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 21, 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 21.

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 the movement of vehicles is a critical task in the field of automotive and transportation systems. Numerous scenarios can arise where different types of object detection is required. The development of a highly sophisticated automated driving system requires the efficient gathering and processing of data from sensors located in the chassis to create a virtual model of the surrounding world.

The primary challenge of developing autonomous vehicles is the development of systems to create a reliable and accurate perception of the vehicle's environment. One of the most widely used solutions 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 fully fulfilling this need by detecting traffic objects in real-time while staying robust and reliable in various conditions. A perfect solution for this task are the deep learning models, especially Convolutional Neural Networks (CNNs).

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. This lack of transparency in how models arrive at conclusions has led to the development of numerous model interpretation techniques, as evidenced by the work of a number of different researchers such as Yu Liang [8]. 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. Additionally, there are several ISO standards that are dedicated to the safety of AI systems, including ISO 5469[5], which sets forth the requirements for ensuring the intended functionality of AI systems . Furthermore, it addresses the necessity for XAI methodologies to be employed in the advancement 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-time scenarios.

In addition to real-time 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, with a particular focus on understanding their underlying principles and applying them to the traffic detection model. The efficacy of these interpretation techniques will be evaluated by assessing their capacity to elucidate 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 study 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 optimising 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

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 segmentation. Given the ability of convolutional neural networks (CNNs) 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 ResNet. 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" [4].

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, outside convolutional layers, CNNs also comprise pooling layers, fully connected layers.

These models are employed in a multitude of applications, including autonomous vehicles, surveillance systems, and medical imaging. As these applications align with the project's objectives, these types of neural networks are selected for the task of traffic object detection.

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

Convolutional layers Convolutional layers represent the fundamental building blocks of convolutional neural networks (CNNs). 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 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),trough backpropagation, allowing the network to enhance feature extraction for specific tasks.

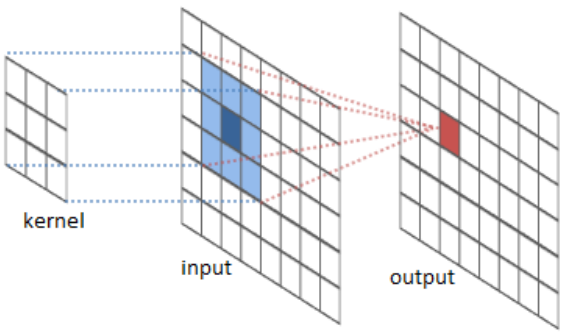


Figure 1: Convolutional layer operation [16]

Sampling layers The objective of sampling layers is to reduce the dimensionality of the data set while ensuring the preservation of the input data's essential features. Sampling can entail the application of techniques such as subsampling or strided convolutions, whereby a specific stride is applied to the convolution operation in order to down-sample the feature maps.

Pooling layers The objective of pooling layers is to reduce the spatial dimensions of feature maps given by the convolutional layers, thereby decreasing the number of parameters and computations in the network. Moreover, this process enhances the model's resilience to minor translations in the input data by mitigating the effects of potential outliers. The most prevalent forms of pooling are max pooling and average pooling. Max pooling selects the maximum value from a specified window, whereas average pooling computes the average. By down-sampling the feature maps, pooling layers assist in maintaining the most important features while discarding less critical information from the picture, also efficient in reducing the computational load.

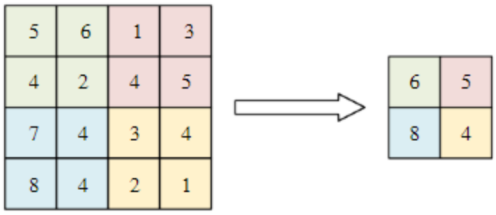


Figure 2: Pooling layer operation [16]

Fully connected layers Fully connected layers, also referred to as dense layers, are typically located at the conclusion(HEAD) of convolutional neural network (CNN) architectures. In these layers, each neuron is connected to every neuron in the preceding layer. This structure enables the model to integrate information from all features and make final predictions. Fully connected layers are particularly useful in classification tasks, where they compute the output probabilities for each class based on the features extracted by the preceding layers. Regularisation techniques, such as dropout, are frequently employed in these layers to prevent overfitting.

