Stabilizing Non-Maximum-Suppression

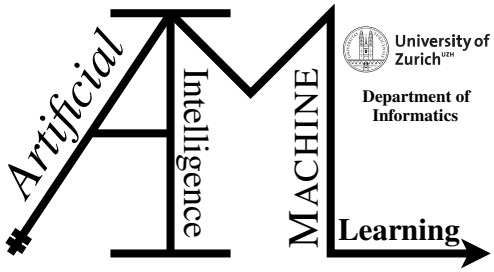
More Stable Replacement for Non-Maximum-Suppression in Object and Face Detectors

Master Thesis

Pascal Engeli
14-671-457

Submitted on
August 26, 2021

Thesis Supervisor
Prof. Dr. Manuel Günther



Master Thesis

Author: Pascal Engeli, pascal.engeli@uzh.ch

Project period: February 22, 2021 - August 26, 2021

Artificial Intelligence and Machine Learning Group
Department of Informatics, University of Zurich

Acknowledgements

At this point, I want to thank my parents Claudia and Markus, for their endless and unconditional support throughout my education and every other life situation. You motivated me and acted as role models to always strive for challenging goals. You did not doubt me when I did not make it to secondary school. Here I am now, turning in my Master's thesis at the University of Zürich and still full of curiosity to learn more. I dedicate this work entirely to you since you anyway paid for it.

I also want to sincerely thank Prof. Dr. Manuel Günther for allowing me to work on this fascinating topic, his regular support, and time to advise me to achieve the best possible with this thesis. You reached to spark my interest in object and face detection, which is still glowing after countless hours invested in this topic.

Abstract

The predictions of object and face detectors suffer from unstable bounding boxes. The main reason for this problem is the post-processing algorithm Non-Maximum Suppression. During the suppression of redundant bounding boxes, only the most confident box does not get removed. This behavior can be observed when inspecting video sequences. Even marginal change in pixel values causes the detected bounding boxes to jitter. In this work, a method is proposed to further inspect this problem by fabricating sequences of augmented static images to simulate moving objects and faces. Combining this approach with an evaluation metric from video detection leverages the quantification of temporal and spatial stability of detected bounding boxes compared to their associated ground truth annotations. Simultaneously, two alternative Non-Maximum Suppression algorithms are proposed to solve the problem of jittering bounding boxes. The algorithms are called Average and Average IoU Non-Maximum Suppression. Both alternatives consider aggregating the overlapping bounding boxes and their detection scores using a weighted average of the individual coordinates and class probabilities. An increase in stability can be evidenced by implementing Average and Average IoU NMS into post-processing the multi-stage detectors Faster R-CNN and MTCNN and comparing it to their default NMS function. By evaluating the object and face detectors on the MS COCO, PASCAL VOC, and WIDER Face datasets, even an improvement in accuracy can be observed.

Zusammenfassung

Die Ergebnisse der Lokalisierung und Klassifizierung von Objekt- und Gesichtsdetektoren, die mittels objekt- oder gesichtseingrenzenden Rechtecken erkenntlich gemacht werden, leiden unter Instabilität. Das Hauptaugenmerk liegt dabei auf dem in den Detektoren nachgelagerten Algorithmus Non-Maximum Suppression (NMS). Während dem Auswahlverfahren von NMS werden alle Rechtecke eliminiert, die sich mit einem benachbarten Rechteck überschneiden und welche über einen nicht maximalen Klassifikationswert verfügen. Auswirkungen dieses Selektionsverfahrens können speziell in der Objekt und Gesichtserkennung in Videosequenzen festgestellt werden. Noch so minimale Änderungen in den Pixelwerten eines einzelnen Ausschnittes führen zu wackelnden Rechtecken. Zur zielgerichteten Untersuchung dieses Problems, wird in der vorliegenden Arbeit eine neue Methode vorgeschlagen. Diese Methode beinhaltet das sequentielle Zusammenführen von augmentierten Einzelbildern, womit die Bewegung von Objekten und Gesichtern simuliert werden kann. Diese Methode ermöglicht die zielführende Anwendung einer Bewertungsmetrik zur Messung von zeit- und raumbezogener Stabilität dieser Rechtecke. Die Messung kann mittels vorhandener Annotationen mit einem Soll-Zustand verglichen werden. Zeitgleich werden zwei alternative Selektionsverfahren vorgeschlagen, die zur Lösung des Problems von wackelnden Rechtecken ihren Beitrag leisten. Dabei handelt es sich um Average und Average IoU NMS. Beide Alternativen beziehen den gewichteten Mittelwert von allen benachbarten Rechtecken bei, um daraus eine resultierende Vorhersage zu berechnen. Zu der Vorhersage gehört ein objekt- oder gesichtseingrenzendes Rechteck und ein Klassifikationswert. Die gemeinsame Umsetzung des vorgeschlagenen Ansatzes zur Messung von Stabilität in Einzelbildern kombiniert mit den alternativen Selektionsverfahren wird mittels vorhandener Netzwerke realisiert. Zu den verwendeten Netzwerken gehören Faster R-CNN und MTCNN unter Berücksichtigung der Datensätze von Microsoft COCO, PASCAL VOC und WIDER Face. Anhand dieser Implementierung kann eine erhöhte Stabilität der Ergebnisse im Vergleich mit herkömmlichen Methoden zur Selektion gemessen werden. Ausserdem kann den alternativen Selektionsverfahren auch eine Verbesserung der Genauigkeit nachgelegt werden.

Contents

Chapter 1

Introduction

Object and face detection are fundamental tasks in computer vision, which enable many subsequent tasks, such as segmentation, pose estimation, object and face tracking, or action recognition. Holistically viewing, computer vision empowers the technological realization of, for instance, self-driving cars, reading hand-written characters, medical image analysis, video surveillance, and autonomous disease detection by tracking honey bees.[1] Computer vision even enables us to unlock mobile phones using our faces. These achievements are due to significant improvements in recent years concerning Deep Learning in general and convolutional neural networks (CNNs) in particular (Bodla et al., 2017; Wu and Li, 2021; Zou et al., 2019; Tripathi et al., 2020).

Generally, the problem of detecting objects and faces in static images is conducted in three interdependent steps: (I) Proposing hundreds of regions with various sizes where any object or face might be located using bounding boxes. (II) Classifying these bounding boxes. (III) Suppress redundantly classified bounding boxes. Between steps I and II and especially in step III, object and face detectors rely on an algorithm called Non-Maximum Suppression. This algorithm takes overlapping bounding boxes into account and removes all redundant boxes but one. The remaining bounding box survives this selection process by being the fittest in terms of being most confident of surrounding an object among all other suppressed boxes. By relating to the initially mentioned applications of computer vision, many of those include continuous video sequences. The problem of detecting objects and faces in videos becomes dynamic. Object and face detectors solve this problem by breaking down this dynamic object and face detection task in videos by reducing it back to detecting these objects and faces in single frames. Consequently, the same three general steps can be applied for every frame in the video sequence. The difference to the static problem is that a subsequent frame might feature any change to the previous frame. This change can be monumental in significant movements or minimal in slight camera noise, not recognizable by human visual perception. Nevertheless, any change might cause the three general steps to be performed on different pixel values of the frame. Hence, the selection process of overlapping bounding boxes is performed with new confidences. This time, a neighboring bounding box of the previously maximally scoring box might get selected as the fittest. Only the differently selected, thereby jittering bounding box becomes evident for the observant by visually inspecting this video sequence with such marginal changes (Bodla et al., 2017; Zhou et al., 2017; Wu and Li, 2021; Zhang and Wang, 2016).

The accuracy perspective often neglects these jittering bounding box detections when evaluating the object and face detectors in their ability to localize and classify objects. The evaluation process of such detectors relies exclusively on the comparison between detected bounding boxes and ground truth annotations. A detection is accounted as successful and accurate as long as it shares an overlap with its associated ground truth annotation. Slight deviations from the ground

[1] Beelivingsensor: https://en.beelivingsensor.org/

truth are tolerated. There is not much evidence in the literature on using different evaluation procedures in image detection compared to video detection apart from treating single video frames as static images. However, in direct comparison with the amount of literature available in object and face detection accuracy, findings concerning the problem of jittering bounding boxes are rare. Solving the problem of varying marginal yet noticeable deviations from the ground truth and the previously detected bounding boxes has not come to recent research attention. This problem adds a complementary perspective on evaluating object and face detectors in terms of stability to the already well-established accuracy evaluation perspective (Zhang and Wang, 2016).

Concerning these two exhibited problems of unstable detections of objects and faces and the limited measurability of this instability, this work intends to bridge the identified research gaps. This thesis, therefore, aims to provide an alternative Non-Maximum Suppression algorithm to stabilize bounding box detections during the selection process. A further goal is to make the stability of bounding box detections compared to its associated ground truth annotations quantifiable. Therefore, the following two research questions are addressed in this work:

- Research question 1: How does a weighted average approach of Non-Maximum Suppression influence the stability of bounding boxes in object and face detection?
- Research question 2: How can the stability of bounding box detections be quantified and measured based on static images?

In order to find an answer to these two research questions, first, a thorough literature review is conducted. The aim of this literature review is to identify potential image sources, capable object and face detectors, recent trends in Non-Maximum Suppression optimization, and starting points for extending the evaluation perspective. As a second step, alternative Non-Maximum Suppression algorithms are proposed and a benchmark established. The third step represents a proposition to turn static images into sequences to simulate object and face movements. The first, second, and third steps are then combined in a fourth step and explained by implementation in an experimental setup to answer the two research questions. The fifth step presents the results of the experimental setup and delivers quantifiable evidence. These shreds of evidence are finally summarized, discussed, and concluded to answer the two above-stated research questions.

The implementation of this work in Python can be found on Gitlab.[2] Only the code regarding the first research question is presented in its entirety in appendix B. One third of the code to answer research question 2 is exemplarily shown also in appendix B. For the sake of environmentalism, no further pasting, and therefore, printing of computer code will be done.

[2] Stabilizing NMS on Gitlab: https://gitlab.ifi.uzh.ch/aiml/theses/engeli

Chapter 2

Related Work & Background

2.1 Datasets

The very first step in object and face recognition is the selection of relevant data. Here, two datasets are the most commonly analyzed and reported in object detection tasks: Microsoft COCO and PASCAL VOC. Since these datasets are often used to compare either entire frameworks or incremental changes in frameworks, such as replacing incremental parts, there are a reasonable number of benchmarks against which one's implementation can be measured. Therefore, for the object detection task, both datasets are used. On the other hand, the focus of this work is also on face detection. For the same reasons as object detection, there are datasets for face detection commonly used in the literature. The dataset selected for the face detection task is the WIDER Face dataset. In this section, the three selected datasets are presented in the following.

2.1.1 Microsoft COCO

The Microsoft COCO datasets consist of images in complex contexts; therefore, the name "Common Objects in Context" (COCO) (Lin et al., 2014). The Microsoft COCO datasets are referred to plural because there are multiple versions available online for yearly challenges on object detection, keypoint detection, panoptic segmentation, and stuff segmentation.[1] As seen in the related work chapter, the most commonly used dataset for evaluating object detection is the Microsoft COCO 2017 version, which is referred to as COCO going forward. Even though the submission deadline for the 2017 object detection challenge lies in the past, the dataset is still commonly used for benchmarking models in object detection tasks.

The 2017 MS COCO dataset is split into three parts: 118'000 training images, 5'000 validation images, both with dedicated annotations, and 41'000 test images without annotations.[2] In order to evaluate the performance of a pre-trained model, only the validation set will be used since the test set can only be evaluated on a limited frequency upon online submission.[1] However, the validation, as well as the training and testing set, follow the same class distribution with images containing one or multiple instances of 80 different object classes.[3] These 80 individual object classes are grouped into 12 super categories such as person, vehicle, outdoor, animal, accessory, sports, etc. The class distribution plot is depicted in figure 2.1

The images are publicly available in jpeg format within the validation set and refer to an annotation instance in a JSON file. In total, the MS COCO validation labels dispose of 36'781 ground truth annotations, each referring to an object in one of the 5'000 validation images. Each ground

[1] COCO Guidelines: https://cocodataset.org/#guidelines
[2] COCO Download: https://cocodataset.org/#download
[3] COCO Detection: https://cocodataset.org/#detection-2017

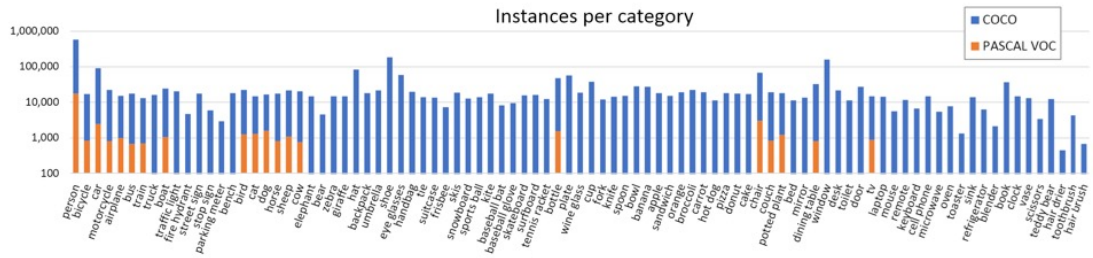


Figure 2.1: CLASS DISTRIBUTION COCO & PASCAL VOC DATASETS. The class distribution of COCO and PASCAL VOC datasets. Objects of the PASCAL VOC dataset are a subset of the objects in the COCO dataset. *Source: (Lin et al., 2014)*

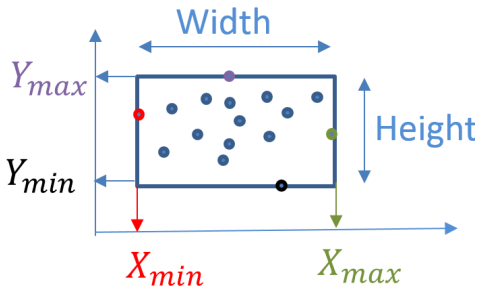


Figure 2.2: BOUNDING BOX COORDINATES. An example how bounding box coordinates are annotated. *Source: (Gomaa et al., 2019)*

truth instance has seven attributes. Among those attributes are identifiers such as a unique object id and image id for cross-references between images and objects. There are also two types of coordinate attributes in the form of segmentation and bounding box coordinates. The segmentation coordinates indicate the detailed outer shape in the form of a mask of the object, and the bounding box coordinates enclose all the segmentation coordinates in a rectangle. For the task of object detection of this thesis, only the bounding box coordinates are relevant. An example of such coordinates is displayed in figure 2.2.

2.1.2 PASCAL VOC

The second dataset utilized in this thesis not only shares the name with the author of this thesis but is also widely used in the literature for object detection. The Pattern Analysis, Statistical Modelling, and Computer Learning (PASCAL) Visual Object Class (VOC) challenge also consists of a publicly available dataset and an annual competition for classification, detection, segmentation, action classification, and person layout (Everingham et al., 2010). These competition series held place between 2005 and 2012.[4] For comparison purposes with existing research in object

[4] PASCAL VOC 2012: http://host.robots.ox.ac.uk/pascal/VOC/

detection and localization, the present thesis uses the dataset affiliated with the PASCAL VOC challenge from 2007. Similar to the COCO dataset breakdown, the dataset from the PASCAL VOC 2007 challenge disposes of a total of 9'963 annotated images and is divided into training, validation, and testing data. However, unlike the COCO test dataset, after the submission deadline for the PASCAL VOC 2007 challenge, the annotations for the test data were made publicly available. Therefore, the test set as the largest part has 4'952 images with 12'032 annotated objects.[5] The training set contains 2'501 images and 6'301 objects.[5] And lastly, the validation set features 2'510 images with 6'307 objects located within these images.[5] All of the three parts of the PASCAL VOC 2007 challenge follow approximately the same object class distribution with 20 different objects and are grouped into four categories of person, animal, vehicle, and indoor objects.[6] [5] The class distribution plot together with the COCO classes is shown in figure 2.1

Since the test dataset provides the most annotated data, this dataset is used for the thesis at hand. Similar to the COCO images, the 4'952 PASCAL VOC test images can be downloaded in jpeg format. Each image refers to one annotation file in XML format with the ground truth objects located within the referenced image. The ground truth consists of five attributes: class labels, bounding boxes, view angle, whether the whole object is visible or if it is truncated, and a boolean value of difficulty to recognize the object.[5] Compared to the ground truth annotations of the COCO dataset, the bounding boxes differ concerning the coordinates of the width and height.

2.1.3 WIDER Face

The third dataset differentiates itself from the datasets as mentioned earlier in the singularity of the object class. The WIDER Face dataset is a subset of the WIDER dataset but only features images with faces in it. WIDER Faces consists of 32'203 images which are evenly split into training and testing sets with a total of 393'703 labeled face bounding boxes (Yang et al., 2016). The labeled faces vary in scale, pose, and occlusion . Yang et al. (2016) proposes a random selection of 40% for training, 10% for validation, and 50% of the total data as testing data. In contrast to COCO and PASCAL VOC datasets, the WIDER Face dataset did not originate from a dedicated annual challenge. However, it was constructed for the sake of research to provide a benchmarking dataset. Therefore, the availability of the dataset is established through the privately hosted website of the author.[7] Even though the WIDER Face benchmark does not follow a competition procedure with a strict deadline, the annotation file to the test set (50% of the data) is not publicly available. Therefore, only the validation set can be used for this thesis, which results in the availability of a dataset containing 3'226 images with 39'697 annotated faces.

As seen with COCO and PASCAL VOC, the WIDER Face validation images are also provided in jpeg format. The annotations to the images are stored in a single text file which can be parsed line by line. The first line of the annotation file indicates the location of the associated image. A single integer value is written in the second line with the number of annotations for this particular image. The following lines contain the annotations themselves with ten integer values split by white space. This sequence repeats for all the referenced images in the validation set. The first four attributes of the annotation lines indicate the bounding box of the labeled face with the same information provided by the COCO ground truth in figure 2.2.

The form of the annotated bounding box means that the minimum values for the bounding box on the x and y axis come as indices 0 and 1. At indices 2 and 3, the width and height are declared. The remaining six attributes give information about the visual context of the labeled face in terms of blur, expression, illumination, occlusion, pose, and validity of the labeled face. Since

[5] PASCAL VOC Development: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/htmldoc/index.html

[6] PASCAL VOC 2007: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html

[7] WIDER Face: http://shuoyang1213.me/WIDERFACE/

only faces are labeled, the annotations of WIDER Face do not come with a class affiliation. However, the images of WIDER Face are split into 60 event categories, such as traffic, riot, football, interview, or family group (Yang et al., 2016). For each of these 60 event categories, the images were characterized based on the factors of scale, occlusion, and pose (Yang et al., 2016). With the help of these factors, the detection rate was determined, which led to partitioning into three different difficulties: easy, medium, and hard (Yang et al., 2016). Intending to stabilize bounding boxes and, for this particular dataset, the facial landmarks, it is more reasonable to focus on images that are not small-scaled and not occluded. Therefore, only the images within the event categories classified as easy are regarded for the analysis. With this restriction, the WIDER Face validation set gets reduced to 985 images and 5'645 annotations. By inspecting the annotations it came clear that one image is annotated with an invalid bounding box ($x_{min} = 0, y_{min} = 0, w = 0, h = 0$). To prevent the invalid load of annotations, a check has to be implemented to ignore such bounding box annotations. Because that invalid bounding box was the only annotation for this image, also the image is ignored for this analysis. With this final reduction, the WIDER Face validation dataset contains 984 images with 5'644 ground truth annotations.

Dataset Overview		
Dataset	Number of images	Number of ground truth annotations
COCO	5'000	36'781
PASCAL VOC	4'952	12'032
WIDER FACE	984	5'644

Table 2.1: DATASET OVERVIEW. Number of images and annotations for each dataset

To sum up the overview of the different datasets used in this thesis, the number of images and ground truth annotations are listed in table 2.1. These datasets are again a topic in a later chapter when creating a data preparation pipeline to iterate through the data and apply different affine transformations.

2.2 Object & Face Detection

The detection of objects or faces in the image datasets presented above is performed, especially in recent years, by object/face detection frameworks based on Deep Learning. Modern object/face detectors primarily process the images in individual, interdependent, and sequentially combined process steps. Such detectors are called multi-stage detectors. Frameworks belonging to this category are R-CNN (Girshick et al., 2014), Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2015), Feature Pyramid Network (Lin et al., 2017a), HyperFace (Ranjan et al., 2017) and MTCNN (Zhang et al., 2016). On the other hand, frameworks aim to produce a comparable predictive outcome within a single stage. These are called single-stage detectors. Examples of this category are OverFeat (Sermanet et al., 2013), YOLO (Redmon et al., 2016) and RetinaNet (Lin et al., 2017b). These single- and multi-stage detectors are explained in more detail in the following.

2.2.1 Single-Stage Detectors

OverFeat is one of the first single-stage frameworks that perform simultaneous classification, localization, and detection based on a single convolutional network (Sermanet et al., 2013). Within this framework, features are extracted from a sliding window over a multi-scale pyramid of images. The region-wise features are then used for classification, localization, and detection. Hence,

it represents an integrated pipeline for different tasks by sharing common features. Additionally, part of that pipeline is an aggregating Non-Maximum Suppression method that leverages the localization task of the integrated detector.

A straightforward method of detecting objects in images is proposed by Redmon et al. (2016) with YOLO (You Only Look Once). YOLO consists of a single convolutional neural network that predicts bounding box coordinates and class probabilities from input images. Thereby, features from the whole image are used to predict multi-class bounding boxes simultaneously. The prediction is made by dividing the input image into a grid. Multiple-sized bounding boxes are predicted in each grid cell with a confidence score of generally containing an object. The grid cell containing the center of the object is responsible for its detection. The confidence score of an object is the product of the probability that this bounding box contains an object and the ratio of overlap between the predicted bounding box and the ground truth annotation from the training images. The grid cell also predicts conditional class probabilities of the object belonging to one of the targeted object classes, resulting in a class probability map with one set of class probabilities per grid cell. By utilizing this grid cell approach, the resulting bounding box proposals are spatially constrained, and in each image, only 98 bounding boxes can be produced. Nevertheless, for large objects or objects close to the border of multiple cells, Non-Maximum Suppression is applied to reduce the case of redundant bounding boxes.

Another single-stage object detector is presented in Lin et al. (2017b) with RetinaNet. It is a unified network consisting of two sub-networks built on top of the output of a Feature Pyramid Network based on a ResNet architecture as the backbone. This backbone produces multi-scale feature maps and is introduced later in the section of multi-stage detectors. By using these multi-scaled feature maps, multiple anchor boxes are added. The first sub-network uses the anchor boxes from the multi-scale feature maps to classify objects using convolutional layers. In parallel, the second sub-network is responsible to regresses bounding box coordinates. Additionally, RetinaNet addresses the problem of the significant class imbalance during the training procedure in such single-stage detectors. The class imbalance emerges when proposing regions. While most regions do not contain an object of desire (background class), only a few of these regions cover an object. Most of these background class proposals are considered as easy to determine their background class membership. This imbalance leads to inadequate training, with the majority of proposals not contributing to the learning capability of the framework. Furthermore, the amount of background class proposals can dominate the training with cross-entropy loss and lead to model degeneration. Lin et al. (2017b) propose the focal loss to tackle this imbalance problem. The focal loss is a dynamically scaled cross-entropy loss. During training, the dynamic component of the loss function decays easy negatives and emphasizes the hard negatives. The focal loss is employed in the classification sub-network. At the end of this single-stage framework, classic greedy Non-Maximum Suppression is applied to produce the final set of detections. With this proposed method, Lin et al. (2017b) achieve comparable results in terms of accuracy compared to modern multi-stage detectors. This kind of detector framework is discussed in the next section.

2.2.2 Multi-Stage Detectors

R-CNN stands for combining region proposals with convolutional neural networks (CNNs) (Girshick et al., 2014). This method bridges the gap between image classification and object detection by localizing objects with a deep network. In order to classify objects within regions, features of these regions are extracted and fed into the classification module. As a classifier for these feature-enriched regions, a Support Vector Machine (SVM) is used. The class-specific detections are then further processed by classic greedy Non-Maximum Suppression to eliminate redundant class-specific detections. Even though R-CNN outperforms the single-stage framework OverFeat, it has three significant drawbacks according to Girshick (2015). In order to train the multi-staged

architecture of R-CNN, each of these stages has to be trained sequentially. This training procedure is additionally time-consuming and expensive in storage space. Moreover, during test-time, the features are extracted from each region proposal in each image. This procedure results in slow overall object detection.

In order to circumvent the three mentioned drawbacks of R-CNN, Fast R-CNN is introduced. Fast R-CNN is also a convolutional network framework to detect objects based on images and the associated regional proposals within the images (Girshick, 2015). Compared to its predecessor R-CNN, with Fast R-CNN detection, quality and speed can be increased. Fast R-CNN requires an image and region proposals as input to a convolutional neural network. Multiple convolutional and maximum pooling layers process the images to produce a convolutional feature map. The region proposals are extracted from the feature map in a region of interest (RoI) pooling layer. The segmented region proposals are then fed into a sequence of fully connected layers to be regressed to bounding boxes and classified into class-specific probabilities. The training is conducted single-stage using a multi-task loss for class probability prediction and bounding box regression. The region proposals originate from a backbone CNN pre-trained on ImageNet data to localize regions of interest in images. Experiments with Fast R-CNN utilize the output of different backbones as the input to the classification stage. Parts of these backbones are also fine-tuned to increase the localization of regions of interest. The reusability further leverages the classification and regression ability combined in the Fast R-CNN object detector.

Faster R-CNN is a unified cascading network, consisting of a Regional Proposal Network (RPN) to generate proposals and a Fast R-CNN object detector to detect objects based on those proposals (Ren et al., 2015). Faster R-CNN follows a modular design with two main modules and continues on the foundation of the Fast R-CNN, introduced above. Although it is modularly structured, the object detection system is a unified network, which is trainable end-to-end. By detecting objects within images, Faster R-CNN's first module proposes class agnostic regions for the second module to classify objects within these regions. The main goal of Faster R-CNN is to share computation between region proposals and the Fast R-CNN object detection network in the form of weights in the convolutional layers. The RPN is responsible for turning an input image into class-agnostic regional proposals with the help of a fully convolutional network, which is also called the backbone of the Faster R-CNN (Ren et al., 2015). One possible backbone choice of Faster R-CNN is a ResNet-50 FPN that employs a Feature Pyramid Network (FPN) (Ren et al., 2015; He et al., 2016; Lin et al., 2017a). ResNet-50 is a residual network with 50 layers (He et al., 2016). It consists of a convolutional input layer (conv1), four convolutional blocks (conv2 - conv5), and one fully connected dense output layer. The four convolutional blocks are built up by repeating the structure of 3 stacked convolutional layers multiple times; for example, in conv2, the sequence of 3 stacked convolutional layers is repeated three times. The convolutional blocks are displayed in context of the ResNet-50 architecture in figure 2.3.

By proposing the ResNet-50 as a residual network, He et al. (2016) state the importance of the increased depth of convolutional networks in visual recognition tasks in contrast to the saturation and rapid degradation of accuracy of deep non-residual convolutional networks. Therefore, skip connections are introduced by adding an identity function connecting in- and output of a convolutional block to allow the training of deeper networks. The output of a backbone in the RPN can be either a single-scale feature map or multi-scaled feature maps. The RPN slides a small network in a sliding window approach over these feature maps to generate region proposals. The region proposals are then fed into a class-agnostic bounding box regression layer and a classification layer. The classification layer only distinguishes between objects and non-objects. These regions are then pruned by an application of the classic greedy Non-Maximum Suppression algorithm. The output of the Region Proposal Network of the Faster R-CNN is then utilized by integrating the Fast R-CNN object detector described above. The advantage of integrating the Fast R-CNN predictor into a unified network is to jointly train these two modules to leverage the

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.3: RESNET ARCHITECTURES. Visual comprehension of the ResNet. The ResNet-50 is marked with a red rectangle. The convolutional blocks are indicted in brackets *Source: (He et al., 2016)*

performance of Faster R-CNN. Moreover, since feature maps can already be extracted from the RPN, computation between these two modules can be shared.

Feature Pyramid Networks (FPN) can be applied as part of the backbone of the above-introduced approaches of RetinaNet, Fast R-CNN, and Faster R-CNN. FPN aims to produce multiple proportionally sized feature maps at different levels for a given input image with the help of fully convolutional layers (Lin et al., 2017a). A pyramid gets constructed by laterally merging a bottom-up and top-down pathway of intermediate feature maps of the backbone. A simplified example of such is depicted in figure 2.4. The bottom-up pathway includes feature maps from the end of the different stages from the forward computation of a given backbone convolutional network. For example, with the ResNet-50 FPN, the bottom-up pathway extracts feature maps with their respective scale at the end of each convolutional block (conv2 - conv5). The first convolutional block (conv1) is not considered due to the large spatial resolution and memory footprint. Convolutional blocks from early levels in the network (i.e., conv2) produce feature maps higher in spatial resolution but lower in semantic value. Lower semantic value because an input image has not yet gone through many convolutional layers with subsampling resulting from subsampling strides. However, deeper convolutional blocks (i.e., conv5) can detect more high-level features that yield higher semantic values but lower spatial resolution. Therefore, the top-down pathway starts at deeper convolutional blocks of the backbone to upsample feature maps to merge these features via lateral connections from the same spatial size of the bottom-up pathway with a kernel size of 1×1 and the number of channels uniformly reduced to 256. With this procedure, FPN produces four feature maps $\{P_2, P_3, P_4, P_5\}$ from the output of the convolutional blocks conv2, conv3, conv4 and conv5, which are denoted as $\{C_2, C_3, C_4, C_5\}$. Each merged map gets fed into a separate convolutional layer with kernel size 3×3 to generate the final feature maps to eliminate the aliasing effect as a byproduct of upsampling. Additionally, a fifth feature map P_6 can be created with an even bigger scale by subsampling P_5 with a stride of 2 in a maximum pooling layer to cover a bigger anchor scale. The resulting feature maps (pyramid) of FPN are, on the one hand, rich in high-level semantic value and, on the other hand, provided at different scales to enrich the generic feature extraction in visual recognition tasks.

Another multi-stage convolutional neural network framework is HyperFace, which simultaneously detects faces, landmark locations, pose estimation, and gender classification (Ranjan et al., 2017). HyperFace makes use of the AlexNet as the backbone. The authors also propose a variant of HyperFace, called HyperFace-ResNet, which is based on the ResNet-101 backbone. Generally, features resulting from selected intermediate layers of the backbone are fused. Lower level features are used for landmarks localization and pose estimation. Features from higher

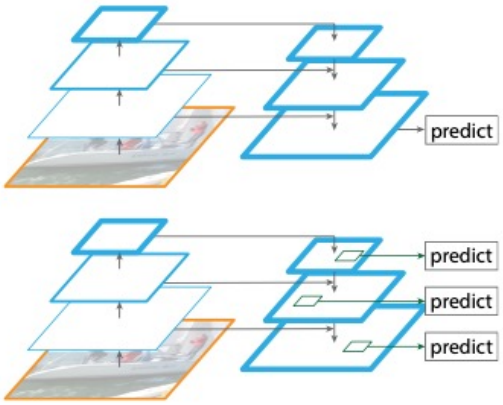


Figure 2.4: FEATURE PROPOSAL NETWORK. The difference between predicting on the first level of a feature pyramid versus using multiple levels of the top-down pathway *Source: (Lin et al., 2017a)*

layers of the backbone are used for detection and classification. Similar to Faster R-CNN, the HyperFace framework also proposes class-agnostic region proposals in its first stage. Regions of interest are proposed by initializing the weights of the backbone with weights from a version of R-CNN, trained explicitly for face detection. The second stage takes up the proposals and feeds the proposals into five different branches: detection, landmarks localization, visibility (to test the presence of predicted landmarks), pose estimation, and gender classification. Each of these branches consists of two fully connected layers to predict task-specific labels. In contrast to Faster R-CNN, HyperFace employs the third stage of iterative region proposals (IRP) and landmarks-based Non-Maximum Suppression. With IRP, new candidate boxes are proposed based on predicted landmark points if a face has not been detected due to illumination or small size. This step increases the ability to localize faces. Landmarks-based NMS performs NMS based on a new bounding box generated by landmark coordinates, which is expected to localize the candidate regions precisely. The landmarks localization ability of HyperFace is evaluated on the normalized mean distance between predicted coordinates and ground truth coordinates from the test data.

