and their corresponding indices, as well as path set descriptor txt-s, the train-, val- and test.txts. It's used by the model to determine the classes, their indices and the paths to the images.

3.3 Model Implementation

3.3.1 Justification behind the selection of YoloV8

For my work, I chose the **Yolov8m** configuration, which stands for **Yolo version 8 medium**. My choice was made on the bases of previous experience with this model architecture and the popularity of its applications, made it so there are an abundance of literature regarding this specific algorithm and architecture. It is also suitable for numerous, different applications, this aspect was important because of the high variance in sizes and shapes of the traffic objects needed to be detected by the network. Training with custom datasets has also been made relatively easy.

I also examined other networks such as Faster R-CNN and even an older Yolo version, Yolov5.

I tested Yolov5 on the same dataset and produced inferior performance KPIs (Key Performance Indicators). I confirmed my findings by a paper that was discussing an application of these models [1], Yolov5 exhibits inferior performance relative to Yolov8 on identical input data and with an equivalent architectural configuration. Additionally it is also faster. This comparison renders the Yolov5 model obsolete, and therefore it has been excluded from further consideration.

The Faster R-CNN while being a less sophisticated model it gets outperformed by Yolo on, a comparison study on these networks "Analysis of the performance of Faster R-CNN and Yolov8 in detecting fishing vessels and fishes in real time" [4] it has a table which contains all the necessary information to decide that it is inferior to Yolov8 in almost every aspect, thus it was not chosen over the Yolo architecture.

Given its superior performance compared with the two models examined as well as the extensive availability of related materials, and it's ability to predict bounding boxes and class probabilities for objects in an image, the Yolov8 algorithm has been selected for this undertaking.

3.3.2 Chosen model's description and parametrisation

Based on paragraph 4.3.1 the Yolov8 bounding box detection model's medium version was picked for this project. In order to facilitate the requirements of the project, the model was utilised in a pre-trained state, with the weights being downloaded from the official Yolov8 repository [17], trained on the COCO dataset, as described in the paper where they introduced the network [12]. Subsequently, the model was fine-tuned on the Cityscapes dataset, which was converted into a format compatible with the model's processing capabilities.

Training was conducted using the Adam optimiser with a learning rate of 0.01 for lr1 (initial learning rate) and lrf (final learning rate). I ran the training on the model for 30 epochs with a batch size of 3, to accommodate the limited local GPU memory. The training process ran on a single PNY NVIDIA Quadro P2000 5GB VCQP2000-PB GPU, with a total training time of 28.056 hours.

3.3.3 Training with the help of MLOps solutions

In order to monitor the performance of the model and to facilitate the visualisation of the learning process, as well as to organise the experiments, an online machine learning monitoring solution, Comet.ml, has been selected. This tool is capable of tracking the training process in real time and of visualising the properties of the model on a user-friendly dashboard, which can be accessed from any device with an internet connection. The Comet.ml platform also provides a range of features that can be used to compare different experiments, such as hyperparameter tuning, model versioning, and collaboration tools, while also offering a comprehensive set of APIs for integration with other tools and platforms.

Figure 7 depicts the Comet.ml dashboard, which offers a comprehensive representation of the training process, encompassing loss and accuracy metrics, the learning rate, and the model's performance on the validation set. This information is vital

Figure 7: Comet.ml dashboard made from our model's training.

for evaluating the model's performance and for identifying potential issues that may arise during training. Furthermore, the Comet.ml platform enables the comparison of different experiments, which can be beneficial for fine-tuning the model's hyperparameters and for improving its performance. Which I did try out, for comparing the performances of the same model on the same dataset, but with different epochs and batch sizes to see and find a good balance between training time and performance.

3.3.4 Model evaluation

The evaluation of the Yolov8 model is performed using a set of metrics that measure its performance on the Cityscapes dataset. These metrics include precision, recall, mAP, and IoU, which are commonly used in object detection tasks. I used the mAP metric to evaluate the model's performance on the test set, which provides a comprehensive measure of its accuracy. The mAP is calculated by averaging the precision-recall curves for each class in the dataset, which gives an overall measure of the model's performance.

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{4}$$

where N is the number of classes and AP_i is the average precision for class i.

The IoU metric is used to evaluate the model's ability to accurately predict the bounding boxes for objects in the image. It measures the overlap between the predicted and ground truth bounding boxes, with a higher IoU indicating a more accurate prediction.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{5}$$

In order to ensure the reliability of the subsequent analysis, only those detections with an IoU value of at least 0.80 were considered, and only those predictions with a relatively high degree of accuracy were utilised.

The model's performance is evaluated on the test set's all classes averaged:

Metric	Value
Precision	0.781
Recall	0.611
mAP	0.41
IoU	0.80

Table 1: Model performance metrics

4 Model Interpretation

4.1 Importance of Interpretation

Interpreting machine learning models is essential for understanding their decision-making processes. It helps in identifying biases, improving model performance by providing information helping us the augment data more efficiently, and building trust with users. Interpretation techniques provide insights into how image processing models make predictions and highlight the most influential features.