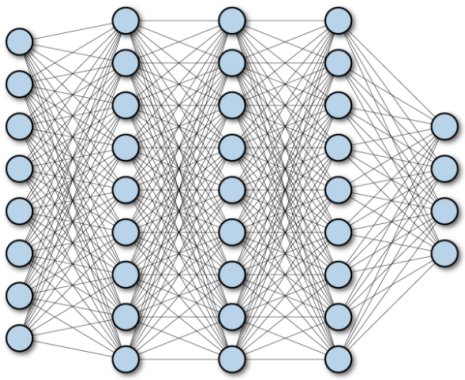
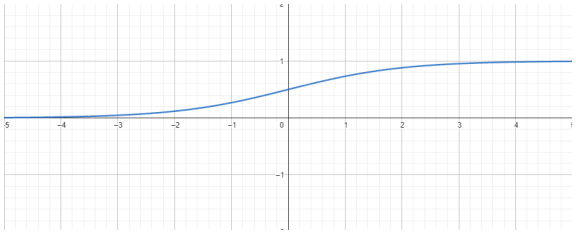


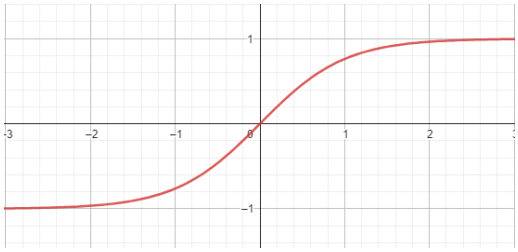
Figure 3: Fully Connected Layer semantics [16]

2.2 Activation Functions

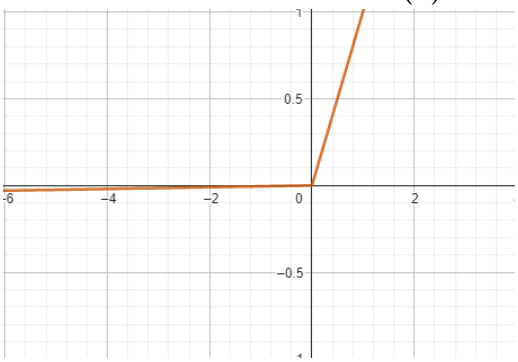
Activation Functions are utilised in CNNs(and other Neural networks) to introduce non-linearity into the model. They assist the model in learning complex patterns and making accurate predictions. In light of the work conducted by Siddharth Sharma and Simone Sharma regarding activation functions [15] the most prevalent activation functions are ReLU, Sigmoid, and Tanh. 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 neural networks (CNNs). 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 approaches the problem as a single regression problem. The algorithm divides the image into a grid and predicts bounding boxes and class probabilities parallel for each grid cell, thereby enabling the detection of multiple types and instances of objects in a single run. This architecture not only enhances speed but also improves detection accuracy by reducing the number of false positives.

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 plethora of applications, including autonomous vehicles, as evidenced by the author's own observations, 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.