The multi-task cascaded CNNs based framework (MTCNN) is a unified framework with three stages for face classification, bounding box regression, and facial landmark localization (Zhang et al., 2016). This approach aims to combine the face detection method and the face alignment method using the correlation between these two methods. Moreover, the MTCNN integrates face detection and alignment with an online hard sample mining training method with a multi-task learning procedure. The face detection and alignment are leveraged by the construction of the framework with three stages in a multi-scale approach (Zhang et al., 2016). Similar to the RPN module in the Faster R-CNN framework, MTCNN also utilizes a multi-scale approach to detect faces in multiple sizes. However, MTCNN differs from FPN by building a multi-scale image pyramid consisting of the input image resized to different scales instead of collecting multi-scale feature maps from intermediate convolutional blocks of the backbone.[8] The first stage of MTCNN is a proposal network (P-Net) in the form of a fully convolutional network. The P-Net estimates candidates by regressing bounding boxes on the multi-scale input image and suppresses highly overlapping candidate boxes with Non-Maximum Suppression. The NMS function, thereby used, is a classic greedy approach. All remaining candidate boxes are passed to the second stage – the refine network (R-Net). In the second stage, CNN calibrates the candidates and applies NMS

[8] Facenet-PyTorch on Github: https://github.com/timesler/facenet-pytorch

again. The third stage is the output network (O-Net), which produces the output in bounding boxes and facial landmark positions. The O-Net is similar to the R-Net with the difference that bounding boxes and probability estimates are produced, and five facial landmark positions are detected within the bounding boxes. The training of the MTCNN is a multi-task learning process based on binary face classification, bounding box regression, and facial landmark localization. According to Zhang et al. (2016) each of the individual stages is trained on different data. P-Net is trained on patches from WIDER Face training to present the net's positive, negative, and partial faces. R-Net is trained on the positive, negative, and partial faces from the P-Net training process and on landmark faces from CelebA. In the last stage, O-Net is trained on data collected from the first two stages and compared to the positive, negative, and partial faces from WIDER Face training and the landmarks from CelebA.

2.3 Non-Maximum Suppression

Non-Maximum-Suppression (NMS) is considered an essential algorithm in the post-processing of the detection frameworks described sections 2.2.1 and 2.2.2 in terms of selecting the final detection result (Zou et al., 2019). Both single and multi-stage detection frameworks produce numerous neighboring proposals for the same object. NMS generally contributes to the final selection by reducing the number of overlapping bounding boxes with different techniques. The most common approach of performing NMS is the classic greedy selection. It is called greedy because the selection process is carried out in each spatial location, overlapping more than one bounding box. For all bounding boxes, which overlap with a certain threshold, only the one with the highest probability estimate of enclosing an object, is selected and the other ones are suppressed. In a multi-class setting, the bounding boxes of different classes are offset to not interfere with differently classified bounding boxes. The overlap for two bounding boxes B_1 and B_2 is calculated using the Intersection over Union (IoU), also known as the Jaccard index (Padilla et al., 2020; Jaccard, 1901):

$$IoU = J(B_1, B_2) = \frac{area(B_1 \cap B_2)}{area(B_1 \cup B_2)} \tag{2.1}$$

If B_1 and B_2 perfectly overlap, their IoU score is 1. The classic greedy NMS would select in this simple example of B_1 and B_2 with an IoU above a threshold, only with a higher detection score. The other bounding box gets suppressed by setting the detection score to 0. Hence, classic greedy NMS applies a discontinuous binary weighting function to suppress overlapping but non-maximally scoring bounding boxes (Bodla et al., 2017).

There have been several proposals to replace the prevailing method of classic greedy NMS in the past years. Nevertheless, many of the previously mentioned frameworks use this same approach in their post-processing steps to arrive at the final result of object and face detection. On closer examination, these recent developments can be clustered into two groups; post-processing NMS substitution and learning NMS.

2.3.1 Post Processing NMS Substitution

The group of post-processing Non-Maximum-Suppression techniques aims to directly replace the classic greedy approach by using another method for the selection process. These methods can be considered for substitution because they work similarly and can be implemented instead of classic greedy NMS. Therefore, these techniques benefit from a low cost of change in the detection pipeline. Such methods are Soft-NMS (Bodla et al., 2017), the OverFeat method of bounding box accumulation (Sermanet et al., 2013), Non-Maximum Weighting (Zhou et al., 2017; Ning et al.,

2017) and Weighted Boxes Fusion Solovyev et al. (2021). Additional NMS alternatives are described in the appendix A with ASAP-NMS (Tripathi et al., 2020) and Chaotic Whale-NMS (Wu and Li, 2021). These two methods are considered representatives of either increasing the speed of NMS or interpreting NMS as a combinatorial optimization problem. These methods are only indirectly related to the methods discussed in this section and not further specified throughout the thesis and therefore not detailed in this section.

In addition to the classic method, Soft-NMS focuses on not directly suppressing overlapping non-maximal detections. Instead of directly suppressing these detections, Soft-NMS reduces the scores of the bounding box as the overlap with the bounding box associated with the local maximum classification score increases (Bodla et al., 2017), which means that probability estimates of overlapping detections are not set to 0 but somewhat decayed. The decay is performed to reduce the likelihood of these neighboring bounding boxes being false positives. The higher the overlap between the maximally scoring detection and a neighboring bounding box, the higher the chance that the latter is a false positive detection. On the other hand, suppressing all neighboring detections would increase the false negatives drastically. In contrast to the discontinuous binary weighting function of the suppression in classic greedy NMS, Soft-NMS works as a greedy rescoring procedure with a continuous function. It penalizes neighboring detections with a high overlap more to reduce their classification score than neighboring bounding boxes with low overlap by considering non-maximally scoring detections with lower scores instead of suppressing them. Softer-NMS implemented in Faster R-CNN increases accuracy on the PASCAL VOC dataset of 1.7% and 1.1% on the COCO dataset. These results only consider the accuracy aspect without the consideration of the bounding box stability.

The single-stage framework OverFeat, introduced in section 2.2.1 by Sermanet et al. (2013), uses a method of accumulating predicted bounding boxes to increase localization and detection performance. More specifically, the method is a greedy merge strategy, which is applied to the regressed bounding boxes to combine a multitude of regressed bounding boxes into a final prediction. A match between boxes is made by iterating over all regressed bounding boxes if the sum of the distance between the box centers and their IoU score is above a certain threshold. All matched boxes then get merged by computing the average of the bounding box coordinates. The single merged bounding box gets the single maximal detection score of all matched boxes assigned. The experiments conducted to evaluate the entire network only indicate the accuracy of predicted bounding boxes without considering the stability.

With Convolutional network, Adapters and Detector (CAD), and Inception Single Shot Multibox Detector (I-SSD), two frameworks for fast and accurate object detection are proposed by utilizing Non-Maximum Weighting (NMW) (Zhou et al., 2017; Ning et al., 2017). NMW is the bounding box selection method that considers all non-maximum boxes' object information that overlaps the maximally scoring bounding box with a given threshold. All bounding boxes above this threshold contribute to one finally predicted weighted bounding box. Every bounding box gets related confidence, which is the product of its detection score and the IoU between the box and the maximally scoring bounding box. The final prediction consists of the sum of all overlapping bounding boxes multiplied by their related confidence and scaled by the sum of all related confidences. Concerning the further processing downstream, it is only stated that the combined bounding box is used. It is assumed that comparably to OverFeat, the corresponding detection score of the combined bounding box results from the maximum score of all considered detections. Aside from the original goal of proposing an efficient network, it was evaluated about the accuracy of its predictions without any focus on the stability of the newly proposed Non-Maximum Weighting method.

A similar combination of bounding boxes is proposed by Solovyev et al. (2021) and is based on the predictions of different object detection models by applying weighted boxes fusion (WBF). Different object detection models are considered as multiple frameworks or the same framework

predicting on differently augmented images. All predictions from different models are unified in a single list. Like classic greedy NMS, the maximally scoring detection and all bounding boxes with an overlap above a certain threshold are selected. The combination of the maximally scoring bounding box and all overlapping bounding boxes to the threshold are fused. This fusion results in an averaged classification score. This score is also considered for the weighted sum of the bounding boxes. The detection score is then re-scaled to reflect the inequality of the number of predictions made by the different models. With this method, the quality of combined bounding boxes is significantly increased. However, this improved quality is only measured in terms of accuracy. No statements regarding the size or scale of the newly produced bounding boxes in contrast to the ground truth or the baseline methods are made. Additionally, the primary purpose stated for this method is to combine the final predictions of multiple models in an ensemble. Therefore, the improved quality reported is also based on the combination of predictions of an ensemble of models. When WBF in a single model replaces classic greedy NMS, it even decreases the model performance.

2.3.2 Learning NMS

The second cluster of Non-Maximum-Suppression alternatives proposes a more integrated implementation of the selection process into the learning process of an object detector. The methods of this group cannot simply replace the classic greedy NMS in the post-processing but need to be considered during the end-to-end learning process of the whole network. As the prominent member of this group of methods, only Softer-NMS (He et al., 2018) is discussed in this section due to its further reference throughout this thesis. Further methods are shortly elaborated in the attachments, including Learning-to-Rank Tan et al. (2019), and Fitness NMS Tychsen-Smith and Petersson (2018).

He et al. (2018) combine object localization through embedding a bounding box regression loss function into the learning procedure of the detection framework and applying a weighted average during Non-Maximum Suppression. Softer-NMS is an improvement of the post-processing substitution method Soft-NMS described above. The localization confidence of Softer-NMS is achieved by regressing not only the bounding box coordinates but instead predicting a probability distribution of the bounding box location. It further improves Soft-NMS by advancing with high-scoring candidate bounding boxes, improving overall performance, and refining those selected bounding boxes with a weighted average based on confidence from the Gaussian distribution. The findings conclude that the classification confidence is not firmly related to localization confidence. The findings justify improving classic greedy NMS, which only relies on the classification confidence to select the final predictions. Even though the implementation of Softer-NMS focuses on the accurate localization of bounding boxes, there is no distinct evaluation of localization or stability compared to a benchmark. The principal metric to draw the final results of the publication is in line with the evaluation metrics used by the discussed methods mentioned above. Thereby, mAP is used to measure the performance of Softer-NMS compared to Soft-NMS and classic greedy NMS. However, the authors attribute the improvement in mAP to the more accurate localization of bounding boxes.

2.4 Evaluation Metrics

According to Zou et al. (2019), the increasing dissemination of the data sets mentioned at the beginning of this chapter leads in the same step to the dissemination of the evaluation metrics used therein. For example, average precision began to gain traction with the release of the PASCAL VOC challenge 2007. With the introduction of the MS-COCO challenge, researchers emphasized

the localization accuracy of the predicted bounding boxes. To this end, an adaption of the metric was proposed to be used in a complementary manner. These two metrics, driven by the PASCAL VOC and COCO challenges, were frequently reported by researchers using these datasets to train and validate their object detectors. For datasets other than these two, these metrics are also used to provide a comparison between different frameworks. In this section, the essential components of the above evaluation metrics are explained in detail. Later, an alternative perspective on evaluation will be introduced by considering the related field of evaluating video detection results.

2.4.1 Object Detection

The assessment of object detection accuracy is established by comparing the annotated ground truth objects of a given dataset with the predictions made by an object detection framework based on the respective image (Padilla et al., 2020). The ground truth annotations consist in many cases of human-labeled bounding boxes that enclose an object. Each of these bounding boxes possesses a label indicating the class membership of the respective enclosed object. An object detector aims to localize the objects present in an image with a detection bounding box close to the ground truth bounding box. In addition, the object detector outputs a probability for each of the class labels, including a background class. Each bounding box has four coordinates encoding the corners of a box, which closely surrounds the object. Three different scenarios are possible by detecting objects and comparing them to the ground truth: (I) An object detector detects an object in an input image with a certain probability. This object is also labeled as such in the ground truth annotation. If the probability estimate of the object detector surpasses a predefined threshold, the prediction is a true positive (FP). (II) If the object detector detects an object where according to the ground truth, no object is located, then it is a false positive (FP). (III) Suppose the object detector misses detecting an object where it is supposed to be, or the estimated probability of being such an object is too low. In that case, it is a false negative (FN) detection. There is also a fourth detection scenario, which is not relevant for object detection evaluation, a true negative (TN) detection. A true negative detection is when the object detector correctly does not detect a non-existing object of the ground truth annotations. TN is not relevant because there is an infinite number of possible true negative detections in an image. There are two main criteria to determine the correctness of object detection: first, to only look at bounding boxes with a high enough probability of enclosing a particular object (Padilla et al., 2020). Second, to compare the detected bounding box with the ground truth bounding box and measure their overlap (Padilla et al., 2020). To measure their overlap, the IoU score, according to equation (2.1) is calculated. With the notation of IoU, it is possible to calculate the overlap between a detected and ground truth bounding box. Then to determine with a threshold t whether the overlap is sufficient to account for detection for being true positive ($IoU \geq t$) or not sufficient and therefore being false positive ($IoU < t$) (Padilla et al., 2020). There also exist different types of false positives about duplicate detection errors if multiple detections overlap with the ground truth with an $IoU \geq t$ (Hoiem et al., 2012). However, this section does not cover distinguishing this kind of error but rather summarizes these errors as false positive detections.

By combining the three object detection scenarios mentioned earlier, it is possible to compute precision P and recall R, according to Padilla et al. (2020), as followed:

$$\text{Precision} = P = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all detections}} \tag{2.2}$$

$$\text{Recall} = R = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all ground truths}}. \tag{2.3}$$

The precision indicates how well the object detector can detect the correct objects among all detections (Padilla et al., 2020). On the other hand, recall measures the fraction of the correctly identified objects among all ground truth annotations. There is a trade-off between precision and recall, which is influenced by the confidence values of the detections (Padilla et al., 2020). By lowering the allowed confidence of the object detector to predict bounding boxes, it is likely that with a more significant number of predicted bounding boxes that more true positives and fewer false negatives can be achieved, and the recall will increase (Padilla et al., 2020). Simultaneously, by predicting more bounding boxes, it is also likely that there are more false-positive detections, which leads to a decrease in precision (Padilla et al., 2020).

The most commonly used metric to capture accuracy is the Average Precision (AP), which summarizes the precision and recall trade-off (Padilla et al., 2020). One popular way to calculate AP based on precision and recall is the 11-point interpolation (Padilla et al., 2020). This method obtains 11 equally spaced recall levels at $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$ and summarizes the maximum precision $P_{interp}(R)$ at each of these levels (Padilla et al., 2020). If there is no exact recall value equal to the level, then $P_{interp}(R)$ is taken from the next higher recall value R (Padilla et al., 2020). According to Padilla et al. 2020 and Everingham et al. (2010) the sum is then averaged over all 11 levels as in

$$\text{AP}_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots, 0.9, 1\}} P_{interp}(R), \tag{2.4}$$

with

$$P_{interp}(R) = \max_{\widetilde{R}:\widetilde{R} \geq R} P(\widetilde{R}). \tag{2.5}$$

The 11-point interpolation AP is known for being the base metric of the PASCAL VOC challenge, as well as for the WIDER Face benchmark (Everingham et al., 2010; Yang et al., 2016). In a multi-class dataset such as PASCAL VOC, it is also possible to utilize the 11-point interpolation about the different classes (Padilla et al., 2020). Therefore, mean Average Precision (mAP) is introduced, which averages the class-specific AP_i over all classes N (Padilla et al., 2020). The Mean Average Precision, as stated in Padilla et al. (2020), is defined as

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i. \tag{2.6}$$

Mean Average Precision and multiple variants of it are the base metric for the COCO challenge.[9] The main challenge metric for COCO however, differs from the approach described above. First, COCO evaluation does not differentiate between AP and mAP. Thereby, mean Average Precision over all classes is applied but stated as AP.[9] Moreover, the mAP is not measured at a single IoU score, which is often at 0.5 but rather averaged over ten different IoU thresholds.[9] These thresholds are taken from a range between 0.5 and 0.95 with a step size of 0.05.[9] Next to the primary challenge metric, the evaluation of predictions using the COCO dataset employs 11 more metrics.[9] The totally 12 evaluation metrics are divided into 4 groups of 3 metrics as displayed in table 2.2. The first group measures mean Average Precision by applying the aforementioned primary challenge metric over 10 IoU thresholds, over a single 0.5 IoU threshold, and a single 0.75 IoU threshold (Padilla et al., 2020). The second metric, $\text{AP}^{\text{IoU=0.5}}$ is the main metric of the PASCAL VOC challenge and therefore also for the WIDER Face benchmark (Everingham et al., 2010; Yang et al., 2016; Padilla et al., 2020).[9] The COCO challenge also differentiates between scales because the majority (41%) of objects in the COCO dataset are of small area ($< 32^2$), 34% are medium-sized

[9] COCO Validation: https://cocodataset.org/#detection-eval

(between 32^2 and 96^2), and 24% belong to the large category ($> 96^2$).[9] This distinction is made for both Average Precision and Average Recall. Average Recall is, according to Hosang et al. (2015) comparable to the COCO definition of Average Precision at ten different IoU levels between 0.5 and 1 and also refers to the mean Average Recall as AR across all classes (Padilla et al., 2020). Average Recall is also calculated by limiting the number of detections as displayed in table 2.2 in the third section. AR per fixed number of detections is the maximum Average Recall given this limitation and then averaged over all classes and IoU levels.[9]

COCO challenge metrics	
Metric Name	Metric Description
Average Precision (AP)	
AP	mAP at IoU = 0.5:0.05:0.95
$AP^{IoU=0.5}$	mAP at IoU = 0.5 (PASCAL VOC metric)
$AP^{IoU=0.75}$	mAP at IoU = 0.75
AP Across Scales	
AP^{small}	mAP for small objects with area < 32^2
AP^{medium}	mAP for small objects with area between 32^2 and 96^2
AP^{large}	mAP for small objects with area > 96^2
Average Recall (AR)	
$AR^{max=1}$	AR with 1 detection per image
$AR^{max=10}$	AR with 10 detections per image
$AR^{max=100}$	AR with 100 detections per image
AR Across Scales	
AR^{small}	AR for small objects with area < 32^2
AR^{medium}	AR for small objects with area between 32^2 and 96^2
AR^{large}	AR for small objects with area > 96^2

Table 2.2: COCO CHALLENGE METRICS. An Overview of the extensive COCO challenge metrics.[9]

2.4.2 Stability

The bounding box stability is a metic proposed to shift the evaluation paradigm in video detection (VID) and multi-object tracking (MOT) (Zhang and Wang, 2016). However, the predominant evaluation metrics for VID and MOT challenges and benchmarks are still based on precision, recall, and their summarization in mean Average Precision (Zhu et al., 2020; Sundararaman et al., 2021). Additionally, there is an imbalance of labeled data available for videos and images. Since an evaluation metric requires ground truth annotations, it is a limiting factor for VID and MOT. The main difference between detecting objects or faces in videos compared to images is the accumulation of these metrics over consecutive frames with highly correlated features instead of independent images (Zhu et al., 2020). Since these metrics are fundamentally based on the overlap between predicted and ground-truth bounding boxes, there can be multiple overlap scenarios resulting in the exact accuracy measurement. This means that predicted bounding boxes might jitter around the ground truth annotation, which might not be accounted for in the traditional accuracy metrics. Moreover, Zhang and Wang (2016) show that these accuracy metrics have a low correlation with the stability metric, which means that both metrics should be considered when evaluating object detection with moving objects.

The stability metric proposed by Zhang and Wang (2016) consists of 3 components. The first

component is the fragment error E_F, which measures the consistency of the detections along all trajectories of the targeted objects. The consistency indicates whether the object detector reports the exact status of an object over all the frames in a trajectory or if it changes the status frequently. The status of an object is the state of being detected or not. The fragment error is negligible when an object has never been detected or the detector consistently produces detections that align with the ground truth trajectory. On the other hand, the fragment error is large when the detector has alternating detections and non-detections regarding a targeted object. The fragment error of a video sequence is defined as

$$E_F = \frac{1}{N}\sum_{k=1}^{N}\frac{f_k}{t_k - 1}, \tag{2.7}$$

where N is the total number trajectories, t_k as the length of the k^{th} trajectory and f_k the number of changes between true positives and false negatives (Zhang and Wang, 2016).

The second component of the stability metric is the center position error E_C, which measures the stability of the center position of a bounding box along a trajectory with regard to the horizontal and vertical direction. The detected bounding boxes of the f^{th} frame and the k^{th} trajectory are considered to have the form $B_p^{k,f} = (x_p^{k,f}, y_p^{k,f}, w_p^{k,f}, h_p^{k,f})$, which correspond to the box center x and y coordinates, width and height. Analogously, the ground truth annotation has the form $B_g^{k,f} = (x_g^{k,f}, y_g^{k,f}, w_g^{k,f}, h_g^{k,f})$. By obtaining these values from a predicted and ground truth bounding box, Zhang and Wang (2016) define the center position error as stated in equation (2.8). E_C is the average of the summed horizontal and vertical standard deviations from the trajectories center positions over multiple frames.

$$\begin{aligned} e_x^{k,f} &= \frac{x_p^{k,f} - x_g^{k,f}}{w_g^{k,f}}, \quad \sigma_x^k = \text{std}(e_x^k), \\ e_y^{k,f} &= \frac{y_p^{k,f} - y_g^{k,f}}{h_g^{k,f}}, \quad \sigma_y^k = \text{std}(e_y^k), \\ E_C &= \frac{1}{N}\sum_{k=1}^{N}(\sigma_x^k + \sigma_y^k) \end{aligned} \tag{2.8}$$

The third component contributing to the stability metric measures the scale and aspect ratio of the bounding boxes along a trajectory. The bounding boxes shapes of the detections and ground truth are the same as stated above. According to equation (2.9) from Zhang and Wang (2016), E_R is the average of the summed scale and aspect ratio standard deviations of the trajectories. The scale errors are held in the same magnitude as the ratio errors by applying the square root; otherwise, this error term would proportionally contribute more to the overall scale and ratio error E_R.

$$\begin{aligned} e_s^{k,f} &= \sqrt{\frac{w_p^{k,f} h_p^{k,f}}{w_g^{k,f} h_g^{k,f}}}, \quad \sigma_s^k = \text{std}(e_s^k), \\ e_r^{k,f} &= \frac{w_p^{k,f} h_g^{k,f}}{h_p^{k,f} w_g^{k,f}} \quad \sigma_r^k = \text{std}(e_r^k), \end{aligned} \tag{2.9}$$

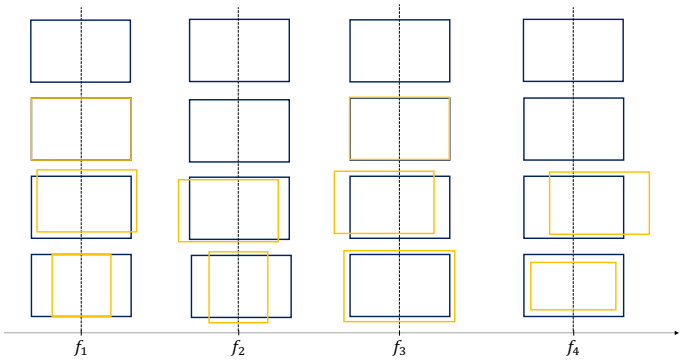


Figure 2.5: TRAJECTORIES OF SEQUENCE OF FRAMES. An example of four trajectories. The top row shows the ground truth annotation. The second row shows fragment errors, where f_1 is detected, f_2 is missing, f_3 is detected and f_4 is missing again in the trajectory. There are 3 status changes. This would result in the maximum fragment error of $3/(4-1) = 1$. The third row indicates deviations in the center position of the trajectory. The bottom row indicates deviations in scale and ratio of the trajectory. *Own illustration, based on (Zhang and Wang, 2016)*

$$\mathrm{E_R} = \frac{1}{N}\sum_{k=1}^{N}(\sigma_s^k + \sigma_r^k)$$

By having introduced all three components of the stability metric, these terms can then be added together to obtain the overall stability error Φ (equation (2.10)). However, some fundamental assumptions must be made before calculating Φ or one of its components. First, each detection in a frame has to be assigned to one trajectory. Since $\mathrm{E_C}$ and $\mathrm{E_R}$ measure the standard deviation of the errors on the horizontal and vertical axis for one trajectory over all frames, the objects must be matched between the frames. Second, the stability metric only considers detections matched to ground truth. Therefore, the stability metric is a complementary metric to the accuracy metric (Zhang and Wang, 2016).

$$\Phi = \mathrm{E_F} + \mathrm{E_C} + \mathrm{E_R} \tag{2.10}$$

The metrics described in this section are paramount to compare different object detectors concerning the accuracy and stability of their predictions. Even though different object detection challenges rely on different evaluation metrics, it is shown in this chapter that they all aim to measure the ability of an object detector to localize and classify objects in images. The PASCAL VOC challenge and WIDER Face benchmark both employ a shared measurement, which is also a subset of the extensive evaluation metrics of the COCO challenge. This lack of empirical evidence is mainly because the stability is measured over a sequence of frames, which is not the case for static images. On the other hand, the stability metric lacks practical application in object detection challenges. However, this metric enables another perspective on the ability of the detector to tightly fit bounding boxes with spatial stability compared to their associated ground truth boxes. Therefore, these metrics also play a crucial role in evaluating the performance of different models in multiple experimental setups with varying implementations of Non-Maximum-Suppression, which are elaborated in the next chapter.

Chapter 3

Stabilizing Bounding Boxes

The first research question of this thesis and, therefore, a primary goal of this thesis concerns the stabilization of bounding box predictions for face and object detection. The center of attention, therefore, lies on the post-processing step of Non-Maximum Suppression. The previous chapter on background & related work 2 indicates that the main focus of the literature is on either increasing accuracy or speed by substituting the classic greedy NMS. Nevertheless, the classic greedy approach is the method of choice for most face and object detectors. In this chapter, it is described how the general approach to performing NMS is challenged by altering the post-processing of two independent multi-stage detection frameworks – one for object detection with Faster R-CNN using the COCO and PASCAL VOC datasets and one for face detection with MTCNN using the WIDER Face dataset. During the forward pass in both frameworks, multiple process steps require a selection of bounding boxes. By substitution with a variable implementation of this particular bounding box selection, the behavior of the face and object detectors can be studied concerning accuracy and stability. Therefore, in this chapter, a variable implementation of bounding box selection is provided by a function that performs either classic greedy NMS or Soft-NMS of the previous chapter. Both methods are manually re-implemented based on the original functions of PyTorch and Soft-NMS and their algorithmic description. These two methods form the first part of this methodological chapter by creating a baseline benchmark of two established NMS approaches. In the second part, two approaches called Average NMS and Average IoU NMS are introduced to not only challenge the accuracy benchmark but, more importantly, in terms of stability. The bounding boxes created by Average and Average IoU NMS create new bounding box coordinates and probability scores based on all neighboring bounding boxes above a threshold. The last part of this chapter represents a thematically overarching topic of handling multi-class detections.

3.1 Classic Greedy Non-Maximum Suppression

The proposition of a more stable NMS approach requires modifying the selection process in the multi-stage detectors Faster R-CNN and MTCNN. The status quo of bounding box selection is to perform Non-Maximum Suppression in the post-processing step at the end of each stage. Thereby, bounding boxes, class memberships, and their associated predicted probability are sent together with an IoU threshold to an NMS function that performs classic greedy NMS (algorithm 1). As a result, the function returns a single list of indices, with which the original bounding boxes, scores, and labels are sliced to apply the actual selection of the NMS function. This slicing is a solid and rapid selection process since the same bounding boxes are already present and can be chosen by index. Algorithm 1 shows the formal procedure of the classic greedy Non-

Maximum Suppression as stated by Bodla et al. (2017). When comparing this algorithm to the observed practical implementation of PyTorch, two main differences can be observed.[1] First, algorithm 1 assumes to only receive bounding boxes of a single class and therefore does not need to distinguish between bounding boxes of different classes. This topic is discussed at the end of this chapter in section 3.5. Second, the return values are the selected boxes and the respective detection scores instead of the index.

Algorithm 1: Classic Greedy Non-Maximum Suppression

Input: $B = \{b_1, ..., b_N\}$, $S = \{s_1, ..., s_N\}$, N_t
B: list of initial detection bounding boxes
S: list of corresponding detection estimates
N_t: NMS threshold
begin
  D $\leftarrow$ { }
  while $B \neq empty$ **do**
    $m \leftarrow$ argmax S
    $M \leftarrow b_m$
    $D \leftarrow D \cup M; B \leftarrow B - M$
    for b_i *in* B **do**
      if $IoU(M, b_i) \geq N_t$ **then**
        $B \leftarrow B - b_i; S \leftarrow S - s_i$
      end
    end
  end
  return D, S
end

The algorithmic functionality of classic greedy NMS according to Bodla et al. (2017) and shown in algorithm 1 starts with detection bounding boxes, their associated detection probability estimates, and an NMS threshold. During a loop, the index of the detection score with the maximal value is stored in m. This index is then used to store the associated bounding box in M. The own practical implementation also follows the same structure with a loop. However, instead of looking for the index of the maximal score, the boxes and scores are already sorted in descending order of the score values – this way, the initial maximal score, and corresponding bounding box are selected by index 0. The next step is selecting the maximally scoring bounding box by adding it to the set D and taking it out of the set of all bounding boxes B. In the code, this is achieved by appending the maximally scoring bounding box and its score to two separate lists defined outside of the loop. The set difference is made by setting the bounding boxes and scores to a slice of these lists from index 1 to the end of the list. This way, the maximal score and associated bounding box of index 0 are disregarded. The algorithm loops over the remaining bounding boxes and compares the IoU to the maximally scoring bounding box M. The own practical implementation does not require a loop, therefore, but rather works with a two-dimensional tensor of the output of the Torchvision implementation of IoU calculation by calling `torchvision.ops.box_iou()` with M and the remaining boxes B as parameters.[1] With the pairwise IoU scores between M and $b_1, b_2, ..., b_n$ in the second dimension, these scores can directly be compared to the parametrized IoU threshold. The comparison is made with `torch.where`, which compares the second dimen-

[1] `Torchvision Ops:` https://pytorch.org/vision/stable/ops.html

sion to the condition of being smaller than the IoU threshold.[2] While the algorithm performs a set difference on all bounding boxes and scores with those scores greater or equal to the IoU threshold, the practical implementation directly selects only the boxes and scores with a score below the threshold and continues the loop with only those selected. According to Bodla et al. (2017) the pruning step in algorithm 1 can be written as a re-scoring function (equation (3.1)) with the scores set to 0 for an IoU score greater or equal to the threshold.

$$s_i = \begin{cases} s_i, & \text{IoU}(M, b_i) < N_t \\ 0, & \text{IoU}(M, b_i) \geq N_t \end{cases} \tag{3.1}$$

This re-scoring function can be considered a discontinuous binary weighting function with a hard threshold at N_t.

As soon as the set of bounding boxes B is empty, the sets of selected boxes D and associated scores S are returned while all other boxes and scores are suppressed. The own implementation also returns the actual bounding boxes and scores and the class labels, which also undergo the same slicing procedures during the loop as the boxes and scores. Even though the class labels are discussed later, it can be stated that the classic greedy NMS is class agnostic, and the labels are included in the own implementation only to match the length of associated bounding boxes and scores for simplicity purposes.