Interpretation of machine learning algorithms is a fairly new field, but it has gained significant attention in recent years due to the increasing complexity of models and the need for transparency and accountability. This need comes form industrial and judicial actors, who require explanations for the decisions made by models, especially in safety-critical applications such as autonomous vehicles, healthcare, and finance. In the last couple of years, the field of model interpretation has seen significant advancements, each brought us closer to understanding the inner workings of machine learning models, and try to put reason and logic behind otherwise black-box models.

Furthermore, based on these advancements the aforementioned actors are more willing to adopt laws, standards and regulations that require models to be interpretable, if they are to be used in non-research areas.

Building on the aforementioned state of the machine learning world, I felt it was important to try and create a fairly coherent interpretation of the model to meet today's industry standards.

4.2 Introduction to Interpretation methods

Model interpretation is a critical aspect of machine learning that aims to explain the decision-making process of models. Interpretable models are essential for building trust with users, identifying biases, and with this information, developers have a deeper understanding that can help them improve model performance. In this section, there will be a discussion of the importance of model interpretation and a review of different methods of interpretation.

In their survey [9], Liang et al. highlight the significance of model interpretation techniques in providing insights from the model itself, and the output of the model, without knowing the inner state of the model.

Therefore they sorted these techniques into two categories:

1. data driven or more commonly as model-agnostic methods
2. model driven or model-specific methods

The following sections will provide an overview of the two categories. In the event that one is used, an analysis of the algorithms will be presented in light of the implementation used in this project. This will include a comparison and contrast of their respective strengths and weaknesses.

4.3 Model-Agnostic Methods

Based on the work of Liang [9], model-agnostic interpretation methods can be divided into three categories:

1. Perturbation-based interpretation and a Game theoretic approach
2. Adversarial-based interpretation
3. Concept-based interpretation

As discussed in the work of Liang [9], these are designed to explain model predictions without relying on the internal structure of the model, hence their name, making them applicable to a wide range of models. This insight may be more applicable to local examples, as the approach only provides explanations for the given data and not globally for the model itself. For image processing purposes, these methods are particularly easy to understand and implement.

Both the game-theoretic and perturbation-based interpretation methods involve altering the input data and observing the resulting changes in the model's predictions

on the modified image, while Adversarial-based interpretation methods focus on generating small perturbations or adversarial examples that can significantly change the model's predictions, highlighting the model's vulnerabilities.

On the other hand, concept-based interpretation methods aim to link model behavior to high-level human-understandable concepts.

Regardless, I will discuss the different types of data driven interpretation methods in the following paragraphs, each on the basis of Liang's work [9]

4.3.1 Perturbation-based Interpretation

Perturbation-based interpretation methods are model-agnostic techniques that explain model predictions by perturbing the input data. These methods generate perturbed samples by introducing noise (or masking portions of the image) to the input image and observing the alterations in the model's predictions. This masking is the main principle behind these methods.

The most common type of masking is occlusion, where parts of the image are covered to determine their importance for the model's output. It can be interpreted as a form of feature selection, where the model's output is evaluated based on the presence or absence of specific features.

In essence, perturbation-based methods can be conceptualised as an optimisation problem. Given an input vector x, a model function $f(x)$, and a predicted vector y, the objective is to identify which components of x contribute to the generation of the prediction. To accomplish this, the input features are systematically altered by introducing perturbations δx). This approach allows us to infer the importance of specific features by quantifying the extent to which the predicted vector $f(x + \delta x)$. This approach allows us to infer the importance of specific features by quantifying the extent to which the predicted vector y is affected by the introduction of these perturbations.

Perturbation-based methods are focusing on "*local interpretability*", meaning they explain specific predictions rather than providing an overarching understanding of the model's behavior. This local explanation is vastly inferior to global interpretability. Regardless, sometimes it can be extended, to provide broader insights into the models logic.

It works in a similar way to the human visual system and the way we perceive the visual world, so that by obscuring different parts of an object, we can significantly alter our own eye's perception of the object. By analysing the effect of perturbations on the model's output, these methods identify the most important features for a given prediction.

As a prominent implementation of this method is LIME (Local Interpretable Model-agnostic Explanations), which will be discussed in a following section section 6.1, it was selected for use in the project due to its simplicity and effectiveness in interpreting the model's predictions.

There are multiple types of perturbations:

- **Noise insertion**: adding some type of noise (Gaussian or random) to parts of the input data which can reveal the sensitivity of the model to small variations.
- **Blurring/Pixel Modification**: when working with image data, perturbations can be created by blurring parts of an image or modifying pixel intensities, which helps to understand how the clarity or sharpness of particular regions affects the prediction.
- **Feature Deletion for structured data**: remove or zeroing specific features, determining how significant that feature really is to the prediction.
- **Text Data Perturbations**: for text data, this involves removing words, phrases, characters, but it can also mean swapping tokens with synonyms to gauge their importance to the output.

Given the project's domain, I only used perturbations used in image processing tasks, and these exclusively consists of modifying pixels and groups of pixels to determine their usefulness to the detection. In the following, I will discuss two variations of this standard perturbation based approach.

Game theoretic approach with perturbation is a model-agnostic interpretation method that leverage cooperative game theory to explain model predictions. This translates to decomposition of the input image into a number identical, equally-sized components, referred to as "features". These are processed further and a so called Shapley value is calculated to each feature, based on their contribution to the prediction. This will be projected onto the image, where each feature will be coloured according to this value, thus facilitating easier reading and comprehension. A detailed examination of this process can be found in reference 6.1, where the operational mechanics of this methodology will be elucidated.