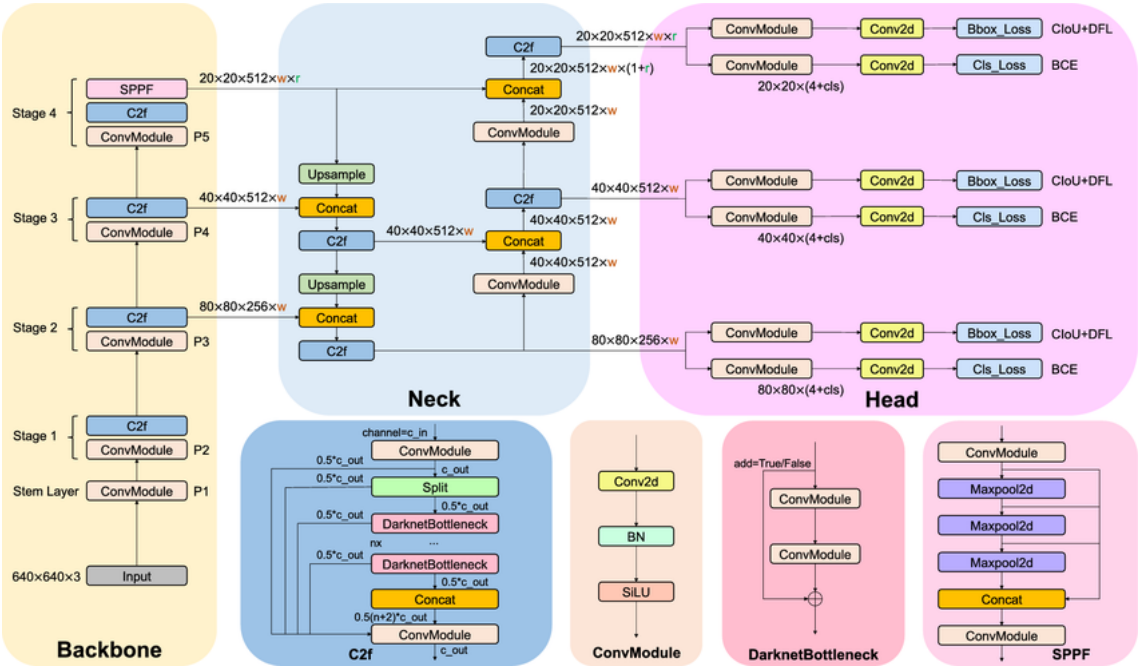


Figure 6: The architecture of the YOLOv8 model [6]

Backbone The YOLOv8 model is based on convolutional neural networks (CNNs) 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 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 labeled 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**:

$$\text{CIoU} = 1 - \left(\text{IoU} - \frac{\rho^2(\mathbf{b}, \mathbf{b}^g)}{c^2} - \alpha v \right) \text{ [7]}$$

  where ρ is the Euclidean distance between the center 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, and v measures the consistency of aspect ratio. It is more sensible to localisation accuracy.

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

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

  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)**:

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

  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.

2.6 Training and evaluation 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.

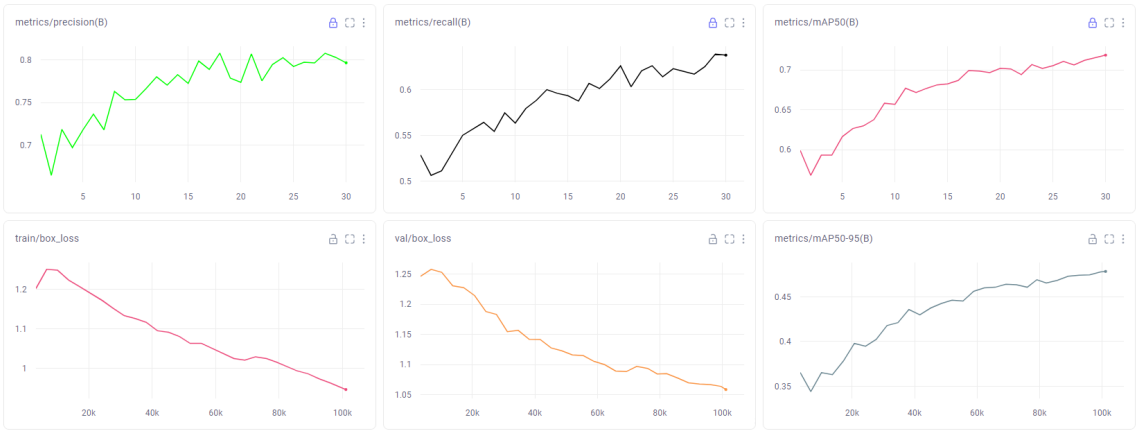


Figure 7: Comet.ml dashboard about our the moodel

Figure **??** 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 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 Object detection component implementation

3.1 Dataset and formats

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 " Thecityscapes dataset for semantic urban scene understanding" the gtFine dataset consists of 5000 images, with a resolution of 1024x2048 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. 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 wheelers***(bicycles and motorcycles)*,
- **On-rails***(trains and trams though the smaller Fine dataset didn't include any)*
- 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) 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.
  - **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 the class labels and their corresponding indices, as well as path set descriptor txt-s. 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.

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

The Faster R-CNN while being a less sophisticated model it gets outperformed by Yolo on, a comparison study on these networks "Performance Study of YOLOv5 and Faster R-CNN for Autonomous Navigation around Non-Cooperative Targets" [10] it has a table which contains all the necessary information to decide that it is inferior to Yolov8 in almost every aspect

I tested Yolov5 on the same dataset and produced inferior performance KPI(Key Performance Indicator)-s. 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.

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.

Chosen model's description and parametrisation Based on paragraph **??** the Yolov8 bounding box detection model's medium version was picked for this project. n 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]. 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 and lrf. 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.

3.3.2 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.

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

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.

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

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

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 improving model performance. In this section, I discuss the importance of model interpretation and review various interpretation methods. In their survey [8], Liang et al. highlight the significance of model interpretation techniques in providing insights into the decision-making process of models. They sort these techniques into two categories:

1. Data Driven or more commonly Model-agnostic methods
2. Model Driven or Model-specific methods

Importance of Interpretation Interpreting machine learning models is essential for understanding their decision-making processes. It helps in identifying biases, improving model performance, and building trust with users. Interpretation techniques provide insights into how 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 legislations, standards and regulations that require models to be interpretable, and provide explanations for their decisions.