3.2 Soft-Non-Maximum Suppression

Algorithm 2: Soft-Non-Maximum Suppression

Input: $B = \{b_1, ..., b_N\}$, $S = \{s_1, ..., s_N\}$
B: list of initial detection bounding boxes
S: list of corresponding detection estimates
begin
  D $\leftarrow$ { }
  while $B \neq empty$ **do**
    $m \leftarrow$ argmax S
    $M \leftarrow b_m$
    $D \leftarrow D \cup M; B \leftarrow B - M$
    for b_i *in* B **do**
      $s_i \leftarrow s_i f(IoU(M, b_i))$
    end
  end
  return D, S
end

The authors of the Soft-NMS approach state that their method improves the accuracy of classic greedy NMS with one line of code (Bodla et al., 2017). From an algorithmic perspective, this statement can be confirmed when algorithm 2 is set against algorithm 1. Instead of comparing the IoU scores between the maximally scoring bounding box and all remaining bounding boxes with a hard threshold, a penalty function is applied to all remaining detections and the respective

[2] TORCH.WHERE: https://pytorch.org/docs/stable/generated/torch.where.html

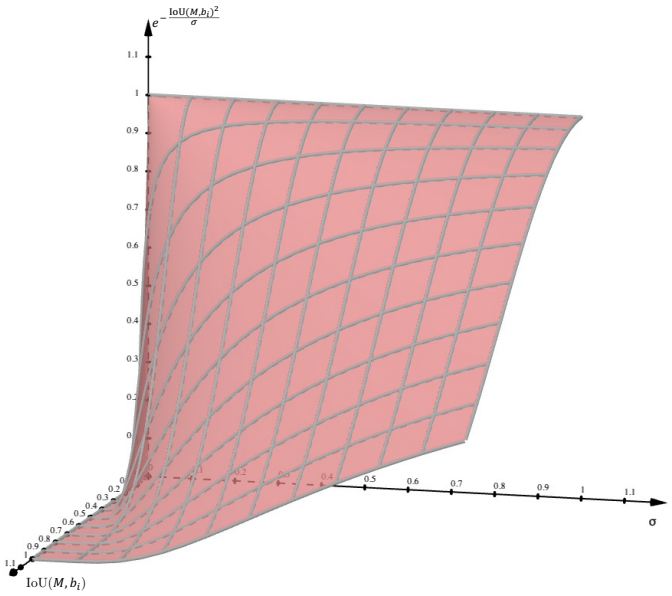


Figure 3.1: SOFT-NMS PENALTY FUNCTION. The Soft-NMS penalty surface as a function of σ and IoU(M, b_i). The ranges of IoU(M, b_i) and σ are $[0, 1]$ and $(0, 1]$ respectively.

scores. In terms of the rescoring function seen with classic greedy NMS in equation (3.1), the penalty function for Soft-NMS can be formulated as follows:

$$s_i = s_i e^{-\frac{\text{IoU}(M,b_i)^2}{\sigma}}, \forall b_i \notin D. \tag{3.2}$$

This function is considered a Gaussian function to prune the overlapping bounding boxes by penalizing their associated probability scores. The hyperparameter σ can be interpreted as similar to the threshold N_t of classic greedy NMS. The smaller σ is chosen to be, the lower is the penalty for the score of marginally overlapping bounding boxes. At the same time holding σ small, the more the bounding boxes overlap, the larger the penalty applied to the corresponding scores. The behavior of the penalty function can be observed on the surface of the graph in figure 3.1. In case M and b_i do not overlap at all, the exponent of e becomes 0, which leads to leaving the score s_i at its initial value since $e^{-0} = 1$. This is exactly the case in the classic greedy NMS re-scoring function if the IoU scores are below the threshold.

Even though it is slightly different from the algorithmic perspective between Soft-NMS and classic greedy NMS, the implementation differs in three significant points. First, from algorithm 2 it is not clear whether the penalized detection scores are only used for the selection during NMS or also assigned to the bounding boxes for further processing. Second, since an arbitrary number of detection scores are updated in every iteration, it is necessary to perform a reordering of bounding boxes based on the new scores afterward. The implementation assumes that the newly computed scores are also returned together with the bounding boxes due to Soft-NMS. Thereby, it can be continued to select the maximally scoring bounding box and its score at index 0.

Moreover, finally, there is also no threshold defined in algorithm 2 to limit the number of bounding boxes to perform a selection on the bounding boxes. Without actually pruning the bounding boxes and associated scores, the same number of detections would be returned from a Soft-NMS application. This would suggest that the detection framework applies an additional

comparison with a threshold by itself after Soft-NMS. Since this can not be assumed to be the case for all detection frameworks, the practical re-implementation of Soft-NMS features a comparison with a low threshold, which is variably chosen to be between 0.1 and 0.01. This means that only bounding boxes with associated re-scored detection probabilities above this score threshold are returned. Additionally, the hyperparameter σ of the Soft-NMS penalty function is chosen to be 0.5 as recommended by Bodla et al. (2017).

3.3 Average Non-Maximum Suppression

As seen in section 2.3, there exist multiple approaches to aggregate proposed bounding boxes to form a final detection that uses the information of neighboring boxes (Sermanet et al., 2013; Zhou et al., 2017; Ning et al., 2017; Solovyev et al., 2021). The first of the two newly introduced methods – Average NMS, is firmly based on the theoretical foundation of Solovyev et al. (2021) and Sermanet et al. (2013). Average NMS unifies the concepts of these two approaches by adopting a similar weighting and re-scoring procedure as seen in Solovyev et al. (2021) and inserting the selection process into the prediction pipeline as stated in Sermanet et al. (2013). On the one hand, the main difference of Average NMS and WBF proposed by Solovyev et al. (2021) lies in the usage of the predicted bounding boxes inside of the multi-stage frameworks Faster R-CNN and MTCNN. WBF is applied to combine predicted bounding boxes of an ensemble of different models in a post-processing fashion. On the other hand, Average NMS differs from the OverFeat bounding box merge procedure in terms of the match criteria between neighboring bounding boxes and the calculation of the final detection score.

By having a more detailed look at the WBF algorithm proposed by Solovyev et al. (2021), it can be observed that the first part of the selection of neighboring bounding boxes works in the same way as it is done in algorithm 1 with classic greedy NMS and in algorithm 2 with Soft-NMS. All neighboring bounding boxes with an IoU score above a given threshold are considered, together with their associated scores, as sets for candidate boxes CB and candidate scores CS. Both sets have the same number of elements c. First, the new score NS gets calculated as the average of all candidate scores CS with

$$NS = \frac{\sum_{i=1}^{c} CS_i}{c}. \tag{3.3}$$

As a second step, the new bounding box NB gets calculated as the weighted average by using the detection scores CS as weights with

$$NB_{WBF} = \frac{\sum_{i=1}^{c} CS_i * CB_i}{\sum_{i=1}^{c} CS_i}. \tag{3.4}$$

The newly proposed Average NMS uses weighted average bounding box aggregation calculation with an additional intermediate step to calculate the weights separately.[3] But in contrast to the calculation in equation (3.3), Average NMS uses these weights also for the re-calculation of the final detection score. Meaning that the final detection score is also a weighted average of all candidate scores instead of the average. Therefore, the weights w get calculated with

$$w_{ANMS,i} = \frac{CS_i}{\sum_{j=1}^{c} CS_j}. \tag{3.5}$$

These weights represent the share of each candidate score compared to all the candidate scores. Suppose an instance i of the candidate boxes CB is predicted with a high candidate score CS_j

[3] Bob IP Facedetect on Gitlab: https://gitlab.idiap.ch/bob/bob.ip.facedetect/-/blob/master/bob/ip/facedetect/detect.py#L15

with $i = j$. In that case, the weight is proportionally more extensive than for a candidate box with a lower candidate score. The sum of all individual weights equals 1. Then these weights are used to calculate the new aggregated bounding box coordinates with

$$NB_{ANMS} = \sum_{i=1}^{c} w_{ANMS,i} * CB_i. \tag{3.6}$$

By plugging equation (3.10) into equation (3.7), it can be shown that the Average NMS box aggregation is the mathematical equivalent to the WBF method:[4]

$$NB_{ANMS} = \sum_{i=1}^{c} w_{ANMS,i} * CB_i = \sum_{i=1}^{c} \frac{CS_i}{\sum_{j=1}^{c} CS_j} * CB_i = \frac{\sum_{i=1}^{c} CS_i * CB_i}{\sum_{j=1}^{c} CS_j} = \frac{\sum_{i=1}^{c} CS_i * CB_i}{\sum_{i=1}^{c} CS_i} \tag{3.7}$$

However, there is a significant difference of Average NMS to the WBF method in the calculation of the final detection score NS_{ANMS}. The new score NS_{ANMS} is calculated similarly as the new box NB_{ANMS} with

$$NS_{ANMS} = \sum_{i=1}^{c} w_{ANMS,i} * CS_i. \tag{3.8}$$

The reason for using the weighted average for the final detection score NS_{ANMS} is to attribute the proportional impact of high individual confidence to the final detection score. Meaning that if a bounding box obtains high confidence, it should contribute more to the overall weighted average detection score. Hence, the influence the detections have on the aggregation of the final bounding box is the same as for the formation of the final detection score. Therefore, the result of the Average NMS is more consistent between bounding box aggregation and confidence score calculation than WBF.

Following the mathematical formulations of Average NMS, it is shown that the proposed method uses the same weighted average for the bounding box aggregation and the detection score calculation. The procedure to apply these calculations within the bounding box selection is formally stated in algorithm 3. The main differences to the classic greedy NMS are the additional variables used to store candidate boxes and scores. Furthermore, an additional loop is required to compute the weights. By using these weights, the weighted average of the combined box and score is calculated. Thereby, `box_average` employs equation (3.7) to calculate the new boxes NB. At the end of each iteration, the combined box and score are collected by D and S, respectively. D and S are returned after the final iteration as the result of the Average NMS.

3.4 Average IoU Non-Maximum Suppression

The second proposed method is Average IoU NMS, which differs from Average NMS only in calculating the weights. As mentioned in the section above, when calculating the aggregated bounding box of Average NMS, the weighted average is used as stated in Solovyev et al. (2021) for the WBF method. This method was inspired by the calculation of Non-Maximum Weighted (NMW) from Zhou et al. (2017) and Ning et al. (2017). However, NMW also includes the IoU score between the maximally scoring bounding box M and the overlapping bounding box B_i to compute the weight w_i. As already stated in section 2, NMW only uses the IoU to compute the

[4] Weighted Arithmetic Mean Definition: https://en.wikipedia.org/wiki/Weighted_arithmetic_mean

Algorithm 3: Average Non-Maximum Suppression

Input: $B = \{b_1, ..., b_N\}$, $S = \{s_1, ..., s_N\}$, N_t
B: list of initial detection bounding boxes
S: list of corresponding detection estimates
N_t: NMS threshold
begin
  D $\leftarrow$ { }
  while $B \neq empty$ **do**
    $m \leftarrow$ argmax S
    $M \leftarrow b_m$
    $CS \leftarrow s_m$
    $CB \leftarrow M$
    $B \leftarrow B - M$
    for b_i *in* B **do**
      if $IoU(M, b_i) \geq N_t$ **then**
        $CB \leftarrow CB \cup b_i$
        $CS \leftarrow CS \cup s_i$
        $B \leftarrow B - b_i; S \leftarrow S - s_i$
      end
    end
    $weights \leftarrow$ { }
    for cs_i *in* CS **do**
      $weights \leftarrow weights \cup cs_i /$ sum(CS)
    end
    $D \leftarrow D \cup$ box_average($CB, weights$)
    $S \leftarrow S \cup$ sum($CS * weights$)
  end
  return D, S
end

weights for the bounding box aggregation and then assigns the maximal detection score S_m to the aggregated box. Average IoU NMS extends, therefore NMW using the IoU influenced weights to calculate the weighted average of detection scores. Therefore, the weights w_{ANMS} are calculated in the same fashion as in NMW by

$$w_{AINMS,i} = \frac{CS_i * \text{IoU}(M, CB_i)}{\sum_{j=1}^{c} CS_j * \text{IoU}(M, CB_j)}. \tag{3.9}$$

These weights are then used to aggregate the bounding boxes equivalently as in equation (3.7). This procedure is exactly the same as in Zhou et al. (2017) and Ning et al. (2017). Then Average IoU differs from NMW by using

$$NS_{AINMS} = \sum_{i=1}^{c} w_{AINMS,i} * CS_i. \tag{3.10}$$

to calculate the final detection score NS_{AINMS}. This calculation is similar to the detection score calculation of YOLO (Redmon et al., 2016). The interpretation of the weights follows the same nature as with Average NMS with the difference that the product of detection score and IoU score determines the influence on the final aggregation. Neighboring boxes with a high overlap and high confidence contribute proportionally more to creating the final aggregation than boxes

with low overlap and low confidence. This contribution is also reflected in the Average NMS in the aggregation of the final bounding box and the final detection score. The formal application of these mathematical equations is stated in algorithm 4. It only diverges from the structure described in Average NMS in the calculation of candidate scores CS.

Algorithm 4: Average IoU Non-Maximum Suppression

Input: $B = \{b_1, ..., b_N\}$, $S = \{s_1, ..., s_N\}$, N_t
B: list of initial detection bounding boxes
S: list of corresponding detection estimates
N_t: NMS threshold
begin
  $D \leftarrow \{\}$
  while $B \neq empty$ **do**
    $m \leftarrow$ argmax S
    $M \leftarrow b_m$
    $CS \leftarrow s_m$
    $CB \leftarrow M$
    $B \leftarrow B - M$
    for b_i *in* B **do**
      if $IoU(M, b_i) \geq N_t$ **then**
        $CB \leftarrow CB \cup b_i$
        $CS \leftarrow CS \cup s_i *IoU(M, b_i)$
        $B \leftarrow B - b_i; S \leftarrow S - s_i$
      end
    end
    $weights \leftarrow \{\}$
    for cs_i *in* CS **do**
      $weights \leftarrow weights \cup cs_i /$ sum(CS)
    end
    $D \leftarrow D \cup$ box_average$(CB, weights)$
    $S \leftarrow S \cup$ sum$(CS * weights)$
  end
  return D, S
end

3.5 Multi-Non-Maximum Suppression

The above sections discuss four distinct NMS methods to implement experiments into Faster R-CNN and MTCNN. However, these multi-stage detection frameworks already employ the classic greedy NMS method in the post-processing of each stage. Since all of the above-stated NMS methods are considered as members of the post processing NMS substitutional methods from section 2, it is possible to replace the original implementation with a variable Multi-Non-Maximum Suppression function. This function allows a parametrized control over the applied method, thresholds, and multi-class distinction to evaluate different versions of the frameworks during the experimental part of this thesis. Furthermore, by implementing a holistic Multi-NMS function, the differing output, especially of classic greedy NMS and Soft-NMS compared to Average and

Average IoU NMS, can be unified in that regard. The implementation of Multi-Non-Maximum Suppression is stated in the appendix B.1.

The substitution of the original classic greedy NMS function call in both stages of Faster R-CNN and the three stages in MTCNN is implemented by calling the `mnms()` function instead. In the original implementation, both Faster R-CNN and MTCNN call the same NMS function by Torchvision.[5] Within the `mnms()` function, all 4 above mentioned methods are implemented according to the algorithms 1, 2, 3 and 4. Additionally, the function call from the original version of Faster R-CNN and MTCNN is added and set as the default method. The difference between the original function call and the re-implemented classic greedy NMS according to 1 is that the original function is written in C++. In contrast, the re-implemented version is written in Python. The method of choice for the experimental purpose can be passed as a parameter with the IoU threshold, Soft-NMS specific score threshold, a limiting parameter, and the multi-class distinction.

So far, all of the explained Non-Maximum Suppression methods are class agnostic, meaning that no class differentiation is made during the selection or aggregation of the final bounding boxes. This agnostic is due to the design of the original classic greedy NMS implementation by Torchvision, which is also applied to Soft-NMS, Average NMS, and Average IoU NMS.[5] Before performing any of the above-stated NMS algorithms, the bounding boxes are offset depending on the class/level they belong to. It is referred to classes or levels because NMS is, for example, in Faster R-CNN called in the Region Proposal Network, where no classes have been determined for the boxes yet. At this stage, the boxes can only be distinguished by the levels of the Feature Pyramid Network they originate from. The class/level value then gets multiplied with the sum of maximum coordinate value and 1. This multiplication means that the class/level 1 gets an offset of the maximum coordinate plus 1. These offsets are then added to the corresponding bounding box coordinates. This way, boxes of different classes/levels do not overlap and are considered separately during NMS. After the selection or aggregation of NMS, the offset is subtracted again.

In this chapter, four Non-Maximum Suppression methods are either re-implemented based on their algorithms or newly implemented with novel components based on existing literature. The re-implementation of classic greedy NMS and Soft-NMS aims to create a benchmark and draw comparisons between newly implemented aggregation methods. Average NMS and Average IoU NMS are both based on methods observed in the literature but partially altered to develop novel forms of weighted average bounding box aggregation. Especially the last two methods are designed to achieve the overall goal of this thesis to stabilize bounding box predictions. The last section of this chapter takes the four NMS methods and combines them in the Multi-Non-Maximum Suppression function. This function replaces the original function calls in Faster R-CNN and MTCNN to test the effects of the individual NMS methods during the experimental part of this thesis.

[5] Source Code for TORCHVISION.OPS.BOXES: https://pytorch.org/vision/stable/_modules/torchvision/ops/boxes.html#batched_nms

Chapter 4

Measuring Stability

The literature review in chapter 2 shows that there are proposed Non-Maximum Suppression methods that potentially contribute to the increase of stability of bounding boxes without specifically evaluating it. The prevailing metric of evaluating face and object localization and classification in images is the mean average precision either over multiple IoU thresholds as seen in Lin et al. (2014) or over a single IoU threshold as stated in Everingham et al. (2010) and lined out in table 2.2. The argumentation of a detector's ability to localize the objects better is made by attributing it to an increased accuracy as seen in He et al. (2018). While at the same time, there exists an evaluation metric for stability, primarily used for video detection and object tracking, proposed by Zhang and Wang (2016). The usage of this stability metric is not linked to the evaluation in images since it depends on a flow of frames to follow an object along a trajectory to capture the changes in bounding box coordinates and detection statuses. Therefore, a method is proposed to connect single images and detectors with this metric to establish an image stability evaluation. This establishment can validate the contribution toward stable bounding box prediction using the introduced NMS methods of the previous chapter.Consequently, this chapter contributes to answering of the second research question.

A video consists of a sequence of consecutive frames, which are also called still images.[1] The datasets introduced in section 2.1 provide combined 10'936 still images. However, these images are independent captures of objects or faces depending on the theme of the dataset, and they can hardly be sequentially combined for stability analysis. As mentioned in the introduction of this chapter, the stability metric requires the same object along a trajectory of at least two frames. Hence, there is a possibility to emulate multiple frames by creating multiple augmented versions of the same image and sequentially combining these versions as frames. When additionally the ground truth annotations of these images are augmented in the same way, the prerequisites for a stability evaluation with the introduced metric are given.

4.1 Sequence Construction

For constructing a short sequence of augmented images, it is decided to work with four frames. The reason for at least four frames is, on the one hand, to cover the original image and three different transformations in order to introduce a movement of the objects between the frame and to observe the detectors' behavior to localize them. On the other hand, it is limited to four frames due to the time constraint of this thesis. The first frame f_1 is the original image from the test/evaluation set of the respective dataset. The second frame f_2 represents a horizontal shift of the images to the right. The third frame f_3 is a vertical shift of the images downwards. The

[1] Definition of Frame Rate: https://en.wikipedia.org/wiki/Frame_rate

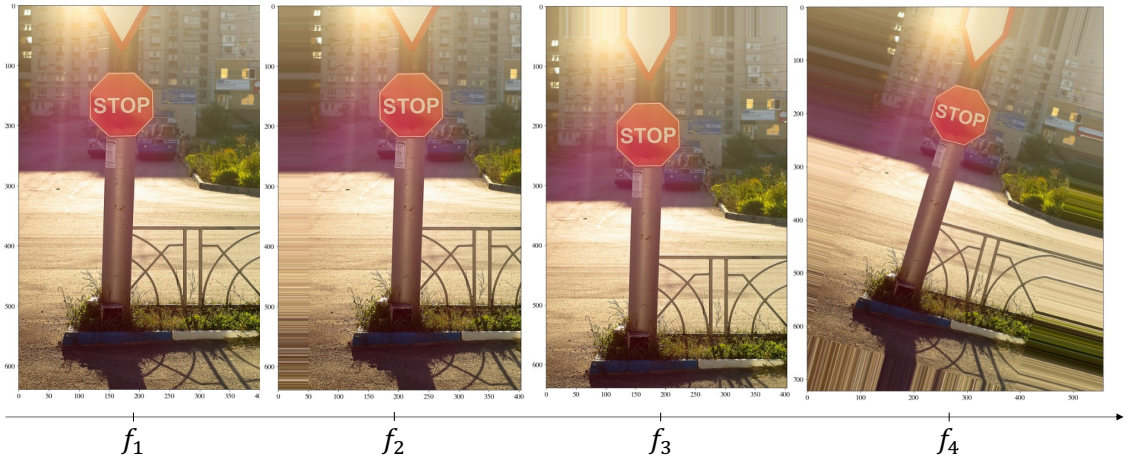


Figure 4.1: FRAME CONSTRUCTION. An example of a sequence of four augmented images as frames. With f_1 as the original image, f_2 with a shift of 50 pixels to the right, f_3 with a shift of 50 pixels downwards and f_4 with a rotation of 15 degrees.

fourth frame f_4 is a rotation of the images. The reason for the shifts on both axes in f_2 and f_3 is to obtain the same bounding box aspect ratio of the ground truth. Since the faces and objects in the still image are moved horizontally and vertically, the aspect ratio of the ground truth does not change. The rotation is to introduce a possible change in the aspect ratio of the ground truth bounding boxes. These three transformations are depicted in figure 4.1 and belong to the affine transformations. Additionally, the figure 4.1 shows the construction of the sequence of augmented images as frames $\{f_1, f_2, f_3, f_4\}$ with an example image and with arbitrary values for the shifts and rotation. All of these transformations are affine transformations and are formally described in the following section.

4.2 Affine Transformations

Formally, an affine transformation is a spatial coordinate transformation to relocated points in an image by translation, rotation, scaling, or sheering. According to Gonzalez et al. (2002), all mentioned transformations can be achieved by multiplying the coordinates with a 3×3 transformation matrix $\mathbf{T}$ as in equation (4.1). The kind of transformation is dependent on the values used for $t_{11}, t_{12}, ..., t_{32}$ in equation (4.1). The original image in f_1 can also be considered a transformation by having $\mathbf{T}$ as the identity matrix $\mathbf{I_3}$. Since this transformation is not specifically applied, it will not be stated as a transformation below.

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} v & w & 1 \end{bmatrix} \mathbf{T} = \begin{bmatrix} v & w & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix} \tag{4.1}$$

The transformations for the frames f_2 and f_3 are a translation in the positive direction of the x-axis and the positive direction of the y axis, respectively. Each transformation simulates a lateral and vertical movement of the objects in an image. These transformations can be achieved by adjusting the transformation matrix $\mathbf{T}$ and choosing the values for t_x and t_y accordingly. The

transformation matrix $\mathbf{T_t}$ for translations has the following form:

$$\mathbf{T_t} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \tag{4.2}$$

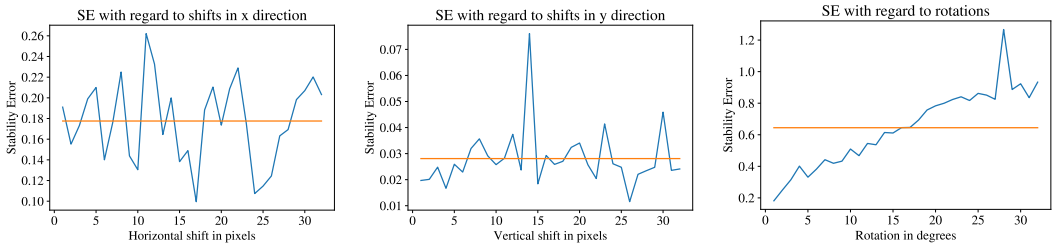
The transformation for the fourth frame f_4 is a rotation of the original image. Therefore, the transformation matrix $\mathbf{T}$ is adjusted in the following way to obtain the transformation matrix for the rotation $\mathbf{T_R}$:

$$\mathbf{T_R} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

In order to find values for t_x, t_y, and θ, an exploratory search is conducted in both frameworks Faster R-CNN and MTCNN separately by employing the stability metric to multiple possible transformations. Therefore, images from the COCO and WIDER Face dataset have been randomly selected, and the detections evaluated for a range of possible values for t_x, t_y, and θ. The evaluation works similarly as described in section 4.1 with a sequence of two frames. The first frame is the base case with the original image of the datasets. The second frame is transformed concerning one of the parameters t_x, t_y, and θ while holding the other two transformations at 0. Specifically, one of the three parameters is chosen to then repeatedly be compared to the base case with a parameter value of the set $P = \{1, 2, 3, ..., 32\}$. This means in the example of the horizontal shifts; the stability is evaluated between the base image and, for example, a shift by 1 pixel, 2 pixels, up to the comparison of the base case and a shift of 32 pixels. They result in 32 comparisons between the base image and a transformed version of it for each evaluation scenario. Hence, there are three evaluation scenarios for Faster R-CNN with images of the COCO dataset and three evaluation scenarios for MTCNN with images of the WIDER Face datasets. The transformation values chosen with the exploratory search with images of the COCO dataset are assumed to have the same effect on PASCAL VOC since both datasets use the same detector, and these values are rather framework-specific than dataset-specific. The goal of this exploratory search is to find values for t_x, t_y and θ, which will determine the augmentation in the construction of the sequence in section 4.1. These transformation parameters should have the same values for Faster R-CNN and MTCNN to enable comparison between these two frameworks in terms of stability depending on the sequence of frames.

The exploratory search for possible values for the transformation parameters t_x, t_y and θ for Faster R-CNN is summarized in figures 4.2 and 4.3. The stability errors in figure 4.2 are the respective sums of fragment error, center position error, and scale and ratio error for Faster R-CNN concerning the affine transformation. Figure 4.3 depicts the same errors for MTCNN. The individual errors for each of the stability error breakdowns are shown in the appendix C.2. It can be observed by looking at both figures that the stability errors for horizontal and vertical translations changes from high to low errors frequently and repeatedly. A periodic pattern could be observed if the same graph plotted for an individual image with only one object located within it. The periodicity comes from the sliding window and maximum pooling of the multi-scale feature maps in the Faster R-CNN and MTCNN. Whenever the horizontal or vertical translation of the original sized image overlaps with the downscaled location of the sliding window, very similar features are observed as in the non-translated base image. Thereby, the stability error is low. On the other hand, if the translation falls into another sliding window location, the observed features might vary more, and the stability error is larger. In the case of the individual image, these stability highs and lows behave periodically.

The periodicity depends on the proposal's origin in the multi-scale feature pyramid in the case of Faster R-CNN and the multi-scale image pyramid in the case of MTCNN. By combining multiple images in the exploratory search, their bounding boxes are detected with different origins



(a) Faster R-CNN stability error for $t_x \in P$

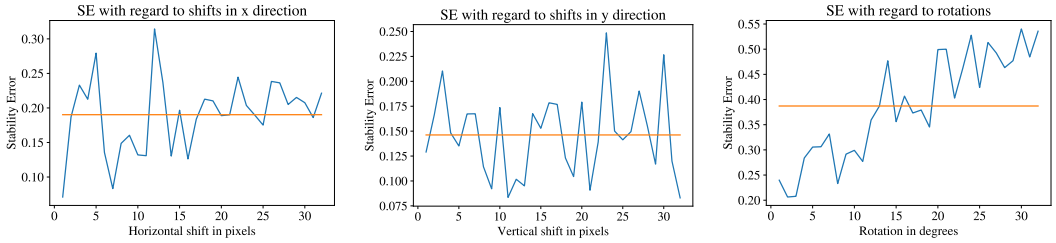
(b) Faster R-CNN stability error for $t_y \in P$

(c) Faster R-CNN stability error for $\theta \in P$

Figure 4.2: FASTER R-CNN STABILITY ERROR FOR AFFINE TRANSFORMATIONS. The stability error curves for each of the three affine transformations with Faster R-CNN. (a) shows the stability error (y-axis) over an increase of horizontal translations (x-axis) in pixels. (b) shows the stability error (y-axis) over an increase of vertical translations (x-axis) in pixels. (c) shows the stability error (y-axis) over an increase of rotations (x-axis) in degrees. The horizontal orange line in all three graphs indicates the mean standard error for the respective curve.

in the feature/image pyramid depending on their size. Therefore, by adding them together, the individual patterns get mixed. When the parameters t_x, t_y, and θ for Faster R-CNN are chosen, the stability error is low; the overall stability evaluation would indicate high stability. However, these parameters would have a different effect on the MTCNN framework since the proposals are generated differently. This would be the case when choosing $t_x = 17$ for example. The same logic applies to parameters where the stability error is high in one of the frameworks in the exploratory search. Therefore, the parameters are chosen around the average level (orange line) of the exploratory search stability error to ensure these levels are comparable between Faster R-CNN and MTCN. When comparing figure 4.2a and 4.3a with the aforementioned objective, a shift by 20 pixels is a valid value for t_x. The horizontal translation by 20 pixels resulted in both Faster R-CNN and MTCNN in a stability error close to the mean, which is comparable between the frameworks. By checking 4.2b against 4.3b, it can be seen that the stability error for vertical shifts in the MTCNN is five times higher than in Faster R-CNN. Hence, a comparable level between the frameworks cannot be established. However, to still comply with the selection of t_y around the individual mean stability errors and to compare the horizontal translation, t_y is also chosen to be 20.

There is also a different behavior of the stability error about rotations visible compared to the horizontal and vertical translations. While the stability errors for translations go up and down, the stability error for rotations primarily increases with rising rotation. This behavior can be exemplarily shown with figure 4.4. The ground truth bounding boxes of the example image in figure 4.4a are tightly fit around the objects, whereas the ground truth boxes in figure 4.4b are loose. Even if an object detector produces tight-fitting bounding boxes for the objects in figure 4.4b, the stability error would be high due to the dependency on the loose ground truth bounding boxes in the stability evaluation metric. The ground truth bounding boxes get loose when rotating the image with its annotations because there is no information about the object passed to the rotated bounding box. The edges of the rotated bounding boxes are always parallel to the x- and y-axis. When rotating an image and its annotations, the bounding box without recalculation would be skewed and not parallel with the axes anymore. The recalculated rotated bounding box aims to enclose all coordinates of the skewed box and therefore inflates. This inflation could be circumvented by calculating a rectangular convex hull of the object information in the form of segmentation coordinates. Unfortunately, these coordinates are only available for the COCO



(a) MTCNN stability error for $t_x \in P$

(b) MTCNN stability error for $t_y \in P$

(c) MTCNN stability error for $\theta \in P$

Figure 4.3: MTCNN STABILITY ERROR FOR AFFINE TRANSFORMATIONS. The stability error curves for each of the three affine transformations with MTCNN. (a) shows the stability error (y-axis) over an increase of horizontal translations (x-axis) in pixels. (b) shows the stability error (y-axis) over an increase of vertical translations (x-axis) in pixels. (c) shows the stability error (y-axis) over an increase of rotations (x-axis) in degrees. The horizontal orange line in all three graphs indicates the mean standard error for the respective curve.

dataset and are missing for the PASCAL VOC and WIDER Face datasets. Therefore, the strategy to prevent ground truth bounding box from inflating is to choose θ big enough (>3) that the transformation is noticeable but small enough (<10) to sustain relatively tight-fitting ground truth bounding boxes. Hence, the value chosen for θ is a 5-degree rotation.

By having decided on the transformation parameters to have values of $t_x = 20$, $t_y = 20$ and $\theta = 5$, the transformation matrices obtain the following form:

$$\mathbf{T_X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20 & 0 & 1 \end{bmatrix}, \mathbf{T_Y} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 20 & 1 \end{bmatrix}, \mathbf{T_R} = \begin{bmatrix} \cos(5) & \sin(5) & 0 \\ -\sin(5) & \cos(5) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.4}$$

When plugging $\mathbf{T_R}$ into equation 4.1, the rotated coordinates can be calculated according to Gonzalez et al. (2002) by vector matrix multiplication using the original coordinates v and w with

$$\begin{aligned} x &= v\cos(5) - w\sin(5), \\ y &= v\sin(5) + w\cos(5). \end{aligned} \tag{4.5}$$

The same vector matrix multiplication with $\begin{bmatrix} v & w & 1 \end{bmatrix}$, $\mathbf{T_X}$ and $\mathbf{T_Y}$ enables the calculation of the translated coordinates by t_x and t_y. Thereby, the translated coordinates x and y are the sum of the original coordinates v and w multiplied with t_x and t_y respectively.

In this chapter, a novel method is proposed that considers a stability evaluation metric that is initially intended for video detection and adopted to evaluate still images. Thereby, augmented versions of the images are sequentially arranged to form four frames. With this frame arrangement, a short sequence of moving objects is simulated to inspect the object and face detectors towards their ability to produce stable bounding boxes. Additionally, in this chapter, a range of possible transformation parameters are evaluated with a two-frame adoption to the proposed stability evaluation. Employed in an exploratory search, the parameter selection is compared between the frameworks Faster R-CNN and MTCNN to provide the best possible comparison between them, analyzed in upcoming chapters. This chapter is paramount to measure the impact of the NMS methods introduced in the previous chapter and therefore contributes to the achievement of the overall goal of this thesis. The proposed method and selected parameters directly influence the experimental setup described in the next chapter.