Based on the work of Lundberg [10] and Liang [9], these method, namely SHAP (SHapley Additive exPlanations), is particularly useful for image processing tasks, as it provides detailed insights into the model's decision-making process. On grounds of the aforementioned, I chose to use SHAP in my project, as it offers a comprehensive and theoretically grounded approach to interpreting the model's predictions.

Adversarial-based interpretation methods, based on the work of Lian [9], represent a subtype of perturbation methods, the objective of which is to provide more robust interpretations. These methods address a common issue in deep neural networks (DNNs), namely poor generalisation performance, as highlighted in the base of my work [9]. Deep neural networks (DNNs) are highly sensitive to perturbations in input data, which can result in significant alterations to the predictions made, thus affecting the stability and reliability of the interpretations produced. The earliest adversarial-based interpretation methods, as exemplified by the approaches put forth by Fong et al. (2017) and other researchers, employed techniques to mitigate the impact of perturbations. This was achieved by utilising random masks and regularising the masks with total-variation norms to smoothen the images that had been perturbed.

Although these methods enhance robustness, they necessitate manual tuning of hyperparameters and yield interpretations of relatively low resolution, thereby constraining their capacity for fine-grained analysis. On these grounds, they are not suitable for applications that require high-resolution interpretations, and such I did not use them in my project.

4.3.2 Concept-based Interpretation

Concept-based interpretation methods, based on the work of Liang [9], employ predefined sets of human-understandable concepts to provide explanations for deep learning models. The generation of these concepts is typically conducted with the assistance of domain experts, thereby rendering this method particularly useful for interpreting the decision-making processes of models in a manner that aligns with human intuition. The process commences with the selection of a concept set (C), which encompasses images or data that correspond to specific attributes of the input. For instance, the concept set may include images of tiger stripes for an image of a tiger.

The definition of these concepts is facilitated by humans, and the concept set is subsequently incorporated into the deep neural network (DNN) in order to ascertain which concepts are most relevant to the model's predictions. Two prominent concept-based interpretation methods are Testing with Concept Activation Vectors (TCAV) and Network Dissection (ND). TCAV explains the behaviour of the model in question by employing human-defined concepts, namely Concept Activation Vectors (CAVs) , which quantify the importance of said concepts to the model's predictions through directional derivatives. This approach allows for the assessment of the sensitivity of the model's predictions to specific concepts, such as texture or colour, across a range of inputs.

In contrast, Network Dissection quantifies the relationship between internal neural network features and visual concepts, utilising metrics such as Intersection over Union (IoU) to ascertain the degree to which individual neurons correspond to concepts such as colour, texture, or objects. This approach facilitates a comprehensive understanding of the specific layers and neurons in a CNN that are responsible for detecting various concepts, thereby offering insights into the interpretability of each layer.

Based on the work of Liang [9], these methods are particularly useful for image processing tasks, as they provide human-understandable explanations for the model's predictions. However, they require the input of domain experts to define the concepts, which can be time-consuming and may introduce bias. Furthermore, the interpretability of these methods is limited by the quality of the concept set, which may not capture all the relevant features of the input data.

As a result, I found that these methods were not suitable for my project, as they require a high level of domain expertise and may not provide the level of interpretability required for fine-grained analysis.

4.3.3 Comparison of chosen Model-Agnostic Methods

In this subsection, I will compare two of the most popular model-agnostic interpretation methods, LIME and SHAP. Based on the work of Liang [9], these model-agnostic interpretation methods have their strengths and weaknesses. Through their example I will illustrate the differences between the two methods.

LIME and SHAP while using different approaches, there are both using some form of perturbation to explain the model's predictions. LIME approximates the model locally by fitting a simple interpretable model around the prediction of interest, providing local explanations by perturbing the input data and observing the changes in the model's predictions. It is generally faster and simpler, making it suitable for quick, local explanations.

On the other hand, SHAP is based on cooperative game theory and uses Shapley values to attribute the contribution of each feature to the prediction. It provides both local and global explanations by considering all possible combinations of features, producing consistent and theoretically sound explanations.

Based on the discussions by [10], Kernel SHAP, a variant of SHAP, is particularly useful for image processing tasks, as it can handle high-dimensional data efficiently. It is based on the idea of approximating the model with LIME and using the Shapley values to explain the model's predictions. This approach combines the strengths

of both LIME and SHAP, providing accurate and efficient explanations for image processing models.

However, SHAP is more computationally intensive due to the need to consider all possible feature combinations. During the interpretation scripts run, SHAP had 4 times larger GPU memory usage than LIME, while the runtime was 100 times longer. While LIME is flexible and can be applied to any model and data type, SHAP offers a more comprehensive and theoretically grounded approach at a higher computational cost.

4.4 Model-Specific Methods

The base idea behind this category is to ground our interpretation on the models internal state. By this in the field of image processing, it aims to somehow visualize some parameters of the inner state of the model, and project it back to the input image. Multiple methods exist in this category, such as Class Activation Maps, Gradient-based methods. The following paragraphs, present a discussion of the Class Activation Maps with reference to the work of Bany Muhammad and his introduction of his interpretation method [11]. Its implementation will be discussed in the next section. Furthermore, I will discuss the Gradient-based methods based on the work of Selvaraju titled "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization" [14].