4.1 Model Agnostic Methods

Based on the work of Liang [8], model-agnostic interpretation methods can be divided into two categories:

1. Perturbation-based interpretation
2. Game theoretic approach

As discussed in the work of Liang [8], 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. 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.

4.1.1 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 adding noise to the input image and observe the changes 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.

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.

Game theoretic approach with perturbation Game-theoretic approaches with perturbation are model-agnostic interpretation methods 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". The aforementioned parts constitute a set. In accordance with the implementation of this approach, SHAP, assigns a Shapley value to each feature. Based on its contribution to the model's prediction. This is calculated by considering all potential combinations of parts. These combinations represent all the subsets of the set of parts, and the Shapley value is calculated for each subset. The greater the variation in the model's prediction when a feature is added or removed, the higher the Shapley value. By considering all the potential combinations of features, these methods offer consistent and accurate explanations for model predictions.

4.1.2 Comparison of Model Agnostic Methods

Based on the work of Liang [8], these model agnostic interpretation methods have their strengths and weaknesses. Two of the most popular model-agnostic interpretation methods are LIME(as an example for Perturbation-based Interpretation) and SHAP(as an example for Game theoretic approach with perturbation). Through their example I will illustrate the differences between the two methods.

LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) 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 [9], 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.

4.2 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 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] and its implementation 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].

4.2.1 Class Activation Maps

Class Activation Maps (CAMs) are a model-specific interpretation method designed to highlight the regions or features in an imag 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 final 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 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 versatility. 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 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 an object 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].

4.2.2 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 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, while beign more accurate than vanilla EigenCAM, I did not use this in my project due to several considerations. First of all, the method is more computationally expensive, and requires more memory to run, and was outweighted by the benefits of the CAM method, or other model-agnostic methods. Secondly, the method is more complex to implement, and requires more knowledge of the model's internal state, which state should have been changed to accommodate the method, which made the others more difficult to implement.

4.2.3 Comparison of Model Specific Methods

Both CAM and Grad-CAM methods have been widely adopted for interpreting convolutional neural networks (CNNs) and are valuable tools for visualizing and understanding model behavior. 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.

4.3 Comparison of Interpretation Methods

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 are designed to be applicable to any machine learning model, irrespective of the model's architectural or structural configuration. These techniques operate by treating the model as a black box and analyzing the input-output relationship without needing to access the model's internal workings. 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 **??**. 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 explanations 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, 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 generalizability versus precision. Model-agnostic methods are particularly usefull 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

EigenCAM EigenCAM is a model-specific interpretation method that visualizes the regions of an image that are most important for a model's prediction. It computes the principal components of the feature maps and highlights the areas that contribute the most to the final decision.

Local Interpretable Model-agnostic Explanations LIME is a widely used model-agnostic interpretation method that explains individual predictions by approximating the model locally with an interpretable model. It perturbs the input data and observes the changes in the model's predictions to identify the most important features.

Shapley Additive explanations SHAP is another model-agnostic method that provides consistent and accurate explanations for model predictions. It is based on cooperative game theory and assigns a Shapley value to each feature, representing its contribution to the prediction.

5.2 Evaluation of interpretation methods

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

6 Results

SHAP:

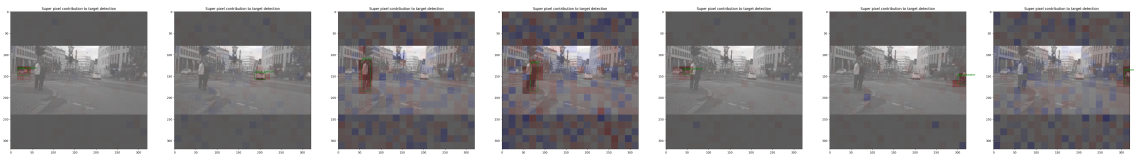


Figure 8: SHAP results for the small vehicle class

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)*, 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] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] Trupti Mahendrakar, Andrew Ekblad, Nathan Fischer, Ryan White, Markus Wilde, Brian Kish, and Isaac Silver. Performance study of yolov5 and faster r-cnn for autonomous navigation around non-cooperative targets. In *2022 IEEE Aerospace Conference (AERO)*, page 8, 2022. DOI: 10.1109/AERO53065.2022.9843537.

[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. 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.