(a) Example image rotated by 2 degrees

(b) Example image rotated by 20 degrees

Figure 4.4: GROUND TRUTH COMPARISON OF DIFFERENTLY ROTATED IMAGES. The tightness of the ground truth bounding boxes changes with the degrees of rotation. (a) shows the example image with a rotation of 2 degrees with the corresponding rotated bounding boxes. (b) shows the same example image with its ground truth boxes rotated by 20 degrees.

Chapter 5

Experimental Framework

"Insanity is doing the same thing over and over again, but expecting different results."

Rita Mae Brown, Sudden Death

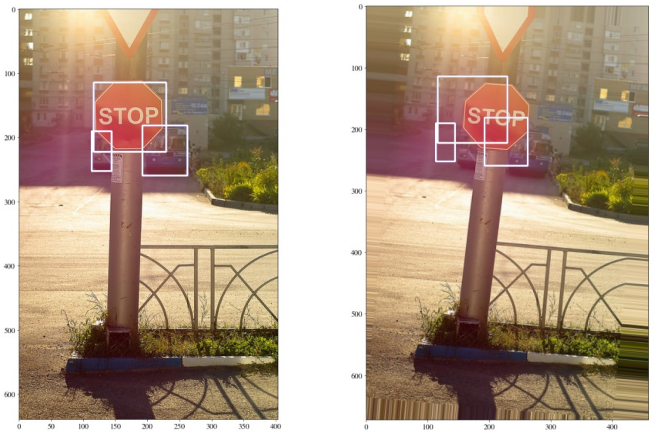
The aim of the experimental framework chapter is to conceptually guide through the implementation of the method proposed in chapter 3 and the establishment of the accuracy and stability measurement based on the procedures introduced in chapters 2 and 4. This chapter follows the pipeline structure, starting with the data preparation, the framework, and model selection, and it ends with the two different perspectives on evaluating the predictions. During each stage, the relevant concepts and decisions based on the related work are further elaborated to maintain the reproducibility of the results. The generated results are then summarized in the next chapter.

5.1 Data Preparation

All of the three datasets, regardless of the origin, follow the same procedure. First, validation/test images and their respective ground truth annotations are loaded. Then, simple transformations are applied and prepared for uniform output functionality. These steps are all integrated with a class for each of the datasets individually. The uniform implementation of these dataset classes is described in detail and provided in the appendix A.2. In this section, the supplementation of these dataset classes is described with the implementation of the affine transformations from chapter 4. The embedding of these transformations into the data preparation is essential to provide a consistent data flow into the streamlined prediction and evaluation pipeline of the following sections.

By building on top of the theoretical groundwork of the previous chapter 4 to enable a stability evaluation with a sequence of augmented still images, additional practical implementation-related topics arise. To replicate the desired affine transformations with its parameters, different approaches have been considered. The PyTorch way of transforming data leads to the problem of unilateral transformations as depicted in figure 5.1. The explanation related to this problem is stated in the appendix A.2 and not further detailed at this point.

In order to apply the same transformation of image and its associated annotations, another approach have to be chosen instead of `torchvision.transforms.RandomAffine`. An alternative way of transforming data is provided by the Python package Albumentations (Buslaev et al., 2020). Within this package, multiple geometric transformations are available, which enable the same functionalities as `RandomAffine` from PyTorch. Such an alternative is `Affine` from



(a) Original example image with its ground truth bounding boxes

(b) Example image rotated by 5 degrees with `fit_output = True`

Figure 5.1: UNILATERAL TRANSFORMATION. (a) shows the original example image without any transformation together with the associated ground truth bounding boxes.(b) shows the same example image rotated. Only the PyTorch unilateral transformation of the image was applied. The annotations stay at the same coordinates as in (a).

`albumentations.augmentations.geometric.transforms`.[1] By utilizing an object of this class, images and annotations can be translated on the x and y axes and rotated.[1] Additional parameters of this class are `fit_output`, `mode` and `always_apply`. With the parameter `always_apply` the probabilistic application of this transform object is ruled out and the transformation is done every time. In order to compare the outcome of the bounding box stabilization it is vital to transform every image and annotation of the dataset. The parameter `fit_output`, which accepts a boolean value, can be interpreted as a second transformation to center the image such that it fits the image plane.[1] However, if a horizontal or vertical translation is applied as the targeted transformation, `fit_output = True` would negate this affine transformation by centring the image again.[1] Therefore, `fit_output` should only be applied for rotational transformations. Although, if applied to rotations the shape of the image changes to fit the whole image including the rotation as seen in figure 5.2. By fitting the output with a rotation, the whole information in the image can be processed and potential objects located in the corners, can be detected. The resulting width and hight after the rotation can be calculated according to equation (5.1).[2]

$$\begin{aligned} \text{new width} &= \text{height}\sin(\theta) + \text{width}\cos(\theta) \\ \text{new height} &= \text{height}\cos(\theta) + \text{width}\sin(\theta). \end{aligned} \tag{5.1}$$

The `mode` parameter references the OpenCV border flag and can be applied to all of the three transformation types.[1] By translating or rotating an image, the shape of the image remains unchanged but the pixel values get shifted to the right (shift in x direction with $t_x > 0$), down (shift in y direction with $t_y > 0$) or rotated according to equations (4.4) and (4.3). This means that the

[1] Albumentations Affine Transformations: https://albumentations.ai/docs/api_reference/augmentations/geometric/transforms/

[2] Image Rotation using OpenCV on Github: https://cristianpb.github.io/blog/image-rotation-opencv

(a) Example image with horizontal and vertical translation neglected by `fit_output = True`

(b) Example image rotated by 5 degrees with `fit_output = True`

(c) Example image rotated by 5 degrees with `fit_output = False`

Figure 5.2: EXAMPLE OF CHANGES IN SHAPE WITH ROTATION. Fitting the transformed image to the output has two different effects on translations and rotations. (a) shows the example image, translated by 20 pixels in positive y direction and 20 pixels in positive x direction. Nevertheless, `fit_output = True` negates these translations and re-centers the image. (b) shows the same example image also rotated by 5 degrees but with `fit_output = False`. The image size stays the same but the corners of the rotated image are disregarded since they lay outside of the original image size. (c) shows the example image rotated, also with `fit_output = True`. With this rotated example the image size changes with regard to equation (5.1).

content of the image in case of a translation in positive x direction, moves by t_x pixels to the right. By comparing f_1 and f_2 of figure 4.1, this shift is visualized. When looked at the first colored column (very left border) of the image in frame f_1 $M \in [0, 256]^{C \times H \times W}$. Then the first column is represented as $M_{[:,:,0]}$ and has the same values as the transformed version $M'_{[:,:,t_x]}$ in the column t_x after a shift by t_x. This shift on the very right border has the effect that pixel values in the span of $M_{[:,:,W-1-t_x:W-1]}$ are disregarded and the column $M_{[:,:,W-t_x-1]}$ are at the position of $M'_{[:,:,W-1]}$ in the transformed image. By shifting the pixel values in the image M by t_x pixels to the right, the values in $M'_{[:,:,0:t_x-1]}$ would all be 0 if the parameter `mode` is left on its default value 0. When `mode = 1` the pixel values at $M'_{[:,:,0:t_x-1]}$ would all be interpolated with $M'_{[:,:,t_x]}$, which are the same values as in $M_{[:,:,0]}$. The same procedure applies for shifts in either positive or negative direction of x and y and also for rotations around the image center.

By embedding these transformation steps into the data preparation pipeline in the dataset classes, the foundation is laid out to predict and evaluate the augmented images in the same way. The consistency and compatibility between data preparation, prediction, and evaluation pipeline are important in combining predictions on differently augmented data and comparing these combinations between models and frameworks.

5.2 Object and Face Detection

The detection pipeline comprises the application of the core contribution of this thesis by replacing Non-Maximum-Suppression at multiple stages of the object and face detectors. As seen in the dataset section 2.1, multiple datasets distinguish themselves in terms of object classes located within the images. Therefore, every dataset requires a dedicated model to localize these objects and then correctly assign them to the corresponding object class. Since both COCO and PASCAL VOC contain multiple classes of objects, these objects are detected with the help of two differently trained Faster R-CNN models. On the other hand, to localize one single object class – faces in the images of the WIDER Face dataset, an MTCNN model is employed. In this chapter, multiple connections with previously mentioned topics can be made. First, two different detection architectures are further elaborated, which are theoretically introduced in section 2.3. Then, the different stages in the frameworks are identified to point out where Multi-Non-Maximum-Suppression replaces the original implementation of classic greedy NMS.

5.2.1 Faster R-CNN

About the theoretical background, provided in section 2.2.2, in this thesis, a Faster R-CNN as a multi-class object detection framework is applied. A pre-trained version on the COCO 2017 training set is available `torchvision.models.detection`, which also employs a ResNet-50 FPN backbone, which in turn is pre-trained on Imagenet.[3] In comparison with Ren et al. (2015) and He et al. (2016), the RPN with a ResNet-50 FPN backbone used in this thesis differs in two points. First, the fully connected output layer with softmax activation function is eliminated from the ResNet-50 backbone. This output layer is intended for the ImageNet classification. This reduction occurs because the integration of the FPN is due to obtaining its feature maps instead of probabilistic outputs from the fully connected layer. The output of the second last layer in the ResNet-50 is a convolutional feature map. It serves in the Faster R-CNN as input for the downstream step of proposing regions and classifying objects. This means that the ResNet-50 backbone with 50 layers only contributes 49 layers to the Faster R-CNN model. This reduction is already made by PyTorch when loading the backbone, and the difference to He et al. (2016) is stated for completeness.

Second, by applying FPN in the RPN, not only a single scale feature map gets consulted to propose regions but rather multiple levels of a feature pyramid (Lin et al., 2017a). The original implementation of the downstream step of region proposals in the RPN as seen in Ren et al. (2015) applies a small subnetwork of sliding windows on top of a single convolutional feature map. By sliding over the convolutional feature map, this small subnetwork takes rectangular spatial windows of the size 3×3 as input. It reduces the dimensionality to a 256-dimensional feature, representing the input for two sibling fully connected layers. Multiple reference boxes with three distinct scales and three different aspect ratios, called anchors, are predicted to propose regions of different shapes at each rectangular sliding window. Meaning that at each position of the sliding window, nine anchors get produced, reduced in dimensionality, and fed into two fully connected layers. These two fully connected layers are one box regression layer and one box classification layer. The box classification layer in the RPN is class-agnostic and only distinguishes between an object and background. As a result, the box regression layer outputs per sliding window and anchor four coordinates per box. The maximum of 36 possible proposal coordinates for nine possible proposal boxes gets predicted in the default case. Similarly, the box classification layer predicts either the membership of a set of classes or a background class in the form of an objectness score per anchor, which results in a maximum of 18 scores per sliding window position.

[3] `Torchvision Models:` https://pytorch.org/vision/stable/models.html#object-detection-instance-segmentation-and-person-keypoint-detection

Figure 5.3: COCO DATASET EXAMPLE IMAGE. The image considered as example for dimension exploration in FPN, RPN and Fast R-CNN predictor.

To explore the inner workings of the Feature Pyramid Network, an example image (figure 5.3) of the COCO dataset is passed through the network, and the dimensions are recorded. For this example of the FPN in symphony with the ResNet-50 backbone, an image is considered with the size $3 \times 426 \times 640$. As a first step the image gets normalized with means $[0.485, 0.456, 0.406]$ and standard deviation $[0.229, 0.224, 0.225]$. The image also gets scaled to $3 \times 800 \times 1201$ by calculating a scaling factor based on the minimal and maximal value of width and height and fixed minimal and maximal values of 800 and 1333, respectively. The normalized and rescaled image serves as input to the ResNet-50 backbone. With the bottom-up pathway via forward pass through the convolutional net, the outputs of the convolutional blocks conv2-conv5 have sizes of $C_1 \in R^{64\times400\times608}$, $C_2 \in R^{256\times200\times304}$, $C_3 \in R^{512\times100\times152}$, $C_4 \in R^{1024\times50\times76}$, $C_5 \in R^{2048\times25\times38}$ with the pixel values $\{x \in R \mid 0 \leq x \leq 1\}$. Note that between the input of the backbone and output of C_1, padding of $(3, 3)$ is applied. The output shapes of $\{C_1, C_2, C_3, C_4, C_5\}$ stay about the input shape with the shape of $3 \times 800 \times 1201$ by 2×2 max-pooling of in every convolutional block. As mentioned by introducing FPN in chapter 2 C_1 is not considered by the bottom-up pathway of the FPN due to its large spatial resolution. The top-down pathway starts at the very end with the output of C_5 and applies a 1×1 convolution via the lateral connection to reduce the dimensions from 2048 to 256 (Lin et al., 2017a). By upsampling again with a factor of 2, the top-down pathway goes the other way around as the bottom-up pathway. It merges via element-wise addition the respective feature map from its subsampled counterpart. Each merged map undergoes another convolution, this time with kernel size 3×3. The resulting feature maps have shapes $P_2 \in R^{256\times200\times304}$, $P_3 \in R^{256\times100\times152}$, $P_4 \in R^{256\times50\times76}$, $P_5 \in R^{256\times25\times38}$. P_6 gets produced by 2×2 max pooling P_5 and has the shape of $256 \times 13 \times 19$. An example without P_6 is depicted in figure 5.4.[4]

By proposing regions based on multiple levels of a feature pyramid with FPN as introduced in chapter 2, the sliding window remains the same with a kernel size of 3×3 (Lin et al., 2017a).

[4] Understanding Feature Pyramid Networks for object detection (FPN) on Medium: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

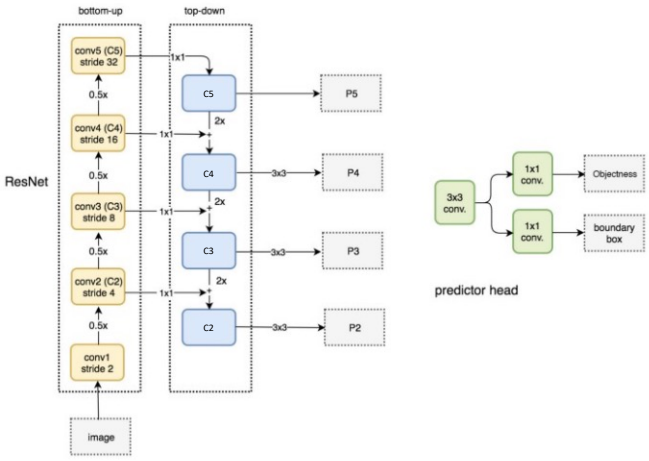


Figure 5.4: DETAILED FEATURE PYRAMID NETWORK. Example illustration to create $\{C_2, C_3, C_4, C_5\}$ and $\{P_2, P_3, P_4, P_5\}$ for the RoI head. The RPN head additionally uses P_6 from max-pooling P_5. *Source: Hui, Jonathan*[4]

The difference is that this sliding window is applied to every level of the feature pyramid instead of a single-scale feature map. This approach makes it obsolete to produce multi-scale anchors for one level because anchors of a single scale are assigned to each level of the feature pyramid, which already represents multi-scale feature maps. In the case of the Region Proposal Network, with FPN, not only four levels of the feature pyramid get produced but the additional fifth as mentioned in chapter 2 as well. This results in 5 differently scaled feature maps where the 3×3 sliding window is applied to generate anchors with areas of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$. Furthermore, Ren et al. (2015) also produce three different aspect ratios of the anchors. This is also done with the FPN approach by using anchors with aspect ratios $\{1{:}2,\ 1{:}1,\ 2{:}1\}$, which results in a total of 15 anchors over the whole feature pyramid. Concerning the example above where it is shown how $\{P_2, P_3, P_4, P_5, P_6\}$ are generated by the FPN, the RPN utilizes these feature maps further. An illustration therefore is depicted in figure 5.4. The forward pass of the Regional Proposal Network takes these five feature maps and passes them first through a 3×3 convolutional layer and then through two siblings 1×1 convolutions to regress bounding boxes and predict object/non-object based on each feature map and the three aspect ratios. For example, when P_6 with size $256 \times 13 \times 19$ gets passed through the RPN head, the sliding window with 3×3 and the two attached sibling convolutional layers produce per sliding window and per aspect ratio one objectness score and four bounding box coordinates. The feature map of size $\times 13 \times 19$, is 247 possible locations for a sliding window. Therefore, $247 * 3 = 741$ anchors are produced. Analogously, with this procedure the RPN head generates for P_5 with size $256 \times 25 \times 38$ a total of $25 * 38 * 3 = 2'850$ anchors. On P_4, P_3 and P_2 are then 11'400, 45600 and 182'400 anchors produced respectively. This results in a total of 242'991 anchors over the whole image, which all get classified with an objectness score and regressed with four bounding box coordinates about those anchors. As mentioned above and in 2 the classification and bounding box regression layer are trained on the COCO 2017 training set and assign object/non-object classes and bounding box coordinates based on the learned weights.

The result of the RPN head comprises 242'911 proposals, each with bounding box coordinates and an objectness score, which need to be filtered to output the final proposals. The filtering process distinguishes between training and testing by limiting the allowed proposals. During

training, the top 2'000 and during testing only the top 1'000 proposals are maximally allowed per level (Ren et al., 2015). This filter reduces the number of proposals in the example to 4'741 proposals. Each proposal then is clamped to the boundary of the image size. This clamping particularly applies to proposals at the border of the image. In the example image, 1'183 coordinates were adjusted to the size of the image. Furthermore, proposals with bounding boxes with a tiny area get removed. As the final step of the filtering process, the RPN Non-Maximum Suppression gets applied to the remaining 4'741 proposals. At this stage, the original implementation is replaced with the Multi-Non-Maximum Suppression function introduced in chapter 3. With the default values of NMS (IoU threshold = 0.7, multi-level distinction = $True$, number of allowed proposals = $1'000$ and classic greedy NMS as the method) the RPN outputs 1'420 proposals. These proposals have five components; each proposal has four bounding box coordinates and an assigned objectness score. According to the RPN, these proposals should cover a region of an object with a certainty represented with the objectness score. Further down the prediction pipeline, these proposals are processed with a Fast R-CNN predictor to predict the refined bounding box coordinates and the class membership of the specific classes of a given dataset.

Fast R-CNN Predictor

The last step of the unified Faster R-CNN is to predict refined bounding boxes and class membership probabilities by materializing its second module – the Fast R-CNN predictor with an RoI pooling layer (Ren et al., 2015; Girshick, 2015). As introduced in chapter 2, the Fast R-CNN requires both an input image and region proposals first to perform the Region of Interest (RoI) pooling. The Faster R-CNN utilization of the Fast R-CNN predictor waives the usage of convolutional layers, which generate, according to Girshick (2015), the single feature map. Since not only one single-scale feature map but rather a multi-scale feature pyramid gets produced with FPN in the backbone, the feature pyramid then serves as input to both the Regional Proposal Network and the Fast R-CNN predictor (Lin et al., 2014). Because of this change in the input of the Fast R-CNN predictor, the subsequent RoI pooling has to be adjusted to assign multi-scale RoIs to the according to pyramid levels (Lin et al., 2017a).

In contrast to the RPN, the RoI pooling only considers four of the five available pyramid levels of the feature pyramid generated by the FPN. RoI of different scales gets assigned to the remaining levels of the feature pyramid about the scale of the input image. This assignment is done by inferring the scale based on the fact that during FPN, a subsampling factor of 2 is used, and therefore, scaling factors $0.25, 0.125, 0.0625, 0.03125$ are applied on the respective level of the feature pyramid. The RoI pooling then extracts 7×7 features from all level proposals with a channel dimension of size 256. The extracted features are sent through two 1'024 dimensional fully connected layers. The input shape of the first layer has a fixed size respective to the extracted features $7 * 7 * 256 = 12'544$. The output of the second fully connected layer then undergoes the final prediction of the Fast R-CNN predictor with a sibling output layer of a class-specific classifier and a bounding box regressor (Lin et al., 2017a; Girshick, 2015). The layer with the classifier outputs a softmax probability estimate over $K + 1$ object classes (K dataset-specific classes and one background class). The regression layer predicts for each of the K classes a refined bounding box with four coordinates.

Since the Fast R-CNN predictor is not class-agnostic, the sibling output layers need to be adjusted to the number of classes K of either COCO or PASCAL VOC. Ren et al. (2015) state in their experiments that fine-tuning of the Faster R-CNN for PASCAL VOC is not necessary since the COCO categories are a superset of the PASCAL VOC and only the softmax layer has to be adjusted to the 20 PASCAL VOC categories plus background class. Even though PASCAL VOC-specific fine-tuning is not necessary, Ren et al. (2015) also state that fine-tuning improves the accuracy of the model. Therefore, a fine-tuning is conducted for the PASCAL VOC dataset on the

Faster R-CNN. The default settings with the pre-trained Faster R-CNN model from PyTorch are based on the COCO classes and trained for that purpose.[3] For simplification of the detection on images from the PASCAL VOC dataset, only the RoI pooling layer and the Fast R-CNN predictor of the Faster R-CNN are replaced and instantiated with the number of classes to be $K = 21$. The ResNet-50 backbone with FPN pre-trained on ImageNet and the RPN pre-trained on COCO 2017 remain the same. However, when instantiating the Faster R-CNN with the ResNet-50 backbone, the last three convolutional blocks get unfrozen per default and are considered during training.

Similarly, the RPN automatically adjusts to training mode as well and gets fine-tuned during training iterations. The fine-tuning of the Faster R-CNN model for PASCAL VOC is manually trained on the union of training and validation set of the PASCAL VOC dataset about chapter 2.1. The hyperparameters for this training process are selected to run for ten epochs with Stochastic Gradient Descent (SGD) optimizer, a learning rate of 0.0001, momentum of 0.9, weight decay of 0.000001. Furthermore, a learning rate scheduler gets applied with a step size of 3 and a gamma of 0.1. Since the main focus of this thesis lies in the evaluation of the bounding box predictions of object and face detectors, the training procedure is not explored in more detail.

With regard to the example from the beginning of this section, the number of detections as the output of the Fast R-CNN predictor remains unchanged to the number of proposals at the output of the RPN. However, these proposals serve to reference the multi-scale feature maps of the feature pyramid for the RoI pooling and class-specific feature generation in the Fast R-CNN predictor. As the last step, similarly to the RPN post-processing, the detections undergo a filtering process. After classifying and regressing over the $K+1$ object classes, the example image from the COCO dataset generates an output of the Fast R-CNN predictor of 1'420 probability estimates for 81 classes and 1'420 bounding boxes with four bounding box coordinates for 81 classes as well. The boxes get first clamped to the image size, which affects 7'421 bounding box coordinates. In the next step, the scores get arranged to represent the 81 classes, and the background class can be removed. The bounding boxes, classification scores, and labels get reshaped to represent a separate instance into one dimension. This reshapes a two-dimensional tensor of detection scores from $1'420 \times 80$ into a single dimension with 113'600 instances. Every bounding box, label, and score with a respective score below 0.05 gets filtered out, which leaves from 113'600 instances only 655 detections.

Furthermore, predictions associated with minimal bounding box areas get removed as well. As the final step of the Faster R-CNN post-processing in the RoI heads, the Non-Maximum-Suppression is applied to the remaining 655 bounding boxes with their respective score and labels. Similar to RPN, the Multi-Non-Maximum Suppression function replaces the original implementation of classic greedy NMS. The default values for this stage NMS are an IoU threshold of 0.7 with the classic greedy NMS method, multi-class distinction, and a limit of 1000 detections per image during testing. With these default values, the final number of detections of the Faster R-CNN for the example image is 88. According to the classification score distribution depicted in figure 5.5, most of the 88 detected objects have low confidence. Compared to the ground truth annotations, the image contains 20 objects. If the predictions were filtered by a score threshold of 0.7, only 19 detections would remain that cover almost all ground truth boxes. The Faster R-CNN predictions get only slightly filtered by detection scores during the post-processing. The intention behind including even low-scoring detections into the output may be due to the holistic output consideration during the accuracy evaluation. Further implementation details regarding the Faster R-CNN post-processing during RPN and RoI Pooling are explained in the appendix A.3.

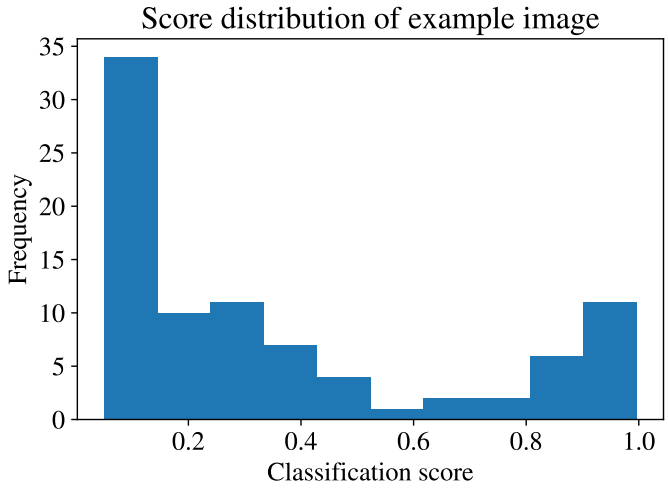


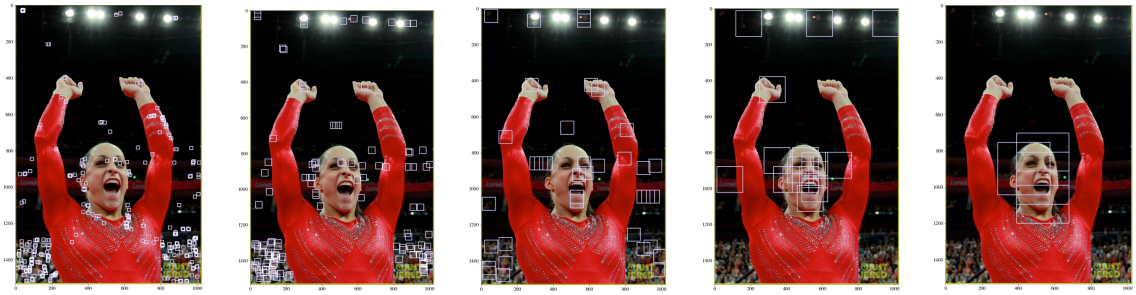
Figure 5.5: FASTER R-CNN EXAMPLE CLASSIFICATION SCORES. The classification scores for the example image in figure 5.3

5.2.2 MTCNN

The Multi-Task Cascaded Convolutional Neural Network (MTCNN) as introduced in 2.2.2 is the framework used in this thesis to detect faces and localize facial landmarks in images of the WIDER Face dataset. A publicly available implementation of the MTCNN using PyTorch can be found in the Github repository, called Facenet-PyTorch.[5] The PyTorch implementation is based on a TensorFlow implementation of Sandberg[6], which is itself based on the original implementation of Zhang in Matlab.[7] The implementation in the Facenet-PyTorch repository is oriented at the architecture of Zhang et al. (2016) and provides pre-trained versions of the P-Net, R-Net, and O-Net. Unfortunately, the author of Facenet-PyTorch does not clearly state what training procedure was applied to the different stages of MTCNN.[7] Therefore, it is assumed to follow the same training process described in Zhang et al. (2016) and the model weights to be converted versions of the models from the Github repository of Zhang.[7] Similarly to the description of the detection pipeline in the previous section 5.2.1, the forward pass is exemplarily shown with an image from the WIDER Face dataset. During the detection in the different stages of the MTCNN, the presence of NMS is pointed out to indicate where classic greedy NMS is replaced with Multi-Non-Maximum Suppression for the experimental purpose of this thesis.

By referring back to the theoretical foundation of MTCNNs in 2.2.2, it is shown that the detection pipeline starts with building an image pyramid before passing the data through the first stage of the framework. Considering an example image of size $1'534 \times 1'024 \times 3$ that gets extracted from the WIDER Face specific PyTorch data loader. By loading the images and annotations via the PyTorch data loader, the pixel values of the images are not normalized because the MTCNN framework normalizes them during detection itself. Therefore, the images can directly be fed into the preprocessing steps of the detection. As described in 2.2.2, the image pyramid builds up with the input image in different scales. The initial scale gets defined by $12/minsize$, where $minsize$ is a parameter that can be defined during the model instantiation and refers to the minimal size

[5] Facenet-PyTorch on Github: https://github.com/timesler/facenet-pytorch
[6] Face Recognition using Tensorflow on Github: https://github.com/davidsandberg/facenet
[7] MTCNN Face Detection Alignment on Github: https://github.com/kpzhang93/MTCNN_face_detection_alignment

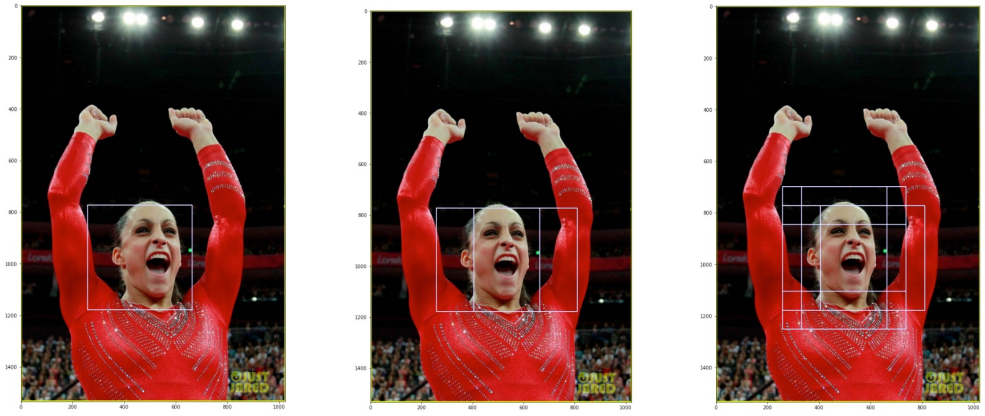


(a) Proposals from scale 0.6 (b) Proposals from scale 0.3016 (c) Proposals from scale 0.1516 (d) Proposals from scale 0.0762 (e) Proposals from scale 0.0383

Figure 5.6: MTCNN MULTI-SCALE PROPOSALS. Selection of proposals made on different scales. With the largest scale (a) the P-Net produces the smallest bounding boxes. As the scale decreases in (b), (c) and (d), the proposed bounding box size increases. One of the smaller of the total of 12 different scales is depicted in (e) with comparably the largest bounding boxes of all examples.

a face can have during detection. Per default the minimal face size is set to $minsize = 20$ and therefore, the initial scale is set to $12/20 = 0.6$. If the initial scale multiplied by the smaller height and width is at least 12 pixels, the scale gets added to the image pyramid. In the case of the example image, the orientation is vertical, and the smaller value of the size is the width with 1'024 pixels. When calculating the first scale about the minimal face size, the width of the first scaled image of the image pyramid becomes $1'024 * 0.6 = 614.4$. If the initial scale gets added to the pyramid, then the scale is discounted by a factor of 0.709 and continued until the minimal side of the image is smaller than 12 pixels. This factor is also a parameter of the MTCNN model and is chosen to stay at the default value. With this procedure, 12 different scales get added to the pyramid, with 0.6 being the largest and 0.014 being the smallest. When keeping the input image size unchanged, the bigger the minimal face size is allowed to be, the smaller gets the initial scale, and the fewer images can be collected for the image pyramid. However, if the minimal face size is exactly 12, the initial scale is one, and the image pyramid gets constructed with 13 scaled images. The smaller the minimal face size gets, the larger the initial scale, and more images fit into the image pyramid. Although there are more images in the image pyramid, the initial scale is in the case of $minsize < 12$ larger than 1, which means that the largest image in the pyramid is larger than the input image itself. Generally, the small images in the image pyramid enable the framework to detect large faces and the large images in the pyramid give MTCNN the ability to detect small faces. To visualize this inversely proportional dependence, in figure 5.9 the proposals (before NMS) for differently scaled images from the image pyramid are plotted, which were detected in scales of $\{0.6, 0.3016, 0.1516, 0.0762, 0.0383\}$ and rescaled relative to the image size.