4.4.1 Class Activation Maps

Class Activation Maps (CAMs) are a model-specific interpretation method designed to highlight the regions or features in an image that are most pertinent to a given model. In an image, these are the regions that are most relevant to a model's decision, particularly in the context of classification tasks. The operation of CAMs is based on the projection of the activations from the convolutional layers back onto the input image. This results in the generation of a heatmap, visually represents the areas that contributed the most to the prediction [11]. This method provides valuable insights into which features the model is focusing on, enabling a better understanding of the model's decision-making process.

The original CAM technique, first introduced by Zhou et al. (2016), is specifically designed for models that include global average pooling layers before the final output layer, often referred to as the "head". These pooling layers facilitate and are responsible for the smooth projections (feature maps) on the methods output image. While the CAM technique is highly effective, it is somewhat limited in that it is optimized for this specific architectural configuration, which restricts its versatil-

ity. It is less flexible for networks that rely on fully connected layers following the convolutional layers [19].

To illustrate, in a classification task where a model identifies a vehicle in an image, CAM would generate a heat map over and around the vehicle. This indicates that the model primarily focused on that object when making its decision, which is a useful insight. In the absence of this information, there is a possibility that the network may be detecting a feature that is not exclusive to that particular object class and this can result in greater confusion within the model.

This type of visual feedback is especially valuable in domains such safety critical (automotive, etc.) or medical fields, where understanding why the model identifies certain patterns is critical [11].

In response to the architectural limitations of the original CAM method, several improvements have been proposed. One significant extension is Grad-CAM (Gradient-weighted Class Activation Mapping), which eliminates the dependence on specific model architectures by using gradients to compute the heatmaps. Details of Grad-CAM will be further discussed in the next section [14].

4.4.2 Gradient-based Methods

Gradient-based methods, such as Grad-CAM (Gradient-weighted Class Activation Mapping), take a different approach by utilizing the gradients flowing through the backpropagation of the network. It can be used to localize the features within the image, that the models learn on. Grad-CAM generates heatmaps that highlight the regions of the image that contributed the most to the model's output. Selvaraju et al. (2019) demonstrated the effectiveness of this technique in their work titled "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization" [14].

By leveraging gradients, Grad-CAM provides more precise localization of features compared to basic CAM techniques, making it particularly useful for deep neural networks in image recognition tasks.

At the end, this method, despite of being more accurate than the vanilla EigenCAM, I did not use it in my project due to two main considerations.

First of all, the method is more computationally expensive, and requires more memory to run.

Secondly, the method is more complex to implement and requires more input from the model's internal state. In order to ensure the model calculates the gradients

correctly, it should be fed with the whole back-propagation chain, which has made the other interpretation methods more time-consuming to implement and run.

Based on that I chose to use the CAM method in my project, as it is simpler to implement and provides valuable insights into the model's decision-making process.

4.4.3 Comparison of Model-Specific Methods

Both CAM and Grad-CAM methods have been widely adopted for interpreting convolutional neural networks and are valuable tools for visualizing and understanding model behaviour. While CAM is limited by its dependence on specific model architectures, Grad-CAM offers a more flexible and generalizable approach that can be applied to a wide range of models. The use of gradients in Grad-CAM allows for more precise localization of features within the image, providing valuable insights into the decision-making process of the model.

In contest of the resulting visualizations, the output of the two networks tends to be really similar to each other. Which can be because the two methods are based on the same principle, and the implementation of the Grad-CAM is a direct extension of the CAM method, utilizing the activation maps with the gradients to provide the explanation.

4.5 Comparison of Interpretation Methods

In this section, I will compare the model-agnostic and model-specific interpretation methods discussed and chosen in the previous subsections and paragraph. Model-agnostic and model-specific interpretation methods each offer unique strengths and weaknesses when applied to an image processing model. Both approaches aim to provide insights into the model's decision-making. However, they are fundamentally different in terms of their underlying assumptions, flexibility, and computational requirements.

Model-agnostic methods often operate by perturbing the input data, observing the changes in the model's predictions, and deriving explanations based on these observations as discussed previously 5.3.1. This makes them highly flexible and applicable to a wide variety of tasks and models. However, this flexibility sometimes comes at the cost of interpretability precision, as model-agnostic methods approximate how a model behaves based on perturbations, which may not capture the complete depth of the model's decision logic.

Conversely, model-specific methods are designed to align with the internal mechanisms of a specific model architecture, thereby facilitating the generation of expla-

nations based on the model's inherent operational logic. These methods are often more precise, as they can directly access the model's components — such as layers, weights, activation vectors, or gradients and utilize this information to interpret how the model made its decision. In fields like image processing, model-specific methods, such as Class Activation Maps (CAMs) or Gradient-based approaches, provide visual insights into the areas of an image that most influenced the model's predictions. By visualizing the model's internal state, model-specific methods offer a more direct way of understanding the model's behavior, especially in tasks where identifying critical features or regions of the data is essential.