Within a loop of all scaled images in the image pyramid, every scaled image gets normalized and undergoes a forward pass of the proposal net (P-Net). The P-Net consists of a sequence of 3 convolutional layers, followed by a parametric rectified linear unit (PReLU) as an activation function. After the first sequence of 3×3 convolution, max pooling with a stride of 2 is applied, halting the spatial dimensions. A sibling convolutional layer is applied after the third sequence of 3×3 convolution and PReLU activation function. One 1×1 convolution regresses the rectangular bounding boxes of the proposals, and the other 1×1 convolution feeds in a softmax layer to estimate the probabilities of the detected proposals. The sibling output layer of the P-Net produces $456 * 303 = 138'168$ proposals at the scale of 0.6. Four bounding box coordinates are regressed for each of these proposals, and two probabilities of being face/non-face are estimated. All proposals with a probability of being a face below a threshold (0.6 per default for the first stage) are



(a) NMS with IoU threshold of 0

(b) NMS with IoU threshold of 0.5

(c) NMS with IoU threshold of 1

Figure 5.7: MTCNN P-NET NMS. (a) IoU threshold of 0, only the maximal bounding box gets selected and all others are suppressed. (b) IoU threshold of 0.5 default settings for scale-specific NMS in the P-Net. (c) IoU threshold of 1, all proposed bounding boxes get selected if they do not overlap to 100%

disregarded. The example image produced in the P-Net with a scale of 0.6, 665 proposals with a face probability greater or equal to 0.6. The other 137'503 proposals have a face probability below 0.6 and are filtered out. The bounding box coordinates of the remaining 665 proposals are then rescaled with a factor of 2 to compensate for the stride of 2 in the max-pooling layer. Within each scale, overlapping bounding boxes with an IoU of 0.5 are then suppressed with the classic greedy Non-Maximum-Suppression (5.7b). In order to visualize the influence of NMS, especially with varying IoU threshold can be seen in figure 5.7, where the same classic greedy NMS function is applied at the same scale of the example image pyramid (0.02716) but with different IoU thresholds. Having the IoU threshold towards 0, neighboring bounding boxes get suppressed even with low overlap (figure 5.7a).

On the other hand, the IoU threshold towards 1 allows bounding boxes to be highly overlapping before suppressing the non-maximal ones (figure 5.7c). This NMS function call is replaced with the Multi-NMS function, comparable to the substitution mentioned in section 5.2.1. The application of classic greedy NMS suppresses 379 proposals, which leaves 286 proposals for the first of 12 scales. In total, for all 12 scales, there are 273'080 detections and 1'842 proposals left after comparing the probabilities to the threshold and 784 proposals after suppressing overlapping bounding boxes with greedy NMS for the example image. After the scale-specific application of NMS, an additional Non-Maximum-Suppression is performed to suppress overlapping bounding boxes across all scales with an IoU threshold of 0.7. This NMS function is also subject to substitution by Multi-NMS. The number of proposals remains the same for the example image after applying overall NMS to all scales.

To prepare the remaining proposals after detection and filtering of the first stage, the proposal references on the input image are cropped out of the image in a 24×24 rectangle and normalized. An example of such a crop before normalization can be seen in figure **??**. This results in an input shape of the R-Net of 784 proposals with $3 \times 24 \times 24$. The R-Net applies two sequences of 3×3 convolution, PReLU activation function, and max pooling with stride 2. Then a sequence of 2×2 convolution, followed by a PReLU activation function and a dense layer with another PReLU activation function. The output of the activated dense layer feeds in a sibling dense layer. One regresses the refined bounding box coordinates, and a softmax activation function follows the other

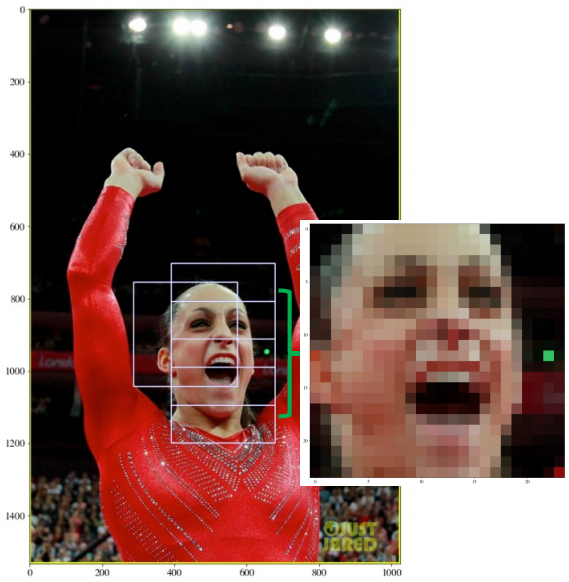
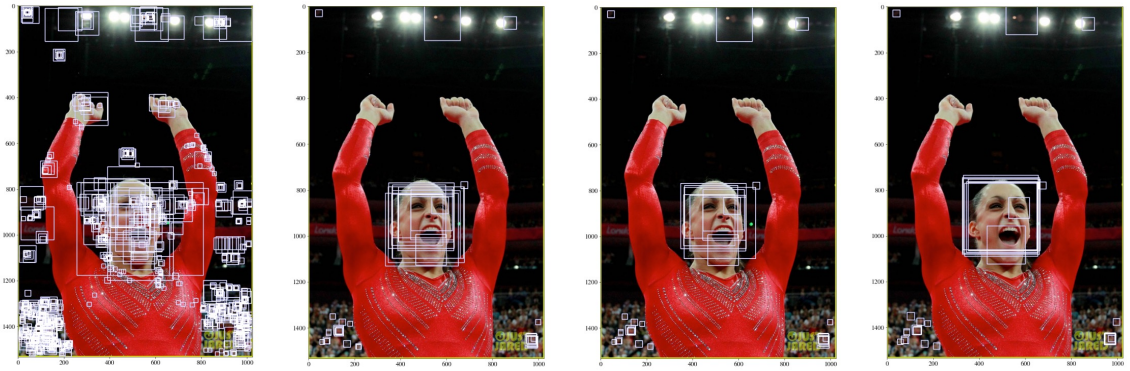


Figure 5.8: MTCNN R-NET PROPOSAL CROP. A high scoring proposal bounding box being cropped out into a $3 \times 24 \times 24$ rectangle

one to estimate the binary probabilities of the refined bounding box containing a face/non-face. By comparing the output probability estimates with a pre-defined threshold of 0.7, the proposals of the respective scores below that threshold are disregarded. Concerning the example image, there are 742 binary probability estimates below that threshold and 42 above it. In figure 5.9a are all P-Net proposals plotted and in figure 5.9b only those with a probability threshold above 0.7. Before the refinement of the R-Net is applied to the proposals, the second stage NMS is applied to the bounding boxes with an IoU threshold of 0.7, which is visualized in figure 5.9c. This application of NMS is similar to the NMS over all scales of the P-Net, with the difference that bounding boxes with a low probability score in the R-Net are filtered out. The R-Net NMS additionally suppresses eight bounding boxes. This NMS function is like the two NMS functions from P-Net subject to replacement with Multi-NMS introduced in chapter 3. The refinement of the R-Net takes place with the remaining 34 bounding boxes, which still possess the same coordinates as detected in the P-Net. By correcting the bounding box coordinates with the regression outputs of the P-Net, the proposals change significantly, as displayed in figure 5.9d.

As mentioned by introducing MTCNN in section 2.2.2, the third stage works similarly to the R-Net. The 34 refined bounding boxes of the R-Net are first cropped out as seen in figure 5.8 but in a larger size of $3 \times 48, \times 48$ and then fed into the O-Net. The O-Net consists of 3 sequences of 3×3 convolutions, PReLU activation functions, and a max-pooling with stride 2. Then a 2×2 convolution followed by a PReLU is applied with a fully connected layer activated again by a PReLU further downstream. The output of the O-Net is a triplet fully connected layer with one layer followed by a softmax activation function for the probability estimates, one layer for the bounding box regression, and one layer for the facial landmarks with an output size of 10. The output layer reflects the outputs obtained when passing the example image through the O-Net, where 34×4 bounding box coordinates, 34×2 probability estimates, and 34×10 facial landmark coordinates are returned. The same filtering procedure of R-Net applies to the O-Net as well. First, all proposals with a probability estimate below the threshold of 0.7 are filtered out. The bounding boxes are still based on the refinement made in the R-Net but classified differently during the



(a) All proposals after the P-Net forward pass

(b) Filtered proposals after R-Net forward pass

(c) Proposals after R-Net NMS

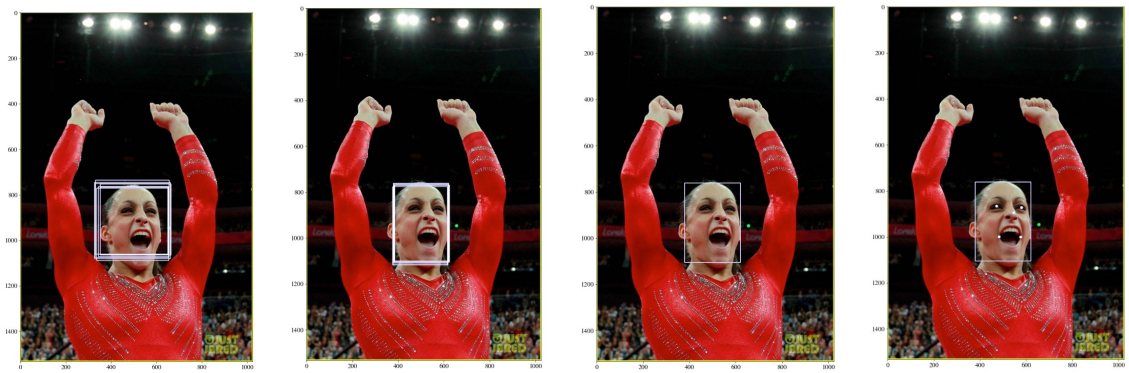
(d) Refined proposals after filtering and NMS application in R-Net

Figure 5.9: MTCNN R-NET STEPS. From all proposals of the P-Net (a) the R-Net estimates binary classification probabilities and filters out low scoring proposals (b). Then NMS is applied to the remaining proposals (c) and the regression output of the R-Net is used to correct bounding box coordinates of the proposals after NMS (d)

forward pass through the O-Net. The seven remaining R-Net refined bounding boxes that are above the threshold are plotted in figure 5.10a. After the filtering process, the bounding boxes get refined with the regression output obtained in the O-Net (5.10b), and then the final round of classic greedy NMS with an IoU threshold of 0.7 is applied. These two steps are switched compared to the second stage of MTCNN. As already mentioned in the first two stages, Multi-NMS replaces classic greedy NMS here as well. After applying the final NMS, there is only one detection left for the example image at hand. This final detection is depicted in figure 5.10c. The final detection has three elements: 4 bounding box coordinates proposed in the P-Net, refined by the R-Net and the O-Net, a probability estimate, and ten facial landmark coordinates that indicate the locations of both eyes, the nose, and both corners of the mouth. The final output of the MTCNN is displayed in figure 5.10d. Additional implementation details of MTCNN are stated in the appendix A.4.

5.3 Evaluation

The construction of an evaluation pipeline with regard to multiple evaluation objectives is a challenging task, considering there are three distinct datasets subject to analysis for this thesis. The objectives for evaluation are twofold: (I) Since an essential part of the frameworks Faster R-CNN and MTCNN is altered by implementing multiple different NMS functions within Multi-NMS, it must be ensured that the frameworks still perform object detection with roughly the same accuracy compared to the baseline. Therefore, the classic greedy NMS, which is the same as in the original implementation of the frameworks, is re-implemented in Multi-NMS as mentioned in chapter 3. The primary metric for inter framework and inter-model comparison is the detection mean Average Precision (mAP) or Average Precision (AP) as mentioned in section 2.4. Therefore, the first evaluation objective is to create a baseline with the original implementation of NMS in Faster R-CNN and MTCNN for the respective datasets and then compare the detection accuracy in the form of mAP with multiple different experimental procedures executions. These different executions of the accuracy evaluation consist of 64 runs for each of the datasets COCO and PAS-



(a) R-Net detections after filtering low scoring bounding boxes in the O-Net

(b) O-Net detections after regression adjustments

(c) O-Net detections after NMS

(d) Final O-Net detection

Figure 5.10: MTCNN O-NET STEPS. (a) Proposals made in P-Net and refined in R-Net are classified in the O-Net and are only kept if the probability estimates are above 0.7. (b) The refined bounding boxes of the R-Net are again refined after regressing them in the O-Net. (c) The final detection bounding boxes after applying NMS to the O-Net refined bounding boxes. (d) The output of the MTCNN of the example image with the final remaining bounding box, 5 facial landmarks and a probability estimate of 0.9999

CAL VOC with the Faster R-CNN and 256 runs for the WIDER Face dataset with MTCNN. (II) The overall goal of this thesis is to stabilize bounding boxes, and consequently, this goal has to be measurable. Unfortunately, the stability of bounding boxes about affine transformations is only partially compatible with the first evaluation objective. Since mAP and AP are strongly dependent on the IoU metric and thresholds, they are not an adequate measurement for the bounding box stability as mentioned in chapter 4. Hence, the second evaluation objective is to combine the different experimental executions from the accuracy evaluation to evaluate the stability of the dataset sequence, and NMS method-specific runs. An overall goal of the evaluation pipeline is to create a consistent implementation of shared metrics to establish comparability between the frameworks, datasets, and NMS methods.

5.3.1 Accuracy Evaluation

The first evaluation objective focuses on the accuracy of the object detection frameworks Faster R-CNN and MTCNN with regard to the respective datasets and their ground truth annotations. As stated in the introduction to this chapter and in section 2.4, the main metric for accuracy in object detection is Average Precision and its variations (Ren et al., 2015; Padilla et al., 2020). By far, the most extensive evaluation metrics for Python are provided by the COCO API when evaluating detections on the COCO dataset.[8] Furthermore, COCO evaluation also covers among other metrics also the main metric employed by the PASCAL VOC challenge and WIDER Face benchmark (Everingham et al., 2010; Yang et al., 2016). Therefore, it is reasonable to design the evaluation pipeline based on the COCO evaluation implementation provided by pycocotools and

[8] COCO Validation: https://cocodataset.org/#detection-eval

Torchvision [9] [10] In this section, the general evaluation pipeline is explained by inspecting the COCO dataset, since it is implemented for that purpose. However, multiple customizations have to be applied to the original implementation to fit this thesis's experimental requirements. Second, the specific implementation adjustments for PASCAL VOC and WIDER Face are stated to produce the same metrics for all three datasets. The prediction pipeline is closely linked to the chapter 4 and section 5.1 with the adjustments made in that regard.

COCO Evaluation

The evaluation pipeline for an object detector using the COCO dataset can be considered as a backward integration of the prediction pipeline by unifying the process of predicting and evaluating sequentially in a batched fashion. This process relies on multiple Python scripts from Torchvision and the site-package pycocotools.[10] [9] While pycocotools can be installed as a standard package, the necessary files from Torchvision have to be downloaded manually and stored on the remote server. Before starting the evaluation itself, the data and the model need to be prepared. This preparation incorporates the steps mentioned in section 5.1 about the customized dataset classes and procedure mentioned in section 5.2 regarding framework-specific NMS configuration. The transformational parameters have to be defined and passed to the customized dataset class, which is then loaded by a PyTorch data loader. At the same time, the model is prepared by accessing a pre-trained version and passing the NMS configuration as a parameter. For this specific example, all default values are chosen, which means no affine transformation is applied, and the choice of NMS is the classic greedy approach.

The starting point of the prediction/evaluation process starts by converting the annotations of the dataset to a COCO-specific format to extract the ground truth bounding boxes from the data loader. Based on the COCO-specific formatted annotations, an instance of the class `COCOEvaluator` is created. The `COCOEvaluator` class can be simultaneously interpreted as the API connection to pycocotools, the registry of the evaluation results. By instantiating the class, the ground truth annotations are loaded into the registry as an instance of the `COCOeval` class from pycocotools. At this point, it is essential to mention that the ground truth annotations are not extracted using the `__getitem__()` method of the `COCODetection` class but rather in a nested fashion during the conversion to the COCO specific format. By skipping the `__getitem__()` method of the dataset, the transformations elaborated in chapter 4 do not affect the ground truth annotations. This would lead to the unilateral transformation of only the images and not the annotations as displayed in figure 5.1. In order to prevent this case, the same affine transformation process is implemented into the `COCOEvaluator` class as well. By customizing this class, the respective transformational parameters have to be embedded into the constructor.

The next step is to iterate over the data and extract each time an image and annotation pair. The image is then passed to the model, predicting bounding boxes, confidence scores, and class labels. The `COCOEvaluator` class gets then updated with the output, evaluates the predictions concerning the ground truth, and complements the registry with the results. This update process triggers multiple functions in the pycocotools API, where the predictions are sorted based on the confidence score and then matched to the ground truth using ten different IoU levels between predicted bounding boxes and annotated ground truth bounding boxes as introduced in section 2.4. As soon as the last image and annotation pair is processed and the `COCOEvaluator` registry is updated, the loop is terminated and the results accumulated. The accumulation process captures the evaluation results of the individual images and calculates the 12 COCO-specific metrics elaborated in section 2.4. After the accumulation is done, the evaluator summarizes the accumulated

[9] `pycocotools on Github:` https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/cocoeval.py

[10] `Torchvision object detection reference on Github:` https://github.com/pytorch/vision/tree/master/references/detection