In comparing these two categories, the fundamental difference lies in their generalization versus precision. Model-agnostic methods are particularly useful because of their ability to work across different models and domains, making them versatile tools in scenarios where multiple model types and architectures are used. However, they frequently offer a higher-level view of the model's behavior, which may be less precise than the insights gained from model-specific methods. Conversely, model-specific methods provide a more detailed data for understanding of the decision-making process, directly interacting with the internal components. This allows for more detailed insights but limits the applicability to certain model types.

5 Model interpretation component

5.1 Methods for model interpretation

This component was implement into On python script file, which loaded in the the model from a ".pt" PyThorch[1] file and reads in the given image. After that process is complete, a quick resizing of the images is taking place, then the different interpretation methods are run after each other.

EigenCAM: this model-specific interpretation method employs the eigenvalues of the convolutional layers to generate class activation maps. The eigenCAMvalues are derived from the activation vectors of the various layers, which are employed in the calculation of the activation maps. Subsequently, the maps are projected back to the input image, thereby highlighting the regions that are of particular importance for the model's prediction.

The aforementioned regions are useful for understanding the output of each layer and the model's decision-making process. The output of the EigenCAM method can be interpreted as a heatmap, where the intensity of the colour represents the

[1] https://pytorch.org

importance of the region for the model's prediction. By analysing these heatmaps for different layers of the same image, we can gain a deeper insight into the model's decision-making process.

Local Interpretable Model-agnostic Explanations: this model-agnostic, perturbation-based method employs a local surrogate model to approximate the behaviour of the original model, by approximating the model's decision boundary by perturbing an instance and fitting a simpler interpretable model (such as linear regression) to the modified samples The method is frequently employed for the purpose of elucidating individual predictions and elucidating the model's decision-making process. For instance, in image classification, LIME can identify which pixels or regions(segments or superpixels) contribute most to the model's decision, while in other uses, for example in texts, it can highlight important words or phrases.

In my implementation of XAI component, superpixels are generated for LIME, using the Slic algorithm, and a linear model is fitted to the perturbed data. The superpixel perturbation represents a more sophisticated approach than regular grid-based perturbations, as it is more likely to capture the relevant features of the image. This is achieved by creating a non-regular shaped superpixel out of similar and/or approximate pixels. Subsequently, the linear model, which is a white-box mode, is employed to approximate the behaviour of the original model and provide an explanation for the prediction.

The output of the LIME method is a set of coefficients that represent the importance of each feature for the model's prediction. By employing these coefficients and superpixels, the superpixels that are most crucial for the prediction can be identified by the colour of the given superpixels on the methods output image. Useful superpixels are green, while harmful superpixels, which impede the detection of objects, are red.

Shapley Additive explanations: this model-independent, perturbation-based method employs cooperative game theory to assign an "importance" value, known as the Shapley value, to each feature, representing its contribution to the model's prediction. This approach is more intricate but nevertheless satisfactory than the LIME method. The image is partitioned into equal-sized segments, which collectively form a grid, these segments are then organised into a set. This set and all its subsets gets perturbed individually and the model's prediction is observed on them, based on these predictions they get assigned a Shapley value. This value is getting calculated from iteration to iterations for each segment, until all the subsets are evaluated. This calculation is based on the Shapley value formula.

In the implementation, I used Kernel SHAP, which can be described as Linear LIME combined with Shapley values calculated by the Kernel SHAP algorithm.

Theorem 1 (Shapley kernel).

The equations below are from [10].

$$\begin{aligned} \Omega(g) &= 0\text{: baseline value for model g,} \\ \pi_{x'}(z') &= \frac{(M-1)}{(\binom{M}{|z'|})|z'|(M-|z'|)}\text{: weight assigned to each subset of z',} \\ L(f,g,\pi_{x'}) &= \sum_{z'\in Z}\left[f(h_x^{-1}(z')) - g(z')\right]^2 \pi_{x'}(z') \end{aligned} \tag{6}$$

where $|z'|$ is the number of non-zero elements in z'.

The loss function is employed to quantify the discrepancy between the model's prediction and that of the surrogate model.

The loss is calculated by taking the squared difference between the predictions of (f) and (g) for each subset (z').

As this method is model-agnostic, it can be employed to interpret the predictions of any model, regardless of its complexity. his can be a significant limitation when interpreting complex models with a large number of features. Thus, the method was only operational on an online platform, Google Colab[2], through a Jupiter notebook environment, as it provides the necessary computational resources and a fitting platform for the SHAP algorithm to run efficiently.

5.2 Evaluation of interpretation methods

In all of the paragraph, the chosen methods will try to explain on the image on the Figure**??**.

On which the models prediction can be found on the image: Figure**??**.

In the following I will describe all the image interpretations given by the different interpretation methods. And try and Interpret the model's workings based on those outputs.

Evaluation of the EigenCAM method In the figure constellation of Figure8 different layers are presented. These layers are

[2] https://tinyurl.com/shapColab

Figure 8: Picture Bonn35

Name	Place	Description
-2 C2f	Feature Pyramid	A composite layer with Cross-Stage Partial Network (CSP) structure, designed to increase gradient flow and reduce memory usage by splitting the feature maps and merging them at the end.
-3 Concat	Feature Aggregation	Concatenates feature maps from different scales or layers to merge information, enabling multi-scale predictions.
-4 Conv	Convolutional Layer	A standard 2D convolution layer that extracts features from the input by applying filters to generate feature maps.
-5 C2f	Backbone Stage	Similar to the C2f layer at -2, it is responsible for feature extraction with a CSP-like structure to improve computational efficiency and gradient flow.

Table 2: Yolov8 architecture layers and their descriptions.

The images can be read and interpreted by identifying the layer responsible for each process and observing the colours on the provided activation maps. The colour red represents the lowest value of activation, while blue represents the highest. The range of layers observed spanned from the second to last layer to the fifth to last layer, allowing for the finest resolution of the head to be observed. This enabled the creation of a visualisation of the areas that were particularly useful in the detection process.

It is evident that the highest activation values were present in the regions where traffic-related objects were identified. Pedestrians were observed to have their outlines and legs presented with higher activation values. Additionally, it was noted

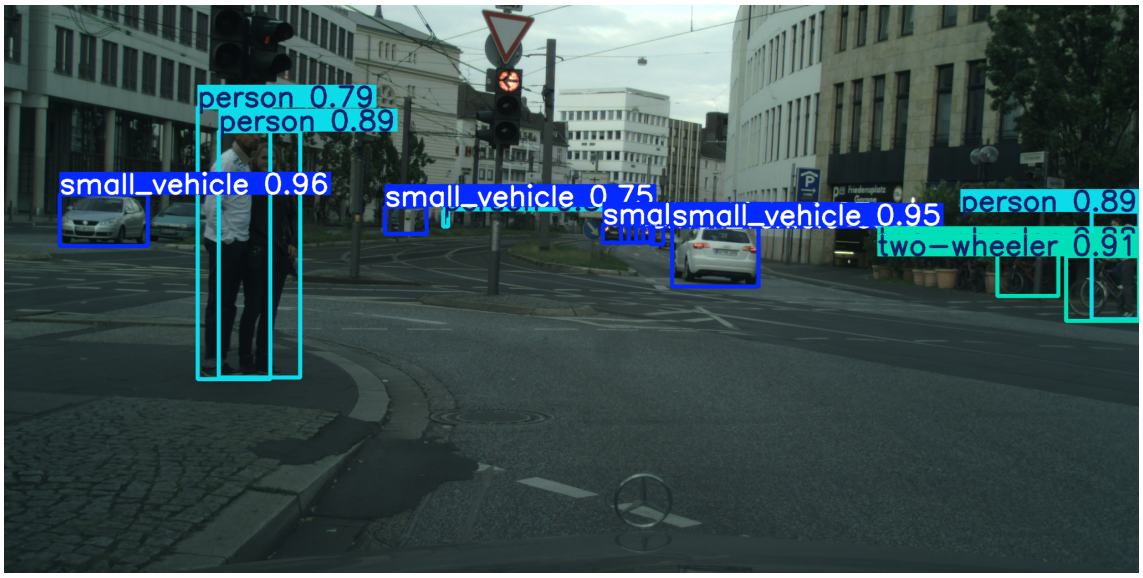
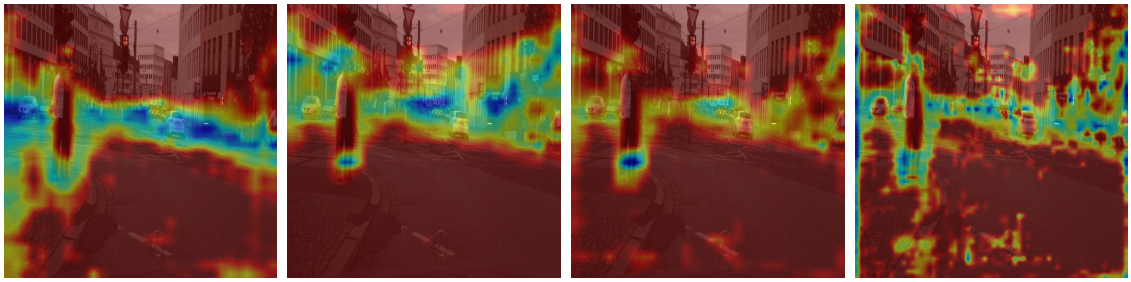


Figure 9: Model's detection to the picture Bonn35

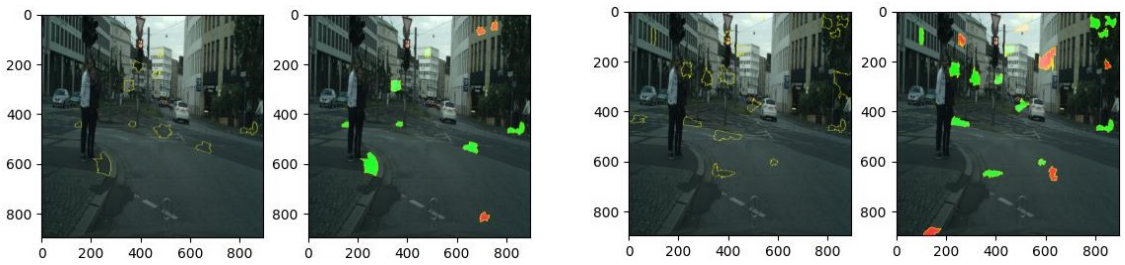


(a) Layer -2 **(b)** Layer -3 **(c)** Layer -4 **(d)** Layer -5

Figure 10: Activation maps for the Layer -2, -3, -4, -5 of Bonn35

that the rough edges from the more inner layers in the network exhibited a process of smoothening layer to layer.