results and prints out the 12 metrics regarding the predicted bounding boxes, confidence scores, and labels as depicted in listing 5.1.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.370
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.581
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.400
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.212
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.401
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.482
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.310
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.491
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.515
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.326
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.550
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.648
```

Listing 5.1: COCO metrics to the baseline settings

In summary, two changes need to be implemented to adjust the COCO evaluation pipeline to the experimental setup. First, the `evaluate()` function is re-implemented in the prediction/evaluation pipeline with additional parameters. These parameters ensure consistent transformation throughout the prediction and evaluation pipeline. Additionally, the `CocoDetection` class has to accept these transformational parameters and perform the transformation accordingly before building up the registry for the ground truth annotations. The sole reason for these adjustments is because the COCO dataset is closely integrated into the evaluation pipeline provided by PyTorch and pycocotools.[10] [9]

In light of the holistic view on the prediction and evaluation pipeline for different NMS and affine transformations, there are 64 combinations to execute the pipeline for the COCO dataset. As pointed out in section 5.2, there are two distinct executions of NMS in the Faster R-CNN. One function call in the RPN head and the other one in the RoI head. By implementing the Multi-NMS function mentioned in 3, which features the selection of four different methods. Therefore, there are $4^2 = 16$ possible NMS permutations with the methods for RPN and RoI heads. When also considering the four different affine transformations of the single frames, all these 16 NMS scenarios have to be selected for each affine transformation. This results in $16 * 4 = 64$ executions of the prediction and evaluation pipeline for COCO, which also produces 64 result tables as displayed in listing 5.1. Up to the evaluation, all the necessary information about the matching predictions and ground truth annotations are stored in the `COCOEvaluator` registry.

PASCAL VOC Evaluation

The evaluation pipeline for predictions based on the PASCAL VOC dataset is closer to the original implementation of Torchvision and pycocotools than the evaluation pipeline for the COCO dataset. Therefore, fewer changes need to be adopted. However, the data and model preparation process before executing the detection/evaluation pipeline has to be performed the same way as with the COCO dataset. The main difference can be observed when converting the PASCAL VOC ground truth annotations to the COCO format. Because the PASCAL VOC dataset class is not as closely integrated into the COCO evaluation pipeline as it is the case for the COCO dataset, the only way to extract the ground truth annotations is by calling the `__getitem__()` method. Furthermore, this method has been customized from the version PyTorch provides, as explained in section 5.1. Thereby, the necessary transformations are applied to both the images and annotations simultaneously and prepared in a COCO-compatible format. These ground truth annotations are then used to populate the registry by instantiation the `COCOEvaluator` class. The

images are used as described above to produce predictions, which are then compared with the ground truth via the pycocotools API.[9]

Since PASCAL VOC and COCO datasets use the Faster R-CNN, the holistic prediction and evaluation pipeline behaves similarly regarding the production of detections and their evaluation. Hence, there are also the two locations of RPN and RoI head to replace NMS with Multi-NMS. Each of these permutations predicts bounding boxes with one of four possible affine transformations. Therefore, the PASCAL VOC prediction and evaluation pipeline also gets executed 64 times, producing 64 accuracy metrics after the standards of the COCO metrics. Since the COCO metrics are applied to the results on the PASCAL VOC dataset, the `COCOEvaluator` registry also holds the information about the ground truth and detection matches during evaluation.

WIDER Face Evaluation

The WIDER Face evaluation pipeline lies in between the number of changes needed to adopt the COCO and PASCAL VOC evaluation pipelines. With regard to data preparation, the pipeline can be set up in the same way as described for COCO. Comparable to PASCAL VOC, the data from the WIDER Face dataset gets extracted via the `__getitem__()` method and therefore transformed accordingly when converted to the COCO format. This is also due to the adjustments in the WIDER Face dataset discussed in section 5.1. However, the model preparation is performed differently. With MTCNN, only the three stages with their parameters for score thresholds and image pyramid calculation are instantiated. The prediction pipeline is performed by calling a function that orchestrates the forward passes of the three nets and the respective post-processing functionalities. Therefore, each of the three stages receives their respective NMS method by passing them as parameters into the prediction pipeline and not during instantiation of the model as with Faster R-CNN.

By instantiating the `COCODetection` class, the registry is created with the WIDER Face ground truth annotations converted to COCO format. Similar to the COCO evaluation, image and annotation pairs are extracted from the WIDER Face data loader. The images are then passed to the MTCNN prediction pipeline together with the desired NMS parameters. The predicted output of the MTCNN framework differs substantially from the expected output in the `evaluate()` function. By manually converting the outcome of the prediction pipeline into a format that the `COCODetection` class accepts, it can be ensured that the registry is updated accordingly. After all images and annotations are analyzed and the results stored in the registry, the accumulation and summarization of the COCO metrics behave similarly to the COCO evaluation.

Concerning the holistic prediction and evaluation pipeline of the MTCNN framework, there are more executions necessary to cover the same experimental reach as described in the Faster R-CNN. Since the MTCNN framework features three stages with a possibility to exchange the NMS function with Multi-NMS, introduced in chapter 3, there are $4^3 = 64$ permutations. It is also desired to perform different experiments by changing these NMS methods' order in the stages. Each of the 64 NMS permutations needs to be run with all four affine transformations to construct the frame sequence, which results in $64 * 4 = 256$ total experimental runs to evaluate the predictions in terms of their accuracy.

5.3.2 Stability Evaluation

As introduced in chapter 4 and section 2.4, there is a complementary perspective to the evaluation of predicted bounding boxes regarding their associated ground truth. The bounding box stability is a metric to summarize the error of detections along a trajectory. Since the analyzed datasets of this thesis only contain still images, the trajectory with respect to the time is one-dimensional. However, as seen in chapter 4, affine transformations to the datasets create four different versions

of each image. Initially, the default image is contained in the dataset, translation to the right by 20 pixels, translation to the bottom by 20 pixels, and a rotation of 5 degrees. By obtaining these different versions of the datasets, each version can be interpreted as a frame–these frames can be grouped and measured towards the second goal of this chapter.

The stability metric Φ by Zhang and Wang (2016) introduced in section 2.4 is the sum of the components E_F, E_C and E_R. Each of these components is dependent on the time dimension in terms of the length of trajectories in frames. E_F measures the fragment error with status changes throughout a sequence. The number of trajectories in a video sequence can be interpreted as the number of objects detected in the dataset. Moreover, the trajectory length is for the experiments ideally 4, for the four different affine transformations.

On the other hand, the two spatial components of the stability metric E_C and E_R are independent of the absolute position of the detection in the image and only measure its relative position to the ground truth annotation. However, they measure the standard deviation of the errors over the number of frames. If the time dimension is missing, the standard deviation of a single frame cannot be computed. Therefore, the crucial part of linking the stability metric of Zhang and Wang (2016) to this thesis is to build a sequence of 4 frames by sequentially matching the transformed detections and annotations of each image.

In order to analyze four augmented still images of the datasets sequentially, it can be continued where the evaluation of accuracy ended. The prediction and evaluation pipeline has produced 64 individual results for each COCO and PASCAL VOC dataset. Additionally, for MTCNN, there have been 256 results. These results of the respective framework's accuracy can directly be fed into the stability evaluation using the detection and ground truth matching from the `COCODetection` registry. This has the advantage that the accuracy and stability evaluation is performed on the same basis of matches between detections and ground truths. In order to create a sequence of detections predicted by one model, the four equally configured NMS experiments, each with another different affine transformed dataset, have to be grouped. Combining the same NMS configurations leaves 16 different models to analyze for each of COCO and PASCAL VOC and 64 for WIDER Face predictions.

The four frames are constructed by accessing the grouped results for each NMS configuration from the accuracy evaluation. The detections and annotations are looked up in the respective result files for that specific image when iterating over the dataset. This loop differs between MTCNN and Faster R-CNN results. Since Faster R-CNN is used for multi-class predictions, the results are stored for each class in a separate entry. However, individual classes in a multi-class setting can be handled separately since they relate to a single trajectory. If more than one detection per frame exists, the order is crucial since each detection refers to one trajectory. The construction of the sequence starts with the selection of the default dataset transformation and its result. The second element of the sequence consists of the results of the translated dataset transformation in the positive x-direction by $t_x = 20$. To check if the order of these results corresponds to the order of the default result, the IoU between the detected boxes in default transformation and the inverse transformation ($\mathbf{T_X^{-1}}$) of the t_x translated results are calculated. The inverse transformations of the translations in equation 4.4 have the form

$$\mathbf{T_X^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -20 & 0 & 1 \end{bmatrix}, \mathbf{T_Y^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -20 & 1 \end{bmatrix}. \tag{5.2}$$

Similarly, the inverse transformation of the rotation matrix $\mathbf{T_R}$ has the form

$$\mathbf{T_R^{-1}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.3}$$

The inverse transformation of the t_x translation is done by subtracting 20 pixels from the horizontal dimension of the bounding box coordinates, which is ($x_{min} - 20$ and $x_{max} - 20$. The same procedure is applied to the t_y translated setting as well with $y_{min} - 20$ and $y_{max} - 20$. The detections resulted from rotated images can not simply be inversely transformed back by subtraction since the image shape. With that, the image center coordinates have changed according to equation 5.1. In order to apply the inverse transformation, the same transformation as described in chapter 4 in general and in the appendix A.2 in detail, is reused. By applying the first rotation with 5 degrees and fitting the output as displayed in figure 5.2, the image shape changed. When then rotating the same image by -5 degrees, the padding triangles created to fit the output during the first rotation are treated as part of the image and, therefore, make the original image smaller in relative perspective. If this inverse transformation is applied to the detected and ground truth boxes, then they are adjusted to the decreased size of the rotated (+5 degrees), padded, and rotated (-5 degrees) image. The image shape after the first rotation is divided by the original image shape to overcome this problem. The resulting scale can factorize the respective dimensions of the inverse translated image to obtain the same size. The bounding boxes are then divided by this factor to be adjusted to the original size of the image. With the inverse transformation of each frame, the detected and ground truth bounding boxes are comparable between frames. Therefore, it is possible to ensure that the trajectory between the four frames refers to the same detection and ground truth annotation.

If there are differing numbers of detections between the frames, the order of the bounding boxes might have to be adjusted to maintain a consistent trajectory. Therefore, the same inter-frame IoU is used to calculate the best and worst overlap of the inverse transformed detections and annotations. It has to be determined how many unique objects/faces are located within these four frames. The number of unique objects dictates how many trajectories should be present in this particular sequence. The detections corresponding to the frames with less than the number of unique detections have to be padded to match this number. The padding takes place in order to achieve the same dimensions of detections for streamlined stability determination. By considering the calculation of IoU scores between the frame f_1, containing the detections of the original image, and f_2 containing the detections of a transformed image with $\mathbf{T_R}$. When calculating IoU scores between the two frames f_1 and f_2 with the corresponding detected bounding boxes $B_p^{f_1} \in [0, 256]^{m \times 4}$ and the inverse transformed bounding boxes $(B_p^{f_2}) \in [0, 256]^{n \times 4}$ with $\mathbf{T_R^{-1}}$ applied and with $m < n$. The resulting IoU scores $IoU(B^{f_1}, B^{f_2})$ have the shape $m \times n$. By selecting the maximum values and indices of the IoU matrix in the m dimension, the n best IoU scores and their index can be retrieved. Since $m < n$ the smallest $n - m$ IoU scores are close to 0. This is because the $(B_p^{f_2})$ contains more objects than $B_p^{f_1}$ and when comparing the m to n objects, only n have a strong overlap. This is only the case for this particular setting. Since the bounding boxes of two almost identical frames are compared (after inverse transformation), it can be assumed that the IoUs of the present boxes are close to 1. On the other hand, it can also be assumed that in the case of mismatching dimensions, the $n - m$ smallest IoUs are missing bounding boxes of $B_p^{f_1}$. Therefore, by looking up the associated indices, the padding of size $(n - m) \times 4$ can be inserted into $B_p^{f_1}$. The padded bounding boxes have the form $B_{padding}^{k,f} = (0, 0, 0, 0)$. This example of inter-frame IoU comparison is performed between all frames of the sequence to match the dimension of the unique objects in the sequence.

After the reordering and padding of the detected and ground truth bounding boxes, the actual accuracy metrics can be calculated for each trajectory separately. The predicted B_p and ground

truth bounding boxes B_g now have the form of

$$B_p = \begin{bmatrix} B_p^{f_1,k_1} & B_p^{f_2,k_1} & B_p^{f_3,k_1} & B_p^{f_4,k_1} \\ B_p^{f_1,k_2} & B_p^{f_2,k_2} & B_p^{f_3,k_2} & B_p^{f_4,k_2} \\ \vdots & \vdots & \vdots & \vdots \\ B_p^{f_1,k_m} & B_p^{f_2,k_m} & B_p^{f_3,k_m} & B_p^{f_4,k_m} \end{bmatrix}, B_g = \begin{bmatrix} B_g^{f_1,k_1} & B_g^{f_2,k_1} & B_g^{f_3,k_1} & B_g^{f_4,k_1} \\ B_g^{f_1,k_2} & B_g^{f_2,k_2} & B_g^{f_3,k_2} & B_g^{f_4,k_2} \\ \vdots & \vdots & \vdots & \vdots \\ B_g^{f_1,k_m} & B_g^{f_2,k_m} & B_g^{f_3,k_m} & B_g^{f_4,k_m} \end{bmatrix}. \quad (5.4)$$

Each of the bounding boxes have the form $B_p^{k,f} = (x_{minp}^{k,f}, y_{minp}^{k,f}, x_{maxp}^{k,f}, y_{maxp}^{k,f})$ and $B_g^{k,f} = (x_{ming}^{k,f}, y_{ming}^{k,f}, x_{maxg}^{k,f}, y_{maxg}^{k,f})$. The first component of the stability metric is the fragment error E_F, which measures the temporal stability by measuring the presence of detections over a trajectory. First, to calculate E_F from equation 2.7 the number of non-zero bounding boxes for each frame has to be extracted from B_p and B_g. By iterating over the frames, and then iterating over the detections, it can be looked at each detection if it is present in B_d. This results in the number of changes per detection for all frames. In reference to 2.7, the number of changes is divided by $t_k - 1$. Since the maximum length of a trajectory is equal to the presence in all frames, t_k is equal to 4. If a detection is present in frames f_1 and f_3 and not present in f_2 and f_4, the number of changes is 3 and dividing it by $4 - 1$, results in the highest possible value of 1 for E_F for one trajectory. On the other hand, if a detection consists of 4 padded bounding boxes or 4 detected bounding boxes, there are no status changes and E_F results in 0 for that particular trajectory. All fragment errors are added up for each image in the dataset. Also the maximal number of possible detections of each image with regard to that specific class is added up to form N. When all fragment errors over all images and all its trajectories are added up, the fragment error gets averaged by the total number of trajectories N. E_F is only calculated based on the bounding boxes of B_p, since B_g mirrors the number of bounding boxes in terms of the ground truth.

The second component of the stability metric is the center position error E_C and measures the spatial stability of a trajectory by the standard deviation from the center position. E_C consists of σ_x^k and σ_y^k, which themselves are the standard deviation of $e_x^{k,f}$ and $e_y^{k,f}$. To calculate the error in center position on the x axis $e_x^{k,f}$ and the error in center position on the y axis $e_y^{k,f}$ of the detections, the ground truth annotations are needed and both set of bounding boxes need to be converted first. By converting the bounding boxes, the form of $B_p^{k,f} = (x_p^{k,f}, y_p^{k,f}, w_p^{k,f}, h_p^{k,f})$ and $B_g^{k,f} = (x_g^{k,f}, y_g^{k,f}, w_g^{k,f}, h_g^{k,f})$ is obtained. Then $e_x^{k,f}$ and $e_y^{k,f}$ are calculated according to equation 2.8 for each detection in each frame. For each trajectory all the frames are combined and the standard deviations of e_x^k and e_y^k are computed. Before calculating the standard deviation, the padded bounding boxes are filtered out.

For the last component of the stability metric, also measures the spatial stability of a trajectory. The bounding boxes are converted to only consist of their width and height. The calculation of E_R follows the the procedure of equation 2.9 by first computing $e_s^{k,f}$ and $e_r^{k,f}$ for each frame and each detection. Then $e_s^{k,f}$ and $e_r^{k,f}$ are combined for all 4 frames of the trajectory and the standard deviation is computed. The padded bounding boxes are filtered for this process as well since these boxes are not produced by the framework and added manually. Additionally, the calcuation of $e_s^{k,f}$ and $e_r^{k,f}$ with bounding boxes of width and height 0 would result in a division by 0.

All components of the stability metric are calculated per image and then summed up at the end. After the fragment error, σ_x^k and σ_y^k, σ_s^k and σ_r^k are calculated for the last image, also the total number of possible trajectories N is aggregated. Then E_F, E_C and E_R can be determined by averaging the fragment error, the sum of σ_x^k and σ_y^k and the sum of σ_s^k and σ_r^k by N. The stability Φ is then the sum of E_F, E_C and E_R.

There is one essential adjustment to the initially proposed stability metric by Zhang and Wang (2016). The authors draw the stability error Φ over all IoU overlaps between $[0, 1]$ and calculate the area under the curve as the final metric. Since the stability metric in this thesis taps in on the `COCODetection` registries to obtain the exact detection and ground truth matches as for the accu-

racy evaluation, the lowest possible IoU is determined by the COCO evaluation metric, which is 0.5.

The overall stability evaluation combines the accuracy evaluation of each model NMS-composition in terms of the four different dataset transformations. Each transformation is considered a frame and then sequentially combined to evaluate the stability of detections made on transformed images. When measuring the bounding box stability by comparing inter-frame detections with the transformed ground truth annotations, it can be analyzed how different frameworks with different NMS methods perform on moving objects in the sequence of frames proposed in chapter 4.

Chapter 6

Results

Within this chapter the results are reported from the combination of proposed Non-Maximum Suppression substitutions (chapter 3) and stability evaluation (chapter 4), together with already well established accuracy evaluation (section 2.4) guided by the experimental framework (chapter 5). The foundation built up throughout the entire thesis and specified in the experimental framework leads two a two-part evaluation outcome. The four different Non-Maximum Suppression methods are first evaluated for accuracy in both detection frameworks with the respective datasets. Since the frameworks are multi-stage detectors, there are multiple combinations of possible framework compositions to configure the frameworks. Due to the methodological design, each NMS composition is evaluated for accuracy for each of the four augmented frames. The second evaluation component, the stability evaluation, the accuracy evaluation results, and byproducts are grouped in an ordered sequence. Especially the sequential combination of the accuracy evaluation outcome in the form of matching prediction bounding boxes and ground truth bounding boxes are further analyzed and in the following reported.

6.1 Accuracy

The primary goal for the first part is to utilize established accuracy metrics to compare differently composed frameworks with each other. Therefore, classic greedy and Soft-NMS have been reimplemented in the Multi-Non-Maximum Suppression function to establish a benchmark. The two newly proposed methods of Average NMS and Average IoU NMS are expected to have an accuracy performance close to the benchmark of the reused NMS methods. The primary metrics, therefore, are the COCO-style mean Average Precision (stated as AP) and the PASCAL VOC-style mean Average Precision (stated as $AP^{0.5}$). The difference between these two is stated in detail in section 2.4–the accuracy evaluation as opposed to the stability evaluation performed on still images. However, the single frames of the sequences used for the stability evaluation can still be considered independent images in an augmented version. Therefore, the accuracy evaluation is performed on every single frame of the sequence. Hence, there are four accuracy evaluations for each NMS composition of each framework with its specific dataset(s). An example of such an NMS composition for Faster R-CNN is the usage of classic greedy NMS (Classic) in the Region Proposal Network (RPN) and Average IoU NMS (Avg. IoU) in the Region of Interest Head (RoI). This results in 64 accuracy evaluations for the COCO dataset and 64 accuracy evaluations for the PASCAL VOC dataset, both with Faster R-CNN. The results of these accuracy evaluations are stated in tables 6.1 and 6.2. In light of the upcoming combination of the individual frames of a specific framework composition in the stability evaluation, the individual accuracies are already grouped by these compositions. Therefore tables 6.1 and 6.2 each have 16 rows, corresponding to these NMS compositions in columns NMS RPN and NMS RoI. To indicate the overall accuracy

NMS RPN	NMS RoI	AP	$AP^{0.5}$	AP_{f_1}	$AP^{50}_{f_1}$	AP_{f_2}	$AP^{50}_{f_2}$	AP_{f_3}	$AP^{50}_{f_3}$	AP_{f_4}	$AP^{50}_{f_4}$
Classic	Classic	34.58	**57.03**	37.0	**58.1**	36.1	**57.4**	36.5	**57.7**	28.7	**54.9**
Classic	Soft	**34.78**	56.53	**37.2**	57.6	**36.3**	56.9	**36.7**	57.3	**28.9**	54.3
Classic	Average	32.23	55.23	34.2	56.2	33.5	55.5	33.8	55.9	27.4	53.3
Classic	Avg. IoU	34.28	56.53	36.9	58.0	36.1	57.3	36.5	**57.7**	27.6	53.1
Soft	Classic	34.20	56.58	36.5	57.5	35.7	56.9	36.2	57.4	28.4	54.5
Soft	Soft	34.33	55.95	36.7	57.0	35.8	56.2	36.3	56.8	28.5	53.8
Soft	Average	32.10	54.70	34.2	55.6	33.4	55.0	33.8	55.3	27.0	52.9
Soft	Avg. IoU	32.15	54.70	34.2	54.9	33.4	54.3	33.9	54.7	27.1	52.5
Average	Classic	34.60	56.98	37.0	**58.1**	36.1	57.3	36.5	**57.7**	28.8	54.8
Average	Soft	34.73	56.50	37.1	57.5	**36.3**	56.9	36.6	57.3	**28.9**	54.3
Average	Average	32.23	55.23	34.2	56.2	33.5	55.5	33.8	55.9	27.4	53.3
Average	Avg. IoU	32.38	54.75	34.3	55.6	33.7	55.0	34.0	55.4	27.5	53.0
Avg. IoU	Classic	34.58	56.95	36.9	58.0	36.1	57.3	36.5	**57.7**	28.8	54.8
Avg. IoU	Soft	34.70	56.45	37.0	57.5	**36.3**	56.9	36.6	57.2	**28.9**	54.2
Avg. IoU	Average	32.08	55.10	34.0	56.0	33.4	55.4	33.6	55.7	27.3	53.3
Avg. IoU	Avg. IoU	32.38	54.75	34.3	55.6	33.7	55.0	34.0	55.4	27.5	53.0

Table 6.1: COCO ACCURACY EVALUATION RESULTS. The accuracy evaluation results for the COCO dataset with Faster R-CNN. NMS RPN and NMS RoI stand for the Non-Maximum Suppression method used in the Faster R-CNN. AP is the main metric for the COCO evaluation and is measured as the mean Average Precision over ten IoU thresholds. $AP^{0.5}$ is the main metric for the PASCAL VOC and WIDER Face evaluation and is measured as the mean Average Precision for an IoU threshold of 0.5. Each of these metrics is measured for every frame in $\{f_1, f_2, f_3, f_4\}$. Therefore, the first pair of AP and $AP^{0.5}$ are calculated by averaging all AP and $AP^{0.5}$ over all four frames in the sequence. The bold numbers indicate the best scoring NMS composition for each metric.

over the four frames in a sequence ($\{f_1, f_2, f_3, f_4\}$), the frame specific AP_{f_n} and $AP^{50}_{f_n}$ for $n \in \{1, 2, 3, 4\}$ are averaged and stated in columns AP and $AP^{0.5}$.

In table 6.1, it can be observed that for COCO, the NMS composition consisting of the combination of both benchmarking NMS methods outperform all other framework compositions in terms of overall average AP and $AP^{0.5}$. These high numbers result from a consistent performance in terms of AP for the combination Classic/Soft and in terms of $AP^{0.5}$ for the combination Classic Classic over all frames. However, Average/Soft and Average IoU/Soft combinations perform nearly as good as Classic/Soft in terms of AP over ten IoU thresholds (AP). There is no distinct combination that outperforms all other combinations in AP and $AP^{0.5}$. It can also be observed that superior performance in both AP and $AP^{0.5}$ for the COCO dataset is mutually exclusive. Combinations either achieve a top score for AP or a top score for $AP^{0.5}$. Therefore, it is informative to evaluate the accuracy of Faster R-CNN on the COCO dataset with both Average Precision variations.

Table 6.2 shows the accuracy evaluation results for Faster R-CNN on the PASCAL VOC dataset. Again the combination of benchmark NMS methods Classic/Soft has a high accuracy performance. However, replacing classic greedy NMS in the RPN head with Average IoU NMS, the performance peaks regarding all measured Average Precisions. The combination Average IoU/-Soft directly disproves the observation made in the COCO accuracy evaluation results, that top performance in both AP and $AP^{0.5}$ are mutually exclusive. This combination achieves top performance over all frames in both AP and $AP^{0.5}$. This also results in superior performance in overall AP and $AP^{0.5}$. It is also notable that the order of NMS employment in the Faster R-CNN matters. If Soft/Average IoU is used instead of Average IoU/Soft, the accuracy performance drops to the worst of all measured performances.

In addition to the NMS composition possibilities in Faster R-CNN, MTCNN employs NMS

NMS RPN	NMS RoI	AP	$AP^{0.5}$	AP_{f_1}	$AP^{50}_{f_1}$	AP_{f_2}	$AP^{50}_{f_2}$	AP_{f_3}	$AP^{50}_{f_3}$	AP_{f_4}	$AP^{50}_{f_4}$
Classic	Classic	43.33	79.20	45.50	79.80	44.40	79.00	45.00	79.20	38.40	78.80
Classic	Soft	44.35	80.30	**46.50**	81.00	**45.50**	80.30	46.10	80.20	39.30	79.70
Classic	Average	41.28	76.93	42.40	77.10	41.80	76.70	42.00	76.70	38.90	77.20
Classic	Avg. IoU	41.60	76.00	42.70	76.10	42.30	75.90	42.30	75.70	39.10	76.30
Soft	Classic	41.85	77.95	43.80	78.60	43.00	77.70	43.40	77.90	37.20	77.60
Soft	Soft	42.73	78.48	44.70	79.30	43.90	78.30	44.20	78.30	38.10	78.00
Soft	Average	40.93	77.03	41.90	77.30	41.30	76.60	41.60	76.90	38.90	77.30
Soft	Avg. IoU	38.73	72.05	40.00	72.30	39.40	71.90	39.70	71.80	35.80	72.20
Average	Classic	43.33	79.38	45.30	79.90	44.40	79.10	45.20	79.50	38.40	79.00
Average	Soft	44.03	80.40	46.00	**81.10**	45.00	80.30	45.80	80.30	39.30	**79.90**
Average	Average	41.43	77.13	42.50	77.40	41.90	76.70	42.10	77.00	39.20	77.40
Average	Avg. IoU	41.23	76.13	42.20	76.30	41.70	75.80	42.00	76.10	39.00	76.30
Avg. IoU	Classic	43.38	79.40	45.40	79.90	44.50	79.10	45.20	79.60	38.40	79.00
Avg. IoU	Soft	**44.40**	**80.48**	**46.50**	**81.10**	**45.50**	**80.40**	**46.20**	**80.50**	**39.40**	**79.90**
Avg. IoU	Average	39.05	73.93	40.30	74.20	39.60	73.80	40.00	73.70	36.30	74.00
Avg. IoU	Avg. IoU	41.70	76.18	42.80	76.30	42.30	75.90	42.60	76.10	39.10	76.40

Table 6.2: PASCAL VOC ACCURACY EVALUATION RESULTS. The accuracy evaluation results for the PASCAL VOC dataset with Faster R-CNN. NMS RPN and NMS RoI stand for the Non-Maximum Suppression method used in the Faster R-CNN. AP is the main metric for the COCO evaluation and is measured as the mean Average Precision over ten IoU thresholds. $AP^{0.5}$ is the main metric for the PASCAL VOC and WIDER Face evaluation and is measured as the mean Average Precision for an IoU threshold of 0.5. Each of these metrics is measured for every frame in $\{f_1, f_2, f_3, f_4\}$. Therefore, the first pair of AP and $AP^{0.5}$ are calculated by averaging all AP and $AP^{0.5}$ over all four frames in the sequence. The bold numbers indicate the best scoring NMS composition for each metric.

in three stages. An example of such an NMS composition for MTCNN is the usage of Average NMS (Average) in the proposal network (P-Net), classic greedy NMS (Classic) in the refinement network (R-Net), and Average IoU NMS (Avg. IoU) in the output network (O-Net). Therefore, there are 64 different combinations of NMS to be deployed in all three stages. By evaluating all individual frames in terms of accuracy for all compositions, the MTCNN evaluation on the WIDER Face dataset comprises 256 accuracy outcomes. For the sake of overview, the very low performing AP and $AP^{0.5}$ combinations are filtered out in table 6.3. The entire table for the accuracy evaluation with MTCNN on WIDER Face is depicted in appendix D. By scanning through table 6.3 it is evident that every combination with Soft-NMS is missing. Especially since Soft-NMS was part of every top-performing combination in the COCO and PASCAL VOC result tables. The missing Soft-NMS entries in this result table are due to the consistent low performance of the NMS method in MTCNN. Possible reasons, therefore, are stated in the following chapter when these results are interpreted.

Nevertheless, the benchmark combination with classic greedy NMS in all three stages is among the top-performing MTCNN compositions in terms of accuracy. The benchmark method classic greedy NMS is part of every top scoring combination, especially when combined with Average IoU NMS. The combination Average IoU/Average IoU/Classic performs the best concerning AP and $AP^{0.5}$. The high score in AP is due to the best accuracies in the third and fourth frames. Interestingly, in the fourth frame, which represents the rotated images, this combination achieves an accuracy higher than for both translated images. This score is atypical when comparing the overall performance to the fourth frame in both Faster R-CNN evaluations.

NMS P-Net	NMS R-Net	NMS O-Net	AP	$AP^{0.5}$	AP_{f_1}	$AP^{50}_{f_1}$	AP_{f_2}	$AP^{50}_{f_2}$	AP_{f_3}	$AP^{50}_{f_3}$	AP_{f_4}	$AP^{50}_{f_4}$
Classic	Classic	Classic	26.73	47.35	28.00	47.40	22.60	47.20	28.20	**47.40**	28.10	47.40
Classic	Classic	Average	26.48	46.80	28.00	46.80	21.80	46.80	28.00	46.80	28.10	46.80
Classic	Classic	Avg. IoU	24.65	44.58	26.00	44.60	20.50	44.50	26.10	44.60	26.00	44.60
Classic	Average	Classic	26.83	47.33	28.00	47.30	22.70	47.20	**28.50**	**47.40**	28.10	47.40
Classic	Average	Average	26.55	46.80	28.00	46.80	21.90	46.70	28.10	46.80	28.20	46.90
Classic	Average	Avg. IoU	24.63	44.48	26.00	44.50	20.50	44.50	26.00	44.50	26.00	44.40
Classic	Avg. IoU	Classic	26.80	47.35	28.10	47.30	22.70	**47.30**	28.30	**47.40**	28.10	47.40
Classic	Avg. IoU	Average	26.58	46.95	28.10	47.50	21.90	46.70	28.10	46.80	28.20	46.80
Classic	Avg. IoU	Avg. IoU	24.60	44.63	26.00	45.20	20.40	44.40	26.00	44.50	26.00	44.40
Average	Classic	Classic	26.53	47.20	28.10	47.40	22.00	46.70	28.00	47.30	28.00	47.40
Average	Classic	Average	26.50	46.73	28.10	46.80	22.00	46.70	28.00	46.60	27.90	46.80
Average	Classic	Avg. IoU	24.78	44.73	26.30	44.60	20.60	44.70	26.20	44.80	26.00	44.80
Average	Average	Classic	26.83	47.33	28.30	47.40	22.70	47.20	28.10	**47.40**	28.20	47.30
Average	Average	Average	26.53	46.78	28.10	46.80	21.90	46.80	28.20	46.80	27.90	46.70
Average	Average	Avg. IoU	24.60	44.58	25.90	44.30	20.50	44.50	26.20	44.90	25.80	44.60
Average	Avg. IoU	Classic	**26.88**	47.30	28.30	47.40	**22.80**	47.20	28.20	**47.40**	28.20	47.20
Average	Avg. IoU	Average	26.58	46.70	28.10	46.70	22.00	46.70	28.20	46.80	28.00	46.60
Average	Avg. IoU	Avg. IoU	24.70	44.50	25.90	44.20	20.70	44.60	26.20	44.70	26.00	44.50
Avg. IoU	Classic	Classic	26.78	47.35	28.20	**48.30**	22.40	46.40	28.30	**47.40**	28.20	47.30
Avg. IoU	Classic	Average	26.53	46.83	28.20	47.80	21.80	46.00	28.20	46.80	27.90	46.70
Avg. IoU	Classic	Avg. IoU	24.80	44.93	26.30	45.70	20.60	44.70	26.20	44.70	26.10	44.60
Avg. IoU	Average	Classic	26.83	47.33	28.30	48.20	22.50	46.40	28.30	**47.40**	28.20	47.30
Avg. IoU	Average	Average	26.60	46.95	28.30	47.70	22.00	46.70	28.20	46.80	27.90	46.60
Avg. IoU	Average	Avg. IoU	24.68	44.73	26.10	45.50	20.50	44.50	26.20	44.60	25.90	44.30
Avg. IoU	Avg. IoU	Classic	**26.88**	**47.73**	**28.40**	48.20	22.60	47.20	28.20	**47.40**	**28.30**	**48.10**
Avg. IoU	Avg. IoU	Average	26.63	47.15	28.30	47.70	22.00	46.80	28.10	46.70	28.10	47.40
Avg. IoU	Avg. IoU	Avg. IoU	24.68	44.90	26.10	45.40	20.60	44.60	26.00	44.50	26.00	45.10

Table 6.3: WIDER FACE ACCURACY EVALUATION RESULTS. The accuracy evaluation results for the WIDER Face dataset with MTCNN. NMS RPN and NMS RoI stand for the Non-Maximum Suppression method used in the MTCNN. AP is the main metric for the COCO evaluation and is measured as the mean Average Precision over ten IoU thresholds. $AP^{0.5}$ is the main metric for the PASCAL VOC and WIDER Face evaluation and is measured as the mean Average Precision for an IoU threshold of 0.5. Each of these metrics is measured for every frame in $\{f_1, f_2, f_3, f_4\}$. Therefore, the first pair of AP and $AP^{0.5}$ are calculated by averaging all AP and $AP^{0.5}$ over all four frames in the sequence. The bold numbers indicate the best scoring NMS composition for each metric.

6.2 Stability

The second part of this chapter focuses on the complementary evaluation perspective of this thesis – the stability of detections. The stability metric, in general, measures the temporal and spatial stability of detections. As proposed in chapter 4 and detailed on in section 5.3.2, the stability evaluation works on top of the accuracy evaluation. By tapping into the outcomes of each of the above-stated accuracy evaluations and their outcomes, it is possible to analyze the temporal and spatial stability of the detections to their ground truth annotations in ordered sequences. Each row of the above-stated result tables represents the individual accuracy performance on each frame of the sequence and the averaged accuracy performance on the whole sequence of frames. Therefore, the stability evaluation results have the same form by orienting at the different compositions of NMS methods for the respective frameworks and datasets. Each of the following tables reports first the framework compositions and the averaged accuracy evaluation metrics AP and $AP^{0.5}$. The column SE is the main evaluation metric for the stability evaluation, indicating the overall stability error. The stability error is the sum of the three columns of fragment error (FE), center position error (CPE), and scale and ratio error (CPE). To further indicate the number of total unique detection ground truth matches there have been, column N states the number of

NMS RPN	NMS RoI	AP	$AP^{0.5}$	SE	FE	CPE	SRE	N
Classic	Classic	34.58	**57.03**	0.605	0.415	0.127	0.062	4157
Classic	Soft	**34.78**	56.53	0.598	0.403	0.132	0.062	5047
Classic	Average	32.23	55.23	0.591	0.416	0.123	0.052	4153
Classic	Avg. IoU	34.28	56.53	0.596	0.415	0.124	0.058	4209
Soft	Classic	34.20	56.58	0.603	0.414	0.129	0.060	4305
Soft	Soft	34.33	55.95	0.599	0.404	0.132	0.062	**5234**
Soft	Average	32.10	54.70	0.482	0.296	**0.067**	0.119	4108
Soft	Avg. IoU	32.15	54.10	0.591	0.411	0.126	0.053	4291
Average	Classic	34.60	56.98	0.596	0.413	0.124	0.059	4198
Average	Soft	34.73	56.50	**0.478**	**0.281**	0.071	0.127	4854
Average	Average	32.23	55.23	0.591	0.416	0.123	0.052	4153
Average	Avg. IoU	32.38	54.75	0.488	0.299	0.068	0.120	4099
Avg. IoU	Classic	34.58	56.95	0.595	0.413	0.123	0.059	4203
Avg. IoU	Soft	34.70	56.45	0.598	0.403	0.132	0.062	5047
Avg. IoU	Average	32.08	55.10	0.489	0.301	0.068	0.120	4046
Avg. IoU	Avg. IoU	32.38	54.75	0.586	0.414	0.122	**0.050**	4223

Table 6.4: COCO STABILITY EVALUATION RESULTS. The stability evaluation results for the COCO dataset with Faster R-CNN. NMS RPN and NMS RoI stand for the Non-Maximum Suppression method used in the Faster R-CNN. For AP and $AP^{0.5}$ the averaged mean Average Precision over all frames in the sequence is computed. SE stands for stability error and is the sum of FE (fragment error), CPE (center position error) and SRE (scale and ratio error). N is the number of trajectories by which all of the individual errors are normalized with. The bold numbers indicate the best scoring NMS composition. The error terms are best when they are small.

trajectories for each NMS composition. The results for Faster R-CNN on the COCO dataset and on the PASCAL VOC dataset are depicted in tables 6.4 and 6.5 respectively. The results for MTCNN on the WIDER Face dataset are listed tabularly in table 6.6.

In contrast to the accuracy evaluation metrics, the stability evaluation metrics are the most stable if the errors are minor. Hence, the most stable NMS combination for Faster R-CNN on the COCO dataset in table 6.4 is Average/Soft. This combination also has the second-highest score regarding AP. This means that predictions made with Average/Soft are accurate and the most stable over the sequence of augmented images. This is mainly due to a relatively low fragment error, which indicates that the detections stay stable over the temporal dimension of the sequence. It can further be observed that the number of trajectories is the highest when Soft-NMS is applied in the RPN and RoI heads. Generally, when Soft-NMS is applied in the RoI head, the number of distinct detections is high. A large number of trajectories can benefit the fragment error by giving a possibility to match the detections over the four frames.

Nevertheless, it can also increase the stability error when these matched detections are too different in the center, scale, and ratio deviation from their ground truths. This can be seen in the combination of Soft/Soft, where N is the largest. Overall, the smallest errors can be identified in combinations with Average or Average IoU NMS.

The stability errors for Faster R-CNN on the PASCAL VOC dataset in table 6.5 are closer to each other than the SE in the COCO dataset stability evaluation. There is only one outlier with the combination of Average/Classic. Interestingly, this combination the other way around

NMS RPN	NMS RoI	AP	$AP^{0.5}$	SE	FE	CPE	SRE	N
Classic	Classic	43.33	79.20	0.452	0.263	0.071	0.118	3280
Classic	Soft	44.35	80.30	0.426	0.232	0.073	0.122	4423
Classic	Average	41.28	76.93	**0.409**	0.264	0.056	0.089	3280
Classic	Avg. IoU	41.60	76.00	**0.409**	0.264	0.056	0.089	3277
Soft	Classic	41.85	77.95	0.452	0.258	0.073	0.121	3562
Soft	Soft	42.73	78.48	0.430	0.226	0.077	0.128	4624
Soft	Average	40.93	77.03	0.417	0.260	0.061	0.097	3551
Soft	Avg. IoU	38.73	72.05	0.419	0.259	0.062	0.098	3552
Average	Classic	43.33	79.38	0.746	0.580	0.060	0.107	3171
Average	Soft	44.03	80.40	0.420	0.227	0.071	0.122	4532
Average	Average	41.43	77.13	0.452	0.307	0.055	0.090	3306
Average	Avg. IoU	41.23	76.13	0.452	0.306	0.055	0.090	3303
Avg. IoU	Classic	43.38	79.40	0.481	0.304	0.066	0.111	3316
Avg. IoU	Soft	**44.40**	**80.48**	0.418	**0.225**	0.072	0.122	**4530**
Avg. IoU	Average	39.05	73.93	0.447	0.305	**0.054**	**0.088**	3311
Avg. IoU	Avg. IoU	41.70	76.18	0.448	0.305	0.055	0.089	3307

Table 6.5: PASCAL VOC STABILITY EVALUATION RESULTS. The stability evaluation results for the VOC dataset with Faster R-CNN. NMS RPN and NMS RoI stand for the Non-Maximum Suppression method used in the Faster R-CNN. For AP and $AP^{0.5}$ the averaged mean Average Precision over all frames in the sequence is computed. SE stands for stability error and is the sum of FE (fragment error), CPE (center position error) and SRE (scale and ratio error). N is the number of trajectories by which all of the individual errors are normalized with. The bold numbers indicate the best scoring NMS composition. The error terms are best when they are small.

with Classic/Average shows the smallest stability error. This again emphasizes evaluating all the combinations of NMS methods in the RPN and RoI heads. Another combination with the smallest stability error is Classic/Average IoU. It can generally be observed that Average and Average IoU perform very similarly regardless of the second component of the composition. The combination of Average IoU/Soft, which scored the highest in terms of AP and $AP^{0.5}$ during the accuracy evaluation, also has a relatively low stability error.

Additionally, it shows the lowest fragment error. This is due to the consistent superior accuracies over all frames in the accuracy evaluation. The combination of Average IoU/Average in the stability evaluation of Faster R-CNN on the PASCAL VOC dataset results in the lowest center position error and scale and ratio error. However, since a high fragment error dominates the sum, the stability error is relatively large. It can again be observed that if Soft-NMS is placed as the NMS method in the RoI head, the number of trajectories is large.

The last table 6.6 summarizes the stability evaluation results for MTCNN on the WIDER Face dataset. This table is also a filtered version of the entire table (appendix D.1) with the same criterion of low scoring AP and $AP^{0.5}$ exclusion. The lowest error for MTCNN is the combination of Average NMS in the P-Net and the O-Net with classic greedy NMS in between in the R-Net. This is the result of low scale and ratio error and low center position error. Also, the fragment error for this combination is only 0.0001, larger than the minimum. The combination with the lowest fragment error in the stability evaluation also shows consistent accuracy rates in the accuracy evaluation above. Even though the accuracies are not performing top, the fragment error

NMS P-Net	NMS R-Net	NMS O-Net	AP	$AP^{0.5}$	SE	FE	CPE	SRE	N
Classic	Classic	Classic	26.73	47.35	0.1853	0.0639	0.0485	0.0729	2937
Classic	Classic	Average	26.48	46.80	0.1637	0.0637	0.0345	0.0654	2939
Classic	Classic	Avg. IoU	24.65	44.58	0.1696	0.0636	0.0378	0.0681	2938
Classic	Average	Classic	26.83	47.33	0.1895	0.0655	0.0502	0.0737	2940
Classic	Average	Average	26.55	46.80	0.1652	0.0647	0.0350	0.0655	2938
Classic	Average	Avg. IoU	24.63	44.48	0.1720	0.0647	0.0387	0.0686	2940
Classic	Avg. IoU	Classic	26.80	47.35	0.1901	0.0659	0.0505	0.0737	**2944**
Classic	Avg. IoU	Average	26.58	46.95	0.1660	0.0646	0.0353	0.0661	2942
Classic	Avg. IoU	Avg. IoU	24.60	44.63	0.1724	0.0647	0.0388	0.0689	2942
Average	Classic	Classic	26.53	47.20	0.1864	0.0646	0.0484	0.0733	2925
Average	Classic	Average	26.50	46.73	**0.1629**	0.0637	**0.0339**	**0.0653**	2925
Average	Classic	Avg. IoU	24.78	44.73	0.1696	**0.0636**	0.0375	0.0685	2924
Average	Average	Classic	26.83	47.33	0.1902	0.0667	0.0497	0.0738	2933
Average	Average	Average	26.53	46.78	0.1663	0.0659	0.0347	0.0656	2932
Average	Average	Avg. IoU	24.60	44.58	0.1730	0.0657	0.0386	0.0687	2933
Average	Avg. IoU	Classic	**26.88**	47.30	0.1916	0.0674	0.0502	0.0740	2939
Average	Avg. IoU	Average	26.58	46.70	0.1669	0.0663	0.0349	0.0657	2936
Average	Avg. IoU	Avg. IoU	24.70	44.50	0.1727	0.0655	0.0385	0.0687	2936
Avg. IoU	Classic	Classic	26.78	47.35	0.1893	0.0681	0.0485	0.0726	2932
Avg. IoU	Classic	Average	26.53	46.83	0.1662	0.0665	0.0343	0.0654	2932
Avg. IoU	Classic	Avg. IoU	24.80	44.93	0.1733	0.0666	0.0383	0.0684	2932
Avg. IoU	Average	Classic	26.83	47.33	0.1941	0.0700	0.0505	0.0736	2939
Avg. IoU	Average	Average	26.60	46.95	0.1698	0.0684	0.0353	0.0662	2939
Avg. IoU	Average	Avg. IoU	24.68	44.73	0.1762	0.0682	0.0391	0.0689	2939
Avg. IoU	Avg. IoU	Classic	**26.88**	**47.73**	0.1935	0.0683	0.0512	0.0741	2940
Avg. IoU	Avg. IoU	Average	26.63	47.15	0.1684	0.0666	0.0355	0.0663	2939
Avg. IoU	Avg. IoU	Avg. IoU	24.68	44.90	0.1752	0.0663	0.0395	0.0693	2939

Table 6.6: WIDER FACE STABILITY EVALUATION RESULTS. Stability evaluation results for the WIDER Face dataset with MTCNN. The Non-Maximum Suppression methods used to evaluate the MTCNN is stated in NMS RPN and NMS RoI. For AP and $AP^{0.5}$ the averaged mean Average Precision over all frames in the sequence is computed. SE stands for stability error and is the sum of FE (fragment error), CPE (center position error) and SRE (scale and ratio error). N is the number of trajectories by which all of the individual errors are normalized with. The bold numbers indicate the best scoring NMS composition. The error terms are best when they are small.

values consistency over the detections in the four frames. It can also be observed that the number of trajectories does not vary vastly. Between the maximum and minimum are only 20 distinct detections in the sequence. This contrasts with the other stability evaluation results, where mainly Soft-NMS drives the number of trajectories up.

However, concerning the first research question, the stability evaluation results listed so far do not yet allow any statistically meaningful conclusion to be drawn. For this reason, the stability errors are further investigated through a t-test. Although the stability errors are assumed to be normally distributed, the population sizes differ (column N in tables 6.4, 6.5 and 6.6. It can also be assumed that every two compared samples are independent within and between each other. In this case, two independent sample comparison of means test with unequal variances is applicable.[1]

In reference to the first research question, the hypothesis is formulated that the stability error (SE) of the Non-Maximum Suppression substitution (SE_S) is greater or equal to the stability error

[1] Definition Welch's t-test: https://en.wikipedia.org/wiki/Welch%27s_t-test

NMS RPN	NMS RoI	Classic/Classic	Classic/Soft	Soft/Classic	Soft/Soft
Classic	Average	3.35E-02	1.82E-01	6.04E-02	1.43E-01
Classic	Avg. IoU	1.31E-01	4.33E-01	2.01E-01	3.72E-01
Soft	Average	**1.52E-55**	**7.02E-54**	**2.26E-54**	**4.71E-55**
Soft	Avg. IoU	3.01E-02	1.69E-01	5.50E-02	1.32E-01
Average	Classic	1.08E-01	3.87E-01	1.71E-01	3.28E-01
Average	Soft	**8.14E-64**	**1.09E-62**	**1.30E-62**	**4.79E-64**
Average	Average	3.35E-02	1.82E-01	6.04E-02	1.43E-01
Average	Avg. IoU	**2.07E-48**	**1.63E-46**	**2.86E-47**	**1.39E-47**
Avg. IoU	Classic	9.36E-02	3.55E-01	1.50E-01	2.98E-01
Avg. IoU	Soft	1.59E-01	5.00E-01	2.41E-01	4.35E-01
Avg. IoU	Average	**1.17E-48**	**9.20E-47**	**1.61E-47**	**7.84E-48**
Avg. IoU	Avg. IoU	**6.30E-03**	5.49E-02	1.33E-02	3.92E-02

Table 6.7: COCO T-TEST. The resulting p-values of the one-tailed t-test for Faster R-CNN using the COCO dataset. Numbers in bold indicate NMS combinations significant on a 99% confidence level.

of the benchmarking Non-Maximum Suppression method (SE_B). This leads to the establishment of the null (H_0) and alternative hypothesis (H_A) in a one-tailed t-test:

- H_0: SE_B - $SE_S \leq 0$
- H_A: SE_B - $SE_S > 0$

The null hypothesis states that the difference between the benchmark and the substitution is at most 0, which means that the substituting NMS method has a stability error that is greater or equal compared to the stability error of the benchmarking NMS method. On the other hand, the alternative hypothesis states that the NMS substitution shows more minor stability errors than the NMS benchmark method. As the benchmarking NMS methods are the combination of classic greedy NMS and Soft-NMS considered. This is an extended benchmark since Soft-NMS is not available in the original implementation of Faster R-CNN as utilized for this thesis. The original implementation of Faster R-CNN only features classic greedy NMS in all stages. As Soft-NMS is already implemented by Bodla et al. (2017) and its results reported, it is considered as part of the extended benchmark for the first part of this analysis. The NMS substitution methods are every other combination of NMS methods in the frameworks. For this hypothesis test, the 99% confidence level is used.

The table 6.7 summarizes the results of the above-described t-test for Faster R-CNN using the COCO dataset. Every combination featuring at least one of the proposed NMS methods is compared to every combination of benchmark NMS methods in terms of their stability error difference. It can be seen that the stability error of Average IoU/Average IoU is significantly smaller than the benchmark. Applying Average IoU/Average IoU as NMS yields significantly higher stability than the original implementation of Faster R-CNN with the COCO dataset. The four combinations of Soft/Average, Average/Soft, Average/Average IoU, and Average IoU/Average show notably low p-values. This indicates that their stability error is significantly smaller than every combination of the benchmarks.

Table 6.8 shows the same comparison of NMS substitutions and NMS benchmarking methods with a t-test but this time for Faster R-CNN using the PASCAL VOC dataset. In this table, only one combination allows significantly higher stability than all of the benchmarking NMS methods. Using classic greedy NMS in the RPN head and Average NMS in the RoI head is significantly more

NMS RPN	NMS RoI	Classic/Classic	Classic/Soft	Soft/Classic	Soft/Soft
Classic	Average	**1.18E-08**	**7.25E-03**	**1.01E-08**	**1.55E-03**
Classic	Avg. IoU	**2.04E-08**	**9.58E-03**	**1.76E-08**	2.16E-03
Soft	Average	**2.13E-06**	8.61E-02	**1.93E-06**	3.09E-02
Soft	Avg. IoU	**8.81E-06**	1.51E-01	**8.07E-06**	6.28E-02
Average	Classic	1.00E+00	1.00E+00	1.00E+00	1.00E+00
Average	Soft	**4.20E-06**	1.63E-01	**3.76E-06**	6.42E-02
Average	Average	5.02E-01	1.00E+00	5.04E-01	9.99E-01
Average	Avg. IoU	4.98E-01	1.00E+00	5.00E-01	9.99E-01
Avg. IoU	Classic	1.00E+00	1.00E+00	1.00E+00	1.00E+00
Avg. IoU	Soft	**1.38E-06**	1.08E-01	**1.22E-06**	3.77E-02
Avg. IoU	Average	2.78E-01	9.98E-01	2.79E-01	9.92E-01
Avg. IoU	Avg. IoU	3.23E-01	9.99E-01	3.24E-01	9.95E-01

Table 6.8: PASCAL VOC T-TEST. The resulting p-values of the one-tailed t-test for Faster R-CNN using the PASCAL VOC dataset. Numbers in bold indicate NMS combinations significant on a 99% confidence level.

stable than Classic/Classic, Classic/Soft, Soft/Classic, and Soft/Soft. As seen in table 6.5, when switching to Average NMS in the RPN head and classic greedy NMS in the RoI head, the stability error is the largest. This can also be observed in the p-values, being at 1.

Furthermore, significantly more stable than the original implementation of Faster R-CNN with classic greedy NMS in both the RPN and RoI head are all the combinations Classic/Average IoU, Soft/Average, Soft/Average IoU, Average/Soft, and Average IoU/Soft. The same case is confirmed by comparing these substituting NMS methods to the benchmark Soft/Classic. The benchmark of Classic/Soft already has a low stability error such that only Classic/Average and Classic/Average IoU yield significantly smaller stability errors. Moreover, the benchmark Soft/Soft can only be significantly outperformed in terms of stability by Classic/Average on a confidence level of 99% for Faster R-CNN using the PASCAL VOC dataset.

The last table with the results of the one-tailed t-test is table 6.9 showing the comparison of NMS methods in MTCNN with the WIDER Face dataset. Since every combination with Soft-NMS performed severely in the MTCNN with given hyperparameters, these combinations are not part of the accuracy and stability evaluation results mentioned above. Nonetheless, it is not reasonable to include these combinations into the comparing t-test and neither as substituting NMS methods nor benchmarking NMS methods. Furthermore, to increase the readability of table 6.9, MTCNN compositions with NMS methods yielding larger stability errors than the benchmark are not shown in this statistical test. Nevertheless, these results of the t-test are meaningful concerning the available data. The results show the comparison with the available implementation of MTCNN. There are 12 combinations of substituting NMS methods that are significantly more stable than the original implementation of MTCNN with classic greedy NMS.

This chapter exemplifies the two perspectives of object/face detection evaluation, which are considered throughout this thesis. First, the accuracy evaluation results are presented in three densely aggregated tables. Each table shows the results of a specific dataset and its respective object/face detector. Then, by holding the aggregation level consistent, the results of the stability evaluation are reported. The tables showing these results summarize the stability metrics and provide an overview of the best performing framework compositions concerning the accuracy evaluation. In addition, a one-tailed t-test is conducted to inspect further the difference in stability of the proposed NMS methods Average NMS and Average IoU. Every possible combination is

NMS P-Net	NMS R-Net	NMS O-Net	Classic/Classic/Classic
Classic	Classic	Average	**3.48E-06**
Classic	Classic	Avg. IoU	**5.98E-04**
Classic	Average	Classic	8.12E-01
Classic	Average	Average	**1.51E-05**
Classic	Average	Avg. IoU	**2.97E-03**
Classic	Avg. IoU	Classic	8.44E-01
Classic	Avg. IoU	Average	**3.03E-05**
Classic	Avg. IoU	Avg. IoU	**3.97E-03**
Average	Classic	Classic	5.95E-01
Average	Classic	Average	**1.29E-06**
Average	Classic	Avg. IoU	**5.37E-04**
Average	Average	Classic	8.48E-01
Average	Average	Average	**4.12E-05**
Average	Average	Avg. IoU	**5.61E-03**
Average	Avg. IoU	Classic	9.10E-01
Average	Avg. IoU	Average	7.44E-05
Average	Avg. IoU	Avg. IoU	4.85E-03
Avg. IoU	Classic	Classic	7.95E-01
Avg. IoU	Classic	Average	**4.59E-05**
Avg. IoU	Classic	Avg. IoU	**7.33E-03**
Avg. IoU	Average	Classic	9.66E-01
Avg. IoU	Average	Average	**8.76E-04**
Avg. IoU	Average	Avg. IoU	3.32E-02
Avg. IoU	Avg. IoU	Classic	9.56E-01
Avg. IoU	Avg. IoU	Average	**2.88E-04**
Avg. IoU	Avg. IoU	Avg. IoU	2.01E-02

Table 6.9: WIDER FACE T-TEST. The resulting p-values of the one-tailed t-test for MTCNN using the WIDER Face dataset. NMS combinations with a stability error greater than the benchmark are not regarded by this test. Therefore, Soft-NMS has an overall bad performance on MTCNN with given hyperparameters, this method is not considered conducting this test. Numbers in bold indicate NMS combinations significant on a 99% confidence level.

compared with the benchmark to decide on a 99% confidence level about the significance of the lower stability error. Five combinations of substituting NMS methods in Faster R-CNN with the COCO dataset outperform the available implementation in terms of more stable detections. Four of these five even outperform every other combination of the benchmarks. For Faster R-CNN using the PASCAL VOC dataset, 6 NMS combinations provide detections with statistically significantly fewer stability errors than the available implementation. For MTCNN using the WIDER Face dataset, there are 16 NMS combinations with significantly increased stability compared to the available implementation. Dataset and framework-specific differences are noticeable. This especially can be seen when comparing the stability errors between the different datasets and frameworks. The stability errors for Faster R-CNN on the COCO dataset are the highest, followed by the PASCAL VOC dataset also evaluated by the Faster R-CNN. The lowest stability errors are stated in the table summarizing the stability evaluation of MTCNN on the WIDER Face dataset. These results are further interpreted and put in comparison with previously stated information in the following chapter.

To summarize the results achieved, table 6.10 shows all frameworks with their respective datasets and results. The benchmark for this table is chosen to be the available implementation of Faster R-CNN and MTCNN. This deviation from the results reported above, where an

extended benchmark with combinations of Soft-NMS was considered. In summary, according to table 6.10, the combination of Average/Classic performs more accurately and more stable than the available implementation of Faster R-CNN using the COCO dataset, with the accuracy measured in the COCO primary evaluation metric AP and statistically significant stability. Regarding Faster R-CNN on the PASCAL VOC dataset, there are combinations of Average/Soft and Average IoU/Soft that both perform better in terms of accuracy (measured with the primary accuracy evaluation metric of PASCAL VOC) and stability (statistically significant) than the available implementation. For MTCNN using the WIDER Face dataset, no combination outperforms accuracy and stability (statistically significant) compared to the available implementation. However, multiple combinations outperform the available implementation in terms of accuracy and statistically insignificant stability.

Framework	Dataset	NMS 1st Stage	NMS 2nd Stage	NMS 3rd Stage	AP	AP^{50}	p-value
Faster R-CNN	COCO	Classic	Classic	-	34.58	57.03	-
		Classic	Average	-	32.23	55.23	3.35E-02
		Classic	Avg. IoU	-	34.28	56.53	1.31E-01
		Soft	Average	-	32.10	54.70	**1.52E-55**
		Soft	Avg. IoU	-	32.15	54.10	3.01E-02
		Average	Classic	-	**34.60**	56.98	1.08E-01
		Average	Soft	-	**34.73**	56.50	**8.14E-64**
		Average	Average	-	32.23	55.23	3.35E-02
		Average	Avg. IoU	-	32.38	54.75	**2.07E-48**
		Avg. IoU	Classic	-	**34.58**	56.95	9.36E-02
		Avg. IoU	Soft	-	**34.70**	56.45	1.59E-01
		Avg. IoU	Average	-	32.08	55.10	**1.17E-48**
		Avg. IoU	Avg. IoU	-	32.38	54.75	**6.30E-03**
	PASCAL VOC	Classic	Classic	-	43.33	79.20	-
		Classic	Average	-	41.28	76.93	**1.18E-08**
		Classic	Avg. IoU	-	41.60	76.00	**2.04E-08**
		Soft	Average	-	40.93	77.03	**2.13E-06**
		Soft	Avg. IoU	-	38.73	72.05	**8.81E-06**
		Average	Classic	-	**43.33**	**79.38**	1.00E+00
		Average	Soft	-	**44.03**	**80.40**	**4.20E-06**
		Average	Average	-	41.43	77.13	5.02E-01
		Average	Avg. IoU	-	41.23	76.13	4.98E-01
		Avg. IoU	Classic	-	**43.38**	**79.40**	1.00E+00
		Avg. IoU	Soft	-	**44.40**	**80.48**	**1.38E-06**
		Avg. IoU	Average	-	39.05	73.93	2.78E-01
		Avg. IoU	Avg. IoU	-	41.70	76.18	3.23E-01
MTCNN	WIDER Face	Classic	Classic	Classic	26.73	47.35	-
		Classic	Classic	Average	26.48	46.80	**3.48E-06**
		Classic	Classic	Avg. IoU	24.65	44.58	**5.98E-04**
		Classic	Average	Classic	**26.83**	47.33	8.12E-01
		Classic	Average	Average	26.55	46.80	**1.51E-05**
		Classic	Average	Avg. IoU	24.63	44.48	**2.97E-03**
		Classic	Avg. IoU	Classic	26.80	**47.35**	8.44E-01
		Classic	Avg. IoU	Average	26.58	46.95	**3.03E-05**
		Classic	Avg. IoU	Avg. IoU	24.60	44.63	**3.97E-03**
		Average	Classic	Classic	26.53	47.20	5.95E-01
		Average	Classic	Average	26.50	46.73	**1.29E-06**
		Average	Classic	Avg. IoU	24.78	44.73	**5.37E-04**
		Average	Average	Classic	**26.83**	47.33	8.48E-01
		Average	Average	Average	26.53	46.78	**4.12E-05**
		Average	Average	Avg. IoU	24.60	44.58	**5.61E-03**
		Average	Avg. IoU	Classic	**26.88**	47.30	9.10E-01
		Average	Avg. IoU	Average	26.58	46.70	**7.44E-05**
		Average	Avg. IoU	Avg. IoU	24.70	44.50	**4.85E-03**
		Avg. IoU	Classic	Classic	**26.78**	**47.35**	7.95E-01
		Avg. IoU	Classic	Average	26.53	46.83	**4.59E-05**
		Avg. IoU	Classic	Avg. IoU	24.80	44.93	**7.33E-03**
		Avg. IoU	Average	Classic	**26.83**	47.33	9.66E-01
		Avg. IoU	Average	Average	26.60	46.95	**8.76E-04**
		Avg. IoU	Average	Avg. IoU	24.68	44.73	3.32E-02
		Avg. IoU	Avg. IoU	Classic	**26.88**	**47.73**	9.56E-01
		Avg. IoU	Avg. IoU	Average	26.63	47.15	**2.88E-04**
		Avg. IoU	Avg. IoU	Avg. IoU	24.68	44.90	2.01E-02

Table 6.10: OVERALL RESULTS. The overall summary of results. This table shows both in this thesis considered frameworks and the results for the respective datasets. The columns AP and AP^{50} show in this table accuracy results and are in bold if the accuracy is greater or equal to the available implementation with NMS combination Classic/Classic(/Classic). The last column shows the p-value of the one-tailed t-test between each combination and the benchmark. Only the available implementation is therefore considered as benchmark. Numbers in bold for this column indicate NMS combinations significant on a 99% confidence level.

Chapter 7

Discussion

In this chapter, the results are put in context and interpreted to enhance bounding box stability with Non-Maximum Suppression. Moreover, the results are regarded as measuring the temporal and spatial stability of predicted bounding boxes in still images and further discussed. Lastly, various limitations must be considered with the approach of this thesis.

7.1 Non-Maximum Suppression Alternatives

In chapter 3 two Non-Maximum Suppression methods are proposed that have not been reportedly implemented so far. Average NMS and Average IoU NMS are based in their single components on different implementations from the literature, but the combination of these components is novel. They are especially recalculating the detection score as the weighted average of the detection scores for Average NMS and the IoU weighted average for Average IoU NMS. Overall, a combination with one of these NMS methods achieves very comparative results in the accuracy of the detections. By inspecting the accuracy of the worst-performing NMS combination for Faster R-CNN using the COCO dataset, it can be calculated that Soft/Average is 7.2% below AP and 4.1% below $AP^{0.5}$ of the benchmark with Classic/Classic. Though, the best performing combination of Average NMS in the RPN head and Soft-NMS in the RoI head scores 0.4% better in AP and 0.9% below $AP^{0.5}$ of the available implementation. For Faster R-CNN using the PASCAL VOC dataset, the worst performing combination featuring Soft-NMS in the RPN head and Average IoU NMS in the RoI Head achieves a 10.6% lower AP and 9% lower $AP^{0.5}$ than the available implementation. On the other hand, the best performing combination of Average IoU in the RPN head and Soft-NMS in the RoI head scores 2.5% higher in AP and 1.6% better in $AP^{0.5}$ than the benchmark. The inspection of accuracy results for MTCNN using the WIDER Face dataset leads to an increase of 0.6% in AP and an increase of 0.8% in $AP^{0.5}$ with Average IoU NMS in the P- and R-Net and classic greedy NMS in the O-Net. The worst performing results in terms of accuracy and stability with MTCNN is a combination with Soft-NMS. As already mentioned in the results in chapter 6 the application of Soft-NMS could not produce any comparable detection results with MTCNN. The reason for bad performance is that the default hyperparameters for MTCNN are not favorable for applying Soft-NMS. Since Soft-NMS works by decaying the detection score of the bounding box, all non-maximum detection scores are decreased. This NMS application works well with Faster R-CNN since there are only minimal thresholds between the two stages. However, with MTCNN, there are thresholds of 0.6, 0.7, and 0.7 for the three stages. These thresholds limit the capability of the framework to gradually regress the detected bounding boxes, classify them and filter detections by these thresholds. Instead, already in the proposal stage are the scores decayed such that a majority of proposals do not reach the refinement stage. Those proposals which make it to the output stage are then mostly equipped with a detection score that is below

the significant threshold of 0.7. Therefore, MTCNN outputs a minimal number of detections. This output results in degenerate accuracy and stability scores. A potential solution would be to adjust the decaying factor of Soft-NMS or lower the thresholds of the MTCNN. Although this represents a valid solution, the results would not be comparable to the other detection results of Faster R-CNN using Soft-NMS or MTCNN with the other three NMS methods. For comparability reasons, the weak performance of Soft-NMS in MTCNN is tolerated.

As discussed in the previous chapter and summarized in table 6.10, there are proposed combinations of Non-Maximum Suppression methods that both outperform the accuracy and are statistically significant in terms of higher stability in the Faster R-CNN regardless of the dataset used. However, the high scoring combinations in AP, $AP^{0.5}$ are not the same. This result implies that there is no unique combination of NMS methods to be placed in the RPN and RoI head to achieve superior accuracy and significantly more stable detection results at the same time. The interpretation of this finding is that the hyperparameters for the Faster R-CNN post-processing, which are also shared with the NMS methods, are not optimally selected and therefore do only partially and distinctively contribute to the generality of well-performing results. Nevertheless, when lowering the aspiration of surpassing all the benchmarks in both AP and $AP^{0.5}$, then there is one combination that can be considered as a successful substitution of Non-Maximum Suppression in the Faster R-CNN framework. When replacing classic greedy NMS in the RPN head with Average NMS and in the RoI head with Soft-NMS, there is a significant increase in stability for both respective datasets. This combination surpasses AP and $AP^{0.5}$ of the available implementation in the Faster R-CNN with the PASCAL VOC dataset and AP of the available implementation in the Faster R-CNN with the COCO dataset. The only metric, which Average/Soft fails to surpass, is $AP^{0.5}$ in the Faster R-CNN with the COCO dataset. When also lowering the aspirations of outperforming the benchmark of the available implementation.

7.2 Measuring Stable Bounding Boxes

The stability evaluation metric introduced in chapter 2 and the strategy to materialize this evaluation metric for still images in chapter 4 are entirely novel approaches. The stability evaluation has its roots in the video detection and multi-objects tracking field. There is already little evidence of reported usage in the literature for the main application of stability evaluation. All the more unknown is this method in the field of object detection in static images. By creating sequences of augmented static images, the stability evaluation metric becomes accessible even for non-video sequences. The decision about the length and the kind of augmentation is not made with evidence. However, the parameters for the augmentation are based on an analysis of a randomly select population of images for Faster R-CNN and MTCNN, respectively. The goal of this selection is to establish comparability between the frameworks in terms of their stability errors. By doing so, as mentioned in the results, it can be observed that the stability error decreases with the decreasing size of the datasets. This observation suggests that the more specialized a framework is for a specific task, the more stable its predicted detections are. Substituting Non-Maximum Suppression methods with stabilizing methods helps even more to increase the stability.

7.3 Limitations

In this section, the limitations are stated concerning the research conducted in this thesis. Especially the proposition of a novel strategy of measuring the stability of predicted detections adds many considerations worth noting in this section.

- Faster R-CNN and MTCNN are evaluated with every combination of NMS methods with their respective default values. Also, NMS-specific parameters are chosen at default values, such as the decaying factor of Soft-NMS. On the one hand, this decision leads to results that cannot indicate a single NMS combination that improves all accuracy-specific metrics and is statistically significant in terms of stability. On the other hand, this leads to depraved results for Soft-NMS in MTCNN. By adjusting thresholds of MTCNN or increasing the decaying factor of Soft-NMS, more informative results for Soft-NMS are feasible
- The focus during the selection of models for the frameworks is on the availability and the allowance to alter NMS functions. There are better-performing models featuring Faster R-CNN and MTCNN available, but this thesis solely focuses on the replacement of classic greedy NMS and, therefore, only measures the relative performance within the frameworks at hand
- There was only reasonable effort invested in fine-tuning the Faster R-CNN model to the PASCAL VOC dataset because there is no pre-trained version available with Torchvision
- The aspect of evaluation speed of the frameworks is not considered in this thesis. Available implementations of the frameworks feature NMS functions written in C++, which is more efficient than the own implementation in Python
- The stability evaluation metric originates from a source with questionable quality. The metric has also not been frequently reported in the past. Nevertheless, it is considered an excellent approach to measure detection results' spatial and temporal stability, and no calculation errors are spotted in the respective equations. Since this specific stability metric is not well represented in the literature, there is no ground for comparison. Therefore, the only comparison that can be drawn internally in this thesis.
- There is a deviation of the usage of the stability metric to the proposed metric in the literature. Instead of using the area under the curve for 10 IoU thresholds, only the single stability error using an IoU threshold of 0.5 is reported. This adjustment is due to the reuse of detection and ground truth matches from the COCO accuracy evaluation, which only registers matches above 0.5
- From the literature, it is unclear what happens to center position error and scale and ratio error if detection is present in only one frame of the sequence. The standard deviation cannot be calculated with a single number, and therefore a standard deviation of 0 was reported instead. It is assumed that the fragment error accounts for such a case. Even though the fragment error for such a detection scenario is only in $\{0.333, 0.667\}$. This depends on whether the detected frame is at the beginning/end or in the middle of the frames. Meaning the fragment error represents the same amount as in a potential case with two or three detections in the sequence
- The construction of the sequences in terms of length and kind of transformations are not empirically validated. Therefore, longer frame sequences would lead to more informative results, and a different arrangement of the frames could lead to another overall outcome. However, the results are consistently evaluated using the same length and the same arrangement
- The transformational parameters of the horizontal and vertical translations and the rotations are selected using an analysis of a limited range of parameter values on a subset of data
- The t-test in the results chapter is conducted assuming that the stability errors are normally distributed and independent within and between each other

- The exact t-test cannot be conducted using the accuracy results of the different NMS combinations with the different frameworks. Since the finally reported AP and $AP^{0.5}$ metrics are the mean of the accuracy metrics of the individual frames; they are dependent on each other. Also, due to the number of different framework compositions, evaluating each framework with a differently seeded subset of the data is not reasonable

Chapter 8

Conclusion & Future Work

This chapter presents a brief synopsis of the milestones and the significant contributions towards answering the two initially stated research questions. Besides, core results are comprehensively summed, and ultimately, considerations about future research are asserted.

8.1 Conclusion

The focus of this thesis lies on the stabilization of bounding box predictions using an aggregating alternative of the classic greedy Non-Maximum Suppression. In chapter 2 the grounds are laid by introducing the WIDER Face, MS COCO and PASCAL VOC; one dataset for face detection and two for various object detection. Furthermore, the concept of object and face detection is explained with instances of single- and multi-stage detectors. A more particular aspect of the presented detectors is elaborated with the Non-Maximum Suppression as a substantial post-processing part. Additionally, recent developments concerning optimizing NMS are stated. The fundamental part with the focus on background information and related work is rounded off with an introduction to the predominantly used evaluation metrics to determine a detector's accuracy of locating and classifying objects and faces. Thereby, a complementary perspective on the holistic evaluation topic is introduced with the term stability. An evaluation metric proposed for the related field of video detection and multi-object tracking is introduced and formally stated.

In chapter 3 the methodological concept of achieving more stable bounding boxes with Non-Maximum Suppression is formalized. As part of this concept and for benchmarking reasons, the well-established method of classic greedy NMS and Soft-NMS are chosen to be re-implemented. In addition, by combining multiple different approaches from the literature, two novel NMS algorithms are proposed. In contrast to the other two methods Average NMS and Average IoU NMS consider all predicted bounding boxes that overlap to a threshold to be aggregated to one informative bounding box with an associated detection score. The proposed algorithms perform still in a greedy manner but utilize the information of all bounding boxes located in close proximity to an object. All four NMS methods are then combined in a single parametrized Multi-Non-Maximum Suppression function. This chapter contributes the foundation of the implementation to answer the first research question.

Since the two newly proposed NMS algorithms aim to stabilize the exact location of detected bounding boxes, the measurability of this goal in still images is limited. Therefore, in chapter 4, a new method is proposed by fabricating consecutive frames of augmented images to simulate moving objects in a video sequence. This approach enables the potential to measure the temporal and spatial stability of object and face detectors, in addition to the already well-established accuracy metric. This chapter materializes the concept necessary towards answering the second research question. Therefore, a major contribution to the answer to this research question is made.

Chapter 5 guides the realization of the proposed Multi-NMS function and the approach to sequence augmented images in an experimental framework. The two detection frameworks Faster R-CNN and MTCNN, are inspected in more detail. The forward pass of example images is studied and identified where the Multi-NMS function replaces the initially implemented classic greedy NMS. This chapter is supplemented by implementation details to the evaluation pipeline of measuring accuracy and stability. This chapter facilitates the experimental setup to collect evidence to answer the first research question. This evidence can also be utilized to support the approach proposed to answer the second research question.

In chapter 6, the extensive results are summarized and presented concerning the two perspectives of measuring the object and face detection outcome. Accuracy- and stability-specific results are discussed and put in comparison. A one-tailed t-test is performed to indicate statistically significant increases in stability. Equipping Faster R-CNN and MTCNN with the newly proposed NMS methods leads almost half of the presented experiments to significantly more stable bounding box predictions. Even in some cases to more accurate predictions in comparison with the initially implemented classic greedy NMS. Despite the numerous well-performing compositions of Faster R-CNN and MTCNN with the proposed NMS methods, there is a minority outperforming the status-quo concerning the accuracy and stability metrics combined. Replacing classic greedy NMS in the post-processing of the Regional Proposal network with the newly proposed Average NMS and substituting the initial implementation in the post-processing of the Region of Interest head with Soft-NMS yields superior results in the Faster R-CNN for both of the used datasets. Predicting with the same scenario with Average IoU NMS in the RPN head instead of Average NMS increases the accuracy and stability performance even more for the PASCAL VOC dataset. However, this is only partially true for the COCO dataset evaluated on the same framework.

Furthermore, especially the absence of Soft-NMS is noticeable in the results of MTCNN. Since a combination of Soft-NMS yields good results in the Faster R-CNN, it is expected to behave similarly in the framework for face detection. Lamentably, the experimental setup does not endorse the functionality of Soft-NMS, and the performance is by no means comparable to any other combination of NMS methods. Even though Soft-NMS does not contribute much evidence to the results of MTCNN, there is still a contribution to the learning process. This failure highlights the importance of not only bounding box detections but also detection probabilities. Both contribute from the forward pass through the framework to the final output. Referring to chapter 3, where Average NMS and Average IoU are proposed, this learning is beneficial. Several sources propose the two algorithms; some already implemented the averaging aspect of averaging bounding box coordinates of overlapping boxes. However, the methods proposed in this thesis differ in terms of recalculating the score. Comparing the results of the previous chapter and considering the weak performance of Soft-NMS in MTCNN, it can be deduced that the computation of the score is just as crucial as the bounding boxes, even though they are often overlooked in the visual representation of the detection results.

In conclusion, this thesis provides an extensive foundation to study the influence of Non-Maximum Suppression on the object and face detection. It exemplarily shows the vast potential of this post-processing algorithm to increase the accuracy further but, more importantly, the stability of bounding box prediction using a weighted average for detection boxes and scores. Therefore, concerning the first research question, the weighted average approach of Non-Maximum Suppression positively influences the stability of bounding box detection in object and face detectors. The positive influence is shown on a 99% confidence level with multiple combinations, including Average and Average IoU Non-Maximum Suppression. Also, the second research question can be confidently answered. Combining the approach of treating static augmented images as frames in a sequence with the stability evaluation metric for video detection and multi-object tracking enables the capability of quantification and measurability to object and face detectors.

8.2 Future Work

While conducting this work, numerous appealing areas for further research emerged. Some are linked to the limitations mentioned in the previous chapter, but most relate to the paucity of evidence in this under-researched area.

- The application of Average NMS and Average IoU NMS likely require a revision of the IoU threshold in the different stages different from the default values used for classic greedy NMS. The overwhelming amount of framework compositions limit the number of experiments conducted, therefore. The potential for further increase in accuracy and stability is probable

- Since the utilization of Average and Average IoU NMS in combination with Soft NMS in the RPN and RoI head respectively yields good results for Faster R-CNN, there might be a tremendous potential of adding a decaying factor to the Average and Average IoU NMS algorithms

- Lowering the thresholds in MTCNN or increasing the decaying factor for Soft-NMS makes this NMS method applicable in the framework

- The models used for the frameworks are mostly underperforming on the selected datasets compared to the state-of-the-art performance of object and face detectors. Better-performing models lead to more matches between detections and ground truths. This increased performance, in turn, enables a more accurate study of the stability of detected bounding boxes

- Measuring the stability of facial landmarks in MTCNN is expected to correlate with the stability of bounding boxes since these are predicted based on the bounding boxes. An alternative way of integrating the additional information of landmarks into the process of Non-Maximum Suppression can be done comparable as proposed by Ranjan et al. (2017). Thereby, the landmarks offer additional information about the correct location of the face

- MTCNN uses softmax for binary classification; this could be replaced by binary cross-entropy. Furthermore, in the P-Net, NMS is individually applied to every single level of the image pyramid. Instead, the Multi-NMS function can handle multiple levels with offset. Hence, this loop can be replaced, and the originating level passed to the M-NMS function

- The limiting factor for the throughput of experimental trials is the computation time. Since the re-implemented NMS algorithms give great insight into the actual bounding box selection process, they perform relatively slow. By orienting at the initial implementation of classic greedy NMS by Torchvision, efficiency can be increased, and framework composition evaluation can be performed faster

- Decoupling the stability evaluation from the accuracy evaluation would require a matching process between ground truths and detections. However, IoU thresholds could be set independently and inspected in that regard, which is closer to the initially proposed stability evaluation metric

- There are numerous ways to adjust the construction of sequences to simulate the movements of objects. By increasing the number of frames in a sequence, usage of more diverse transformations, differently arranging the frames, or simply using labeled video sequences, the evaluation perspective on stability can be enhanced

- The parameters used for horizontal and vertical translations can be adjusted to all kinds of detections. When tracing back detections to their origin in the Feature Pyramid Network, the scaling factor can be considered when choosing the values of these parameters. Therefore, shifts that overlap with the exact position of the sliding window can be prevented
- The stability evaluation can be paired with the research in translation, rotation and scale in-, and equivariance
- For this work, only multi-stage detectors are investigated. The impact on single-stage detectors is worth considering
- Ultimately, using Average and Average IoU NMS during the training can have an impact on the object and face detectors to regress more stable bounding boxes

Appendix A

Further Information & Implementation Details

A.1 Non-Maximum Suppression Alternatives

While Soft-NMS is aiming for a replacement of classic greedy NMS with a method comparable in light of computational complexity, ASAP-NMS focusses on increasing the speed of NMS. Tripathi et al. (2020) point out NMS of being a bottleneck to object detectors and propose the method accelerating Non-Maximum-Suppression using spatially aware priors (ASAP). The speed of NMS is increased by reducing the number of proposals which are not located in close proximity of each other and therefore have a low chance of affecting the suppression decision. With this technique latency is reduced by a pre-computed lookup table to provide an overlap in the anchor space as a proxy for the overlap in the proposal space. Therefore, only proposals with a reasonable overlap are looked up and compared with the maximum scoring bounding box. Without sacrificing the accuracy on COCO and PASCAL VOC, ASAP-NMS achieves a decrease of latency from 13.6ms to 1.2ms on CPU during the the post processing step of Non-Maximum Suppression. Despite the increase in speed of NMS, Tripathi et al. (2020) only measure their method against the accuracy of bounding box prediction and not the stability of these boxes.

Wu and Li (2021) have the goal of eliminating false positives from Non-Maximum Suppression by considering it as a combinatorial optimization problem. Additionally to classic greedy NMS, the chaotic whale optimization algorithm is introduced to solve this problem. The chaotic whale optimization is a meta-heuristic algorithm that is inspired by nature that initializes a set of randomly selected candidate solutions and iteratively optimizes this set under supervision of a fitness function. In more detail, the chaotic whale optimization utilizes the spatial locations of bounding boxes and classic greedy NMS to partition them into candidate regions to approximate the number of real objects. All bounding boxes have the same chance of being selected, this reduces the greediness of classic greedy NMS by the global search ability. With the chaotic search in combination with a fitness function, an optimal combination of bounding boxes can be found. CW-NMS achieves a marginal increase in mAP of COCO and PASCAL VOC compared to the classic greedy NMS and Soft-NMS. However, due to the randomness of the selection process, there is a chance to further increase instability of the detected bounding boxes compared to classic greedy NMS.

Tan et al. (2019) see non-maximum suppression as a ranking problem and propose an end-to-end learning procedure. The subnetwork learns and predicts ranking scores based on the overlap with the ground truth bounding boxes. These ranking scores get combined with the classification scores to obtain a single criterion to conclude on the suppression decision of an associated bounding box.

Tychsen-Smith and Petersson (2018) introduce a bounding box regression loss which goes well with their modification of NMS called Fitness NMS. The bounded IoU loss aims to optimize the proposed region of interest with the ground truth bounding box. Similarly, with Fitness NMS only bounding boxes that maximize their estimated IoU with the ground truth get selected during the post processing step. To conclude the implementation, only an accuracy evaluation was performed without addressing the specifics of stability or localization in detail.

A.2 Data Preparation Implementation Details

As a first step the downloaded datasets get read in by a subclass of `torch.utils.data.Dataset` as declared in the PyTorch documentation.[1] Within such a dataset subclass the images are generally opened using the pillow library (PIL), which enables Python with image processing capabilities.[2] The annotations get parsed depending on the source format of the individual dataset annotations. A dataset subclass has a minimum of three methods built in, namely `__init__` as the constructor, `__getitem__` to access images and annotations by index and `__len__` to provide the length of the dataset.[1] The constructor of the dataset subclass requires multiple arguments, which can be divided into two categories: data origin related and data transformation related. The data origin related arguments differ depending on each of COCO, PASCAL VOC or WIDER Face data and will be explained individually. In general the directory of the stored images and annotations needs to be passed first or whether the data should be downloaded.

The second argument category with the data transformation related arguments is the same for each of the datasets and consists of four different transformational arguments. Three of these will be introduced in the next section of affine transformations. Nevertheless, the first transformational argument is implemented in the reused PyTorch dataset subclass and is inherited from the `VisionDataset` and can be one of `transform`, `transforms` or `target_transform`.[3] All of these possible arguments take a transform function which gets manually defined before creating a dataset object and is called `get_transform()`. `transform` takes the transform function `get_transform()` which itself takes a PIL image and transforms it.[1] `transforms` takes as well the transform function `get_transform()` of an input and target and returns a transformed version of both.[1] And lastly, `target_transform` takes the transform function `get_transform()` which only takes a target and returns the transformed target again.[1] The transform function `get_transform()` is defined as a composition of either standard transform objects or functional transform objects from `torchvision.transforms` or `torchvision.transforms.functional` respectively in the form of a list.[4] The difference between standard transform objects and functional transform objects lies in the randomness of their parameters.[4] Functional transform objects do not contain a random number generator and are used to create reproducible transformations.[4] Regardless of functional or non functional transformations, the composition takes a list of transform objects and is nested in the transform function `get_transform()` which gets passed to the dataset subclass.[4] An example of such a transform object that gets appended to a list to be composed, is the `ToTensor` object, which is a functional transformation object from `torchvision.transforms.functional`. With `ToTensor` an image of type PIL image or `numpy.ndarray` with the shape (H x W x C) with pixel values in the range $[0, 255]$ gets converted into a `torch.FloatTensor` of shape (C x H x W) with a range between $[0.0, 1.0]$.[4] In terms of shape; C stands for the color channels, H for height and W for width. The `ToTensor` object is the first transformation that gets appended to the composition within the `get_transform()` function.

[1] PyTorch Datasets: https://pytorch.org/vision/stable/datasets.html
[2] Pillow on Github: https://github.com/python-pillow/Pillow
[3] Vision Dataset on Github: https://github.com/pytorch/vision/blob/master/torchvision/datasets/vision.py
[4] Vision Dataset on Github: https://pytorch.org/vision/stable/transforms.html

Regarding the practical implementation of the second goal of the data preparation, the datasets not only have to be capable of holding the images and ground truths in an original state but also produce an affine transformed version of the data as mentioned above. Therefore, when looking at the variety of transform objects of `torchvision.transforms`, it can be observed that there exists a class called `RandomAffine`.[4] This class takes among others the parameters degrees and shear, which lead to a transformation in terms of rotation and shifts in x and y dimension.[4] However, when looking closely at the `forward()` function of the `RandomAffine` class, it can be seen that it only accepts an image with the type of a PIL image or a tensor as input.[4] Similarly the return value is only the affine transformed image.[4] This means when utilizing an object of the `RandomAffine` class in the composition of the `get_transform()` function, it can only be passed with the `transform` parameter of the dataset subclass. Which further means that only the image gets transformed but not the associated annotations. The result of such a unilateral transformation of only the images would leave out the annotations from this transformation as observable in figure 5.1. Further down the prediction pipeline the model would predict bounding boxes based on a transformed image but the evaluation metric would then compare those bounding boxes to untransformed ground truth annotations. A possible workaround could be to define a second transform function with a similar transformation pipeline dedicated to the annotations but there I face the problem that the `RandomAffine` class only takes PIL images or tensors of images as inputs in the `forward()` function.[4]

By relating back to the initially mentioned four transformational parameters which are passed when creating such a dataset subclass, the first transformational parameter was explained above. In order to delegate the shifts in x and y direction and the rotation, the last three transformational arguments are needed. Even though two almost independent transformation pipelines are in use, they can be content-wise delimited. Type transformation as seen in the `get_transform()` function gets passed via the first transformational parameter when instantiating an object of the dataset subclass. The affine transformation then has to be moved inside the dataset subclass and implemented in the `__getitem__` method. This means every time an image and annotations pair gets extracted from the dataset subclass the affine transformation gets executed and returned.

After explaining the details behind the data origin related and the four transformational parameters, an object of the dataset subclass can be created and stored in a variable. As the last step of data preparation the dataset subclass object gets passed to a data loader (`DataLoader`) from `torch.utils.data` in order to be able to iterate through the dataset during prediction and evaluation.[5]

With regard to the PyTorch implementation of the COCO, PASCAL VOC and WIDER Face dataset from `torchvision.datasets` substantial changes are applied to comply with with the functionalities required for the detection and especially for the evaluation pipeline for experimental purposes.[1] Most importantly, the constructor of the `COCODetection`, `VOCDetection` and `WIDERFace` subclasses of the `VisionDataset` class must be complemented with the 3 transformational parameters for affine transformations described above to enable shifts and a rotation via the `__getitem__()` method of the class. This addition of the transformational arguments is applied to all the datasets. Additionally, the `WIDERFace` class constructor needs another additional argument to distinguish between the difficulties provided in the data as described in section 2.1.3. To prevent the `WIDERFace` class to load all the data during instantiation and then only select a subset of the data, it is decided to distinguish during the data parsing process. This means that only the respective images and annotations of the chosen difficulty get parsed. This is achieved by implementing an additional filtering function, which is called during the parsing. The original implementation of `torchvision.datasets.widerface` iterates over the annotation file to extract the image file location and annotations from the txt file. The image path, which is a string, indicates to what event category the image and its annotations belong. As described in section 2.1.3

[5] PyTorch Dataloader: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

the event categories between 40 and 60 are classified to belong to the easy difficulty. By splitting up the string and extracting the event category, an integer comparison selects images and annotations belonging to the range $[40, 60]$. Even though it is mentioned in 2.1.3 to only utilize the easy partition of the WIDER Face dataset for this analysis, the selection also works for medium and hard difficulties of the dataset images. Additionally, the mentioned invalidity check for bounding box annotations from 2.1.3 is also placed within this filtering function.

A.3 Faster R-CNN Post Processing

The custom adjustment implementation of the Non-Maximum-Suppression function call in RPN and Fast R-CNN has only small impact in the code but due to strong interleaving of single function calls, it drags a chain of changes in the original code with it. Since both NMS applications are located in the post processing steps of the two modules of Faster R-CNN, they both execute the same NMS function and the same adjustments can be applied. With the original implementation of PyTorch, the orchestration of instantiation of the two modules (RPN and Fast R-CNN) is achieved via submodules of RPN and Fast R-CNN (RPN anchor generation, RPN head, box RoI pool, RoI heads, Fast R-CNN predictor and others) in the `FasterRCNN` class that is itself a subclass of `GeneralizedFasterRCNN`. Both are located in `torchvision.models.detection`. While the majority of instantiations are done within the `__init__()` function of the `FasterRCNN` class (backbone instantiation is done via a standalone function in the same Python file), the forward pass is implemented in the main class `GeneralizedFasterRCNN`. The two NMS function calls are located in `torchvision.models.detection.rpn` and `torchvision.models.detection.roi_heads`, which are both instantiated during the model construction via `FasterRCNN`. The original NMS implementation of PyTorch does not allow parameters upon the model instantiation that can change the behavior of the NMS function, which is itself located in `torchvision.ops.boxes`.[6] Therefore, Multi-NMS replaces the necessity of `torchvision.ops.boxes` by providing a new Python file outside of the Torchvision side package for example in a self maintained repository.[8] To access the Multi-NMS function from the RPN and Fast R-CNN post processing, these two files have to be copied over to the same repository as well. Additionally, the classic greedy NMS function returns only the indices of the selected boxes in a descending order of the objectness or class probability score. With the replacing Multi-NMS function, described in 3, bounding box coordinates and scores are computed in an alternative way and can not be referenced back by index of the input bounding boxes and scores as shown in listing A.1. Therefore, the Multi-NMS function returns instead of indices, all selected bounding boxes with the respective scores and levels/labels according to listing A.2. This change in return values have to be reflected in the RPN and Fast R-CNN post processing as well by replacing the index-specific selection of instances with the return values of the new NMS function. Moreover, to create a flexible experimental setup with varying NMS parameters (NMS methods, IoU thresholds, NMS method specific thresholds and limits) these parameters have to be passed through the RPN and Fast R-CNN instantiation in the `FasterRCNN` class. This means that the `FasterRCNN` class also has to be adjusted and stored in the same repository as mentioned above. By providing customized versions of the `FasterRCNN`, RPN and RoI heads classes and an own implementation of the NMS function, the forward pass of the Faster R-CNN is still done via `GeneralizedFasterRCNN` but utilizes the classes in the Python files located in the newly created repository.

[6] Torchvision NMS on Github: https://github.com/pytorch/vision/blob/master/torchvision/ops/boxes.py