Evaluation of the LIME method In the figure constellation of Figure9 different types of LIME functions are presented. These are LIME_SUM and LIME_MULTI. LIME_MULTI aggregates the contributions of each feature across multiple instances by summing the importance scores. This approach provides a simplified, overall view of how certain features influence the model's decisions on average, making it easier to identify the most significant features contributing to the model's predictions in a broad sense. LIME_SUM, on the other hand, focuses on explaining predictions at a finer level. It applies LIME to multiple instances individually and then analyses the distribution of feature importance across these instances. This method captures more granular information, allowing for a more detailed understanding of feature interactions and their varying contributions across different samples.



(a) LIME_SUM **(b)** LIME_MULTI

Figure 11: LIME results

Evaluation of the SHAP method In the figure10 SHAP is run to the image, and the shap values are clearly visible on the pictures, each picture represents the run on one instance of detected object.

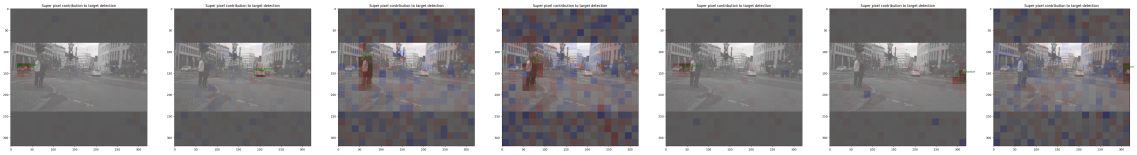


Figure 12: SHAP results

Red representing higher positive SHAP values, while blue represents negative SHAP values. Considering the most of the pictures, especially where overrepresented, highly visible objects can be seen, the interpretation is clean and readable.

5.3 Addressing the Anomaly regarding the noise of SHAP and the probability of the models detection

As previously stated in the previous subsection 6.2, the SHAP values for objects with lower detection probability are characterised by a high degree of noise, rendering them unable to provide a clear and coherent explanation for the model's predictions. This anomaly can be attributed to the low probability of detection for this instance pedestrian class, which results in unreliable, often contradicting information for the SHAP algorithm to interpret. This phenomenon is common in object detection models, where the detection accuracy for small objects is lower than that for larger or more clearly visible objects. Furthermore, as previously discussed in section **??** that the model's detection threshold and the probability of detection can have a significant impact on the actual decision-making process of the model. The combination of perturbation-based interpretation methods may result in unreliable and noisy results, as the perturbed image segments may be misclassified by the model.

Furthermore, an additional cause of this noise can be identified, which is more comprehensible when considering the proportions of the various classes of traffic objects present in the dataset. As illustrated in Figure **??** the dataset exhibits a significant class imbalance, with certain classes, such as vehicles, being overrepresented in comparison to others, such as pedestrians. This imbalance may result in the generation of noise in the model's predictions, as the model may exhibit a tendency to favour the more frequent classes due to the larger volume of data from which it has learned. Consequently, the limited number of examples for less frequent classes, such as pedestrians, may not provide the model with sufficient information to effectively capture their distinctive and often subtle features.

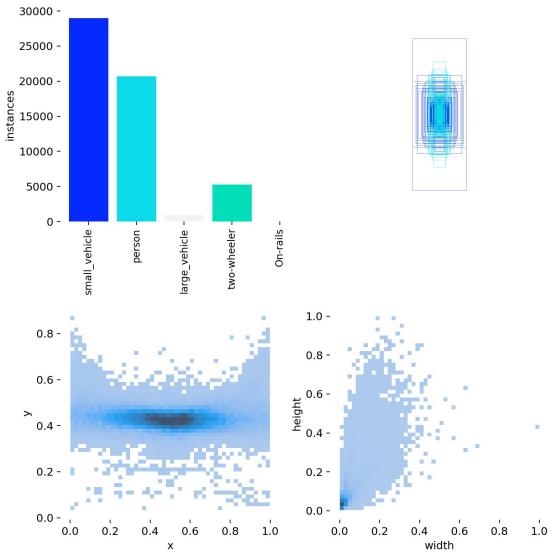


Figure 13: Label distribution

This deficiency in the training data can give rise to a number of issues. For example, the model may encounter difficulties in generalising effectively when it encounters pedestrian objects in real-world scenarios, due to insufficient exposure to their diverse appearances and contexts during training. Consequently, the model may misclassify or fail to identify pedestrians, which can introduce additional noise in its output, manifesting as false negatives or incorrect predictions. Furthermore, the distinctive characteristics that differentiate pedestrian objects, such as specific shapes, movements, or interactions with the environment, may be underrepresented in the model's learned representations. To mitigate this noise and enhance detection accuracy, it is essential to balance the dataset or augment it with more diverse examples of the minority class, thereby enhancing the model's ability to learn these crucial features effectively.

To address this issue, an increase in the model's detection threshold for the noisy object'S class may be beneficial in improving the detection probability of already detected objects. This could be achieved by ignoring detections with a lower prob-

ability, thus enhancing the reliability and decreasing the noise of SHAP interpretations. This adjustment can help reduce the noise but may also result in the exclusion of some objects from the interpretation process. Making the interpretation process more incompletely, but more reliable and less noisy.

6 Results

SHAP:

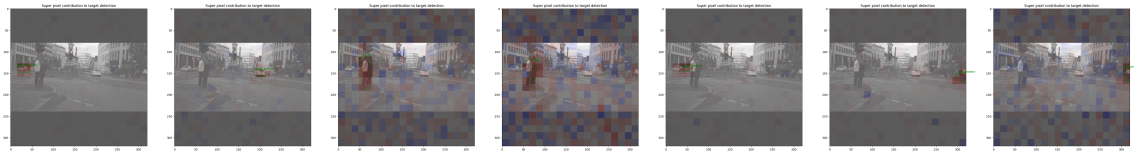
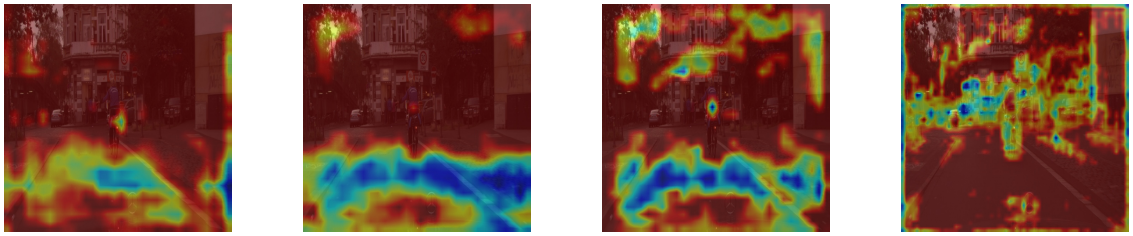


Figure 14: SHAP results

Figure 15: SHAP results



(a) Layer -2 **(b)** Layer -3 **(c)** Layer -4 **(d)** Layer -5

Figure 16: Activation maps for the Layer -2, -3, -4, -5 of Bonn36

List of Figures

Bibliography

[1] Nidya Chitraningrum, Lies Banowati, Dina Herdiana, Budi Mulyati, Indra Sakti, Ahmad Fudholi, Huzair Saputra, Salman Farishi, Kahlil Muchtar, and Agus Andria. Comparison study of corn leaf disease detection based on deep learning yolo-v5 and yolo-v8. *Journal of Engineering and Technological Sciences*, 56(1):66, Feb. 2024. DOI: 10.5614/j.eng.technol.sci.2024.56.1.5. URL https://jets.itb.ac.id/index.php/jets/article/view/6.

[2] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[3] Arun Das and Paul Rad. Opportunities and challenges in explainable artificial intelligence (xai): A survey. page 1, 2020. URL https://arxiv.org/abs/2006.11371.

[4] L. Ezzeddini, J. Ktari, T. Frikha, N. Alsharabi, A. Alayba, A. J. Alzahrani, A. Jadi, A. Alkholidi, and H. Hamam. Analysis of the performance of faster r-cnn and yolov8 in detecting fishing vessels and fishes in real time. *PeerJ Computer Science*, 10:e2033 9/15, 2024. DOI: 10.7717/peerj-cs.2033.

[5] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:355–359, 2018. ISSN 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.10.013. URL https://www.sciencedirect.com/science/article/pii/S0031320317304120.

[6] ISO/IEC. 5469: Artificial intelligence – transparency and explainability, 2021.

[7] Rui-Yang Ju and Weiming Cai. Fracture detection in pediatric wrist trauma x-ray images using yolov8 algorithm. *Scientific Reports*, 13, 11 2023. DOI: 10.1038/s41598-023-47460-7.

[8] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. pages 4–5, 2020. URL https://arxiv.org/abs/2006.04388.

[9] Yu Liang, Siguang Li, Chungang Yan, Maozhen Li, and Changjun Jiang. Explaining the black-box model: A survey of local interpretation methods for deep neural networks. *Neurocomputing*, 419:168–176, 2021. ISSN 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.08.011. URL https://www.sciencedirect.com/science/article/pii/S0925231220312716.

[10] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. pages 4–6, 2017. URL https://arxiv.org/abs/1705.07874.

[11] Mohammed Bany Muhammad and Mohammed Yeasin. Eigen-cam: Class activation map using principal components. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–3. IEEE, July 2020. DOI: 10.1109/ijcnn48605.2020.9206626. URL http://dx.doi.org/10.1109/IJCNN48605.2020.9206626.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 2016. URL https://arxiv.org/abs/1506.02640.

[13] Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10): 5396, 2020.

[14] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, October 2019. ISSN 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL http://dx.doi.org/10.1007/s11263-019-01228-7.

[15] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):311–314, 2017.

[16] Shuhan Song. The use of color elements in graphic design based on convolutional neural network model. *Applied Mathematics and Nonlinear Sciences*, 9, 10 2023. DOI: 10.2478/amns.2023.2.00536.

[17] UltraLytics. GitHub - ultralytics/ultralytics at v8.2.103 — github.com. https://github.com/ultralytics/ultralytics/tree/v8.2.103. [Accessed 20-10-2024].

[18] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. *CoRR*, abs/1911.08287:4, 2019. URL `http://arxiv.org/abs/1911.08287`.

[19] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–3, 2016. DOI: `10.1109/CVPR.2016.319`.