```
keep = box_ops.batched_nms(boxes, scores, labels, self.nms_thresh)
keep = keep[:self.post_nms_top_n()]
boxes, scores, labels = boxes[keep], scores[keep], labels[keep]
```

Listing A.1: ORIGINAL INDEX SELECTION IN FASTER R-CNN

```
indices, boxes, scores, labels = mnms(boxes, scores, labels, nms_thres=self.nms_thresh,
    multi_class=True, method=self.nms_method, score_thres=0.1, limit=self.
    detections_per_img)
```

Listing A.2: REPLACEMENT OF INDEX SELECTION IN FASTER R-CNN

A.4 Implementation Details of MTCNN

The difference to the accessibility of the provided code to Faster R-CNN is that MTCNN is not available as PyTorch implementation but must rather be cloned from Github.[7] The code described in section 5.2.2 depends heavily on the Facenet-PyTorch implementation.[7] As mentioned by explaining the pass through of an example image in section 5.2.2, the only changes that differ from the original implementation are the NMS function calls and the respective usage of its return values. In total there are 4 different NMS function calls (2 in P-Net, 1 in R-Net and 1 in O-Net). The original implementation of Facenet-PyTorch utilizes the same NMS function of PyTorch as Faster R-CNN by importing `torchvision.ops.boxes`.[8] The `torchvision.ops.boxes` NMS works in a classic greedy fashion by returning solely the indices of the selected boxes according the algorithm 1. Since the classic greedy NMS only returns indices, these indices then have to be looked up in the bounding boxes, scores and regression values, which two of them were passed to the function as parameters. The definitive selection only takes place as shown in listing A.3. The Multi-NMS function that implements a variable way to execute different NMS methods with a set of parameters, is required to alter the bounding box coordinates (in case of averaging overlapping boxes above IoU threshold) and returns the new bounding boxes. The aggregated bounding boxes then cannot be retrieved by slicing with indices in the original bounding box tensor `boxes` as displayed in listing A.3. Therefore, the substitution of the NMS function call also drags along a change in the subsequent further slicing of tensors in MTCNN as shown in listing A.4. By having the selected boxes and scores directly as a return value of the Multi-NMS function, there is no need for slicing in the `boxes` tensor. It can simply be continued with the usage of the tensor `nms_boxes` instead. However, the replacement of the selection process as seen in the implementation details for Faster R-CNN in section 5.2.1 cannot be directly applied to MTCNN. MTCNN also requires the indices to slices the non-suppressed bounding boxes when multiple images are processed in a batched fashion. Therefore, the Multi-NMS function not only returns the selected bounding boxes, scores and levels, but also the indices of those instances which where not suppressed.

[7] Facenet-PyTorch on Github: https://github.com/timesler/facenet-pytorch
[8] Torchvision NMS on Github: https://github.com/pytorch/vision/blob/master/torchvision/ops/boxes.py

```
pick = batched_nms(boxes[:, :4], boxes[:, 4], image_inds, 0.7)
boxes, image_inds, mv = boxes[pick], image_inds[pick], mv[pick]
```

Listing A.3: ORIGINAL INDEX SELECTION IN MTCNN

```
pick, nms_boxes, nms_scores, nms_levels = mnms(boxes_scale[:, :4], boxes_scale[:,4],
    image_inds_scale, nms_thres=0.7, multi_class=True, method=first_stage_method,
    score_thres=0.01)
```

Listing A.4: REPLACEMENT OF INDEX SELECTION IN MTCNN

Appendix B

Code

B.1 Multi-NMS Implementation

```
def mnms(boxes, scores, levels, nms_thres=0.5, multi_class=True, method='vision',
    score_thres=0.01, limit=1000):
    """Multi-Non-Maximum Suppression implementation to variably change between NMS
    methods to select or aggregate
        bounding boxes
    Args:
        boxes (Tensor[N, 4], required): Bounding boxes to perform NMS. Boxes are
    expected to have the format
            [xmin, ymin, xmax, ymax] with 0 <= xmin < xmax and 0 <= ymin < ymax.
        scores (Tensor[N], required): Scores for each of the bounding boxes.
        levels (Tensor[N], required)): Levels/Labels for each of the bounding boxes.
        nms_thres (Tensor[N], required): IoU threshold above which NMS is performed.
    Default: 0.5
        multi_class (bool, optional): Multi-class distinction to apply an offset to not
    compare boxes between
            different levels/labels. Default: True
        method (string, optional): NMS method, can be one of 'vision', 'classic', 'soft
    ', 'average', 'average_iou'.
            Default: 'vision'
        score_thres (flaot, optional): Score threshold, only applicable when method='
    soft'. Performs filtering
            on decayed scores. Default: 0.01
        limit (int, optional): Limits the number of boxes to be returned. Default: 1000
    Returns:
        tuple(indices, boxes, scores, levels): indices (Tensor[K]), boxes (Tensor[K, 4])
    , scores (Tensor[K]), levels (Tensor[K])
    Raises:
        Exception: When an invalid method is passed that is not implemented
    """
    device = boxes.device
    if boxes.numel() == 0:
        return torch.empty((0,), dtype=torch.int64, device=levels.device), \
               torch.empty((0, 4), dtype=torch.int64, device=boxes.device), \
               torch.empty((0,), dtype=torch.int64, device=scores.device), \
               torch.empty((0,), dtype=torch.int64, device=levels.device)
```

Listing B.1: OWN IMMPLEMENTATION OF MULTI-NON-MAXIMUM SUPPRESSION PART 4

```
...
    all_methods = ['vision', 'classic', 'soft', 'average', 'average_iou']
    if not any(method.lower() == s for s in all_methods):
        raise Exception("NMS method not defined, try one of {}".format(all_methods))

    if method == 'average_iou':
        iou = True
        method = 'average'
    elif method == 'average':
        iou = False

    indices = torch.sort(scores, descending=True)[1]
    boxes = boxes[torch.sort(scores, descending=True)[1]]
    levels = levels[torch.sort(scores, descending=True)[1]]
    scores = torch.sort(scores, descending=True)[0]

    D_b, D_s, D_l, D_i = [], [], [], []
    if multi_class:
        max_coordinate = boxes.max()
        offsets = levels.to(boxes) * (max_coordinate + torch.tensor(1).to(boxes))
        boxes = boxes + offsets[:, None]

    if method == 'vision':
        nms_indices = box_ops.nms(boxes, scores, nms_thres)
        D_i, D_b, D_s, D_l = nms_indices, boxes[nms_indices],
         scores[nms_indices], levels[nms_indices]

    elif method == 'classic':
        while boxes.shape[0]:
            M = boxes[0]
            D_b.append(M)
            D_s.append(scores[0])
            D_l.append(levels[0])
            D_i.append(indices[0])
            boxes = boxes[1:]
            scores = scores[1:]
            levels = levels[1:]
            indices = indices[1:]
            inds_below = torch.where(
            box_ops.box_iou(
            M.unsqueeze(0), boxes) < nms_thres)[1]
            boxes = boxes[inds_below]
            scores = scores[inds_below]
            levels = levels[inds_below]
            indices = indices[inds_below]
        D_l = torch.stack(D_l)
        D_b = torch.stack(D_b)
        D_s = torch.stack(D_s)
        D_i = torch.stack(D_i)
```

Listing B.2: OWN IMMPLEMENTATION OF MULTI-NON-MAXIMUM SUPPRESSION PART 4

```
...
   elif method == 'average':
        while boxes.shape[0]:
            M_b = boxes[0]
            M_s = scores[0]
            M_l = levels[0]
            M_i = indices[0]
            boxes = boxes[1:]
            scores = scores[1:]
            levels = levels[1:]
            indices = indices[1:]
            ious = box_ops.box_iou(M_b.unsqueeze(0), boxes)
            inds_below = torch.where(ious < nms_thres)[1]
            inds_above = torch.where(ious >= nms_thres)[1]
            cand_boxes = torch.cat((boxes[inds_above], M_b.unsqueeze(0)), 0).to(device)
            cand_scores = torch.cat(
            (scores[inds_above], M_s.unsqueeze(0)), 0).to(device)
            if iou:
                w_iou = torch.cat(
                (torch.ones(1), ious[0][inds_above]), 0)
            else:
                w_iou = torch.ones(cand_scores.shape[0])

            cand_scores = cand_scores * w_iou.to(device)
            weights = cand_scores / cand_scores.sum()
            new_box = torch.sum(
            weights.reshape(1,-1).t() * cand_boxes, dim=0)
            new_score = torch.sum(weights * cand_scores)

            D_b.append(new_box)
            D_s.append(new_score)
            D_l.append(M_l)
            D_i.append(M_i)

            indices = indices[inds_below]
            boxes = boxes[inds_below]
            scores = scores[inds_below]
            levels = levels[inds_below]

            if len(D_l) == limit:
                break

        D_l = torch.stack(D_l)
        D_b = torch.stack(D_b)
        D_s = torch.stack(D_s)
        D_i = torch.stack(D_i)
```

Listing B.3: OWN IMPLEMENTATION OF MULTI-NON-MAXIMUM SUPPRESSION PART 3

```
...
    elif method == 'soft':
        sigma = 0.5
        while boxes.shape[0]:
            indices = indices[torch.sort(scores, descending=True)[1]]
            boxes = boxes[torch.sort(scores, descending=True)[1]]
            levels = levels[torch.sort(scores, descending=True)[1]]
            scores = torch.sort(scores, descending=True)[0]

            M_b = boxes[0]
            D_b.append(M_b)
            D_s.append(scores[0])
            D_l.append(levels[0])
            D_i.append(indices[0])
            boxes = boxes[1:]
            scores = scores[1:]
            levels = levels[1:]
            indices = indices[1:]
            ious = box_ops.box_iou(M_b.unsqueeze(0), boxes)
            scores = scores * torch.exp(-(ious*ious)/sigma).squeeze()

        D_l = torch.stack(D_l)
        D_b = torch.stack(D_b)
        D_s = torch.stack(D_s)
        D_i = torch.stack(D_i)

        D_l = D_l[D_s > score_thres]
        D_b = D_b[D_s > score_thres]
        D_s = D_s[D_s > score_thres]
        D_i = D_i[D_s > score_thres]

    if len(D_s) == 0 or len(D_b) == 0:
        return torch.empty((0,), dtype=torch.int64, device=levels.device), \
               torch.empty((0, 4), dtype=torch.int64, device=boxes.device), \
               torch.empty((0,), dtype=torch.int64, device=scores.device), \
               torch.empty((0,), dtype=torch.int64, device=levels.device)

    if multi_class:
        offsets_ = D_l.to(D_b) * (max_coordinate + torch.tensor(1).to(D_b))
        D_b = D_b - offsets_[:, None]

    return D_i, D_b, D_s, D_l
```

Listing B.4: OWN IMPLEMENTATION OF MULTI-NON-MAXIMUM SUPPRESSION PART 4

B.2 Stability Evaluation

```
1
2
3  def get_transform():
4      transforms = []
5      transforms.append(T.ToTensor())
6      return T.Compose(transforms)
7
8
9  coco_dataset = CocoTransformedDetection(
10     root='/home/user/engeli/detectron2/datasets/coco/val2017',
11     annFile='/home/user/engeli/detectron2/datasets/coco/'
12             'annotations/instances_val2017.json',
13     transforms=get_transform(), x_shift=0, y_shift=0, rotation=0)
14
15 data_loader = torch.utils.data.DataLoader(
16     coco_dataset, batch_size=1, shuffle=False, num_workers=0,
17     collate_fn=utils.collate_fn)
18
19 coco_r_dataset = CocoTransformedDetection(
20     root='/home/user/engeli/detectron2/datasets/coco/val2017',
21     annFile='/home/user/engeli/detectron2/datasets/coco/'
22             'annotations/instances_val2017.json',
23     transforms=get_transform(), x_shift=0, y_shift=0, rotation=5)
24
25 data_r_loader = torch.utils.data.DataLoader(
26     coco_r_dataset, batch_size=1, shuffle=False, num_workers=0,
27     collate_fn=utils.collate_fn)
28
29 rpn = ['standard', 'soft', 'average', 'average_iou']
30 roi = ['standard', 'soft', 'average', 'average_iou']
31 all_methods = [rpn, roi]
32 methods = list(itertools.product(*all_methods))
33
34
35 def convert_to_xyxy(boxes):
36     """ Converts bounding boxes in the form [xmin, ymin, w, h] to bounding boxes with [
      xmin, ymin, xmax, ymax]
37     Args:
38         boxes (Tensor[N, 4], required): Bounding boxes in the form [xmin, ymin, w, h]
39
40     Returns:
41         Tensor[N, 4]: Bounding boxes in the form [xmin, ymin, xmax, ymax]
42     """
43     return torch.transpose(
44         torch.stack([boxes[:, 0], boxes[:, 1], boxes[:, 0] + boxes[:, 2],
45                      boxes[:, 1] + boxes[:, 3]]), 0, 1)
```

Listing B.5: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 2

```
...
def backward_transform(image_id, boxes_dt, boxes_gt):
    """ Inverse transforms detection and ground truth boxes of already transformed boxes
        by considering image size with associated image_id
    Args:
        image_id (int, required): Image ID that the bounding boxes are associated to
        boxes_dt (Tensor[N, 4], required): Detected bounding boxes in the form [xmin, ymin, xmax, ymax]
        boxes_gt (Tensor[N, 4], required): Ground truth bounding boxes in the form [xmin, ymin, xmax, ymax]
    Returns:
        tuple(boxes_dt, boxes_gt): Inverse transformed detection boxes (Tensor[N, 4])
        and ground truth boxes (Tensor[N, 4]), both in the form in the form [xmin, ymin, xmax, ymax]
    """
    if torch.unique(torch.nonzero(boxes_dt)[:, 0]).numel() == 0:
        return torch.zeros(1, 4), torch.zeros(1, 4)
    else:
        img_o_shape = (coco_dataset.coco.imgs[image_id]['height'],
                       coco_dataset.coco.imgs[image_id]['width'])
        r_height = int(img_o_shape[0] * np.cos(np.deg2rad(5)) +
                       img_o_shape[1] * np.sin(np.deg2rad(5)))
        r_width = int(img_o_shape[0] * np.sin(np.deg2rad(5)) +
                      img_o_shape[1] * np.cos(np.deg2rad(5)))
        img_shape = (r_height, r_width)
        img = torch.ones((img_shape[0], img_shape[1], 3))
        boxes_dt = box_ops.clip_boxes_to_image(boxes_dt, img_shape)
        boxes_gt = box_ops.clip_boxes_to_image(boxes_gt, img_shape)
        factor = torch.tensor(img_shape) / torch.tensor(img_o_shape)
        boxes_dt = torch.cat((boxes_dt, torch.ones(
            boxes_dt.shape[0]).unsqueeze(1)), 1)
        boxes_gt = torch.cat((boxes_gt, torch.zeros(
            boxes_gt.shape[0]).unsqueeze(1)), 1)
        boxes = torch.cat((boxes_dt, boxes_gt))
        transform = A.Compose([
            A.augmentations.geometric.transforms.Affine(
                scale={'x': float(factor[1]), 'y': float(factor[0])}, rotate=-5,
                fit_output=False, mode=1, always_apply=True)],
            bbox_params=A.BboxParams(format='pascal_voc', min_visibility=0.3))
        transformed = transform(image=np.array(img) * 255, bboxes=boxes)
        dt, gt = [], []
        for box in transformed['bboxes']:
            if int(box[4]) == 0:
                xmin, ymin, xmax, ymax = box[:4]
                gt.append(torch.tensor([xmin / factor[1], ymin / factor[0],
                                        xmax / factor[1], ymax / factor[0]]))
            if int(box[4]) == 1:
                xmin, ymin, xmax, ymax = box[:4]
                dt.append(torch.tensor([xmin / factor[1], ymin / factor[0],
                                        xmax / factor[1], ymax / factor[0]]))
        if len(dt) == 0:
            return torch.zeros(1, 4), torch.zeros(1, 4)
        else:
            return torch.stack(dt), torch.stack(gt)
```

Listing B.6: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 2

```
...
def get_e_x_y(gt_boxes, dt_boxes):
    """ Calculates deviation of the detection from center position in x- and y-dimension
     from the ground truth
    Args:
        gt_boxes (Tensor[N, 4], required): Detected bounding boxes in the form [xmin,
    ymin, xmax, ymax]
        dt_boxes (Tensor[N, 4], required): Ground truth bounding boxes in the form [xmin
    , ymin, xmax, ymax]

    Returns:
        tuple(center_deviation_x, center_deviation_y): center_deviation_x and
    center_deviation_y as Tensor[N]
    """
    if len(dt_boxes.shape) == 1:
        gt_boxes = gt_boxes.unsqueeze(0)
        dt_boxes = dt_boxes.unsqueeze(0)
    w = gt_boxes[:, 2] - gt_boxes[:, 0]
    h = gt_boxes[:, 3] - gt_boxes[:, 1]
    x_c_d = dt_boxes[:, 0] + 0.5 * (dt_boxes[:, 2] - dt_boxes[:, 0])
    y_c_d = dt_boxes[:, 1] + 0.5 * (dt_boxes[:, 3] - dt_boxes[:, 1])
    x_c_g = gt_boxes[:, 0] + 0.5 * (gt_boxes[:, 2] - dt_boxes[:, 0])
    y_c_g = gt_boxes[:, 1] + 0.5 * (gt_boxes[:, 3] - dt_boxes[:, 1])
    return (x_c_d - x_c_g) / w, (y_c_d - y_c_g) / h


def get_e_s_r(gt_boxes, dt_boxes):
    """ Calculates deviation from scale and aspect ratio of the detection compared to
    the ground truth
    Args:
        gt_boxes (Tensor[N, 4], required): Detected bounding boxes in the form [xmin,
    ymin, xmax, ymax]
        dt_boxes (Tensor[N, 4], required): Ground truth bounding boxes in the form [xmin
    , ymin, xmax, ymax]

    Returns:
        tuple(scale_deviation_x, ratio_deviation_y): scale_deviation_x and
    ratio_deviation_y as Tensor[N]
    """
    if len(dt_boxes.shape) == 1:
        gt_boxes = gt_boxes.unsqueeze(0)
        dt_boxes = dt_boxes.unsqueeze(0)
    w_g = gt_boxes[:, 2] - gt_boxes[:, 0]
    h_g = gt_boxes[:, 3] - gt_boxes[:, 1]
    w_d = dt_boxes[:, 2] - dt_boxes[:, 0]
    h_d = dt_boxes[:, 3] - dt_boxes[:, 1]
    s = (w_d * h_d) / (w_g * h_g)
    return torch.tensor([torch.sqrt(s_i) for s_i in s]), \
           (w_d / h_d) / (w_g / h_g)
```

Listing B.7: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 3

```
...
def get_sigmas(frame, m):
    """ Calculates standard deviations for center deviations and scale and ratio
    deviations.
        And calculates the fragment error.
    Args:
        frame (dict of str: Tensor[N, 4], required): Keys in ['default_gt_boxes', '
    x_gt_boxes', 'y_gt_boxes',
            'r_gt_boxes','default_boxes', 'x_boxes', 'y_boxes', 'r_boxes']
        m (Tensor[4, N], required): Matrix indicating the presence or absence of every
    unique detection in every frame
    Returns:
        tuple(sigma_x, sigma_y, sigma_s, sigma_r, fes): Each element of the tuble has
    the format Tensor[N]
    """
    gts = ['default_gt_boxes', 'x_gt_boxes', 'y_gt_boxes', 'r_gt_boxes']
    dts = ['default_boxes', 'x_boxes', 'y_boxes', 'r_boxes']
    e_xs = []
    e_ys = []
    e_ss = []
    e_rs = []
    sigma_x = []
    sigma_y = []
    sigma_s = []
    sigma_r = []
    for g, d in zip(gts, dts):
        num_el = torch.unique(torch.nonzero(frame[d])[:, 0]).numel()
        max_el = int(m.shape[1])
        if not num_el:
            e_x, e_y, e_s, e_r = torch.ones(4, max_el) * float('Nan')
        else:
            e_x, e_y = get_e_x_y(frame[g], frame[d])
            e_s, e_r = get_e_s_r(frame[g], frame[d])
        e_xs.append(e_x)
        e_ys.append(e_y)
        e_ss.append(e_s)
        e_rs.append(e_r)
    e_x = torch.stack(e_xs)
    e_y = torch.stack(e_ys)
    e_s = torch.stack(e_ss)
    e_r = torch.stack(e_rs)
```

Listing B.8: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 4

```
...
    for i in range(e_x.shape[1]):
        if len(e_x[list(torch.isnan(e_x[:, i]) == False), i]) > 1:
            sigma_x.append(statistics.stdev(
                [float(x) for x in e_x[:, i] if not torch.isnan(x)]))
            sigma_y.append(statistics.stdev(
                [float(y) for y in e_y[:, i] if not torch.isnan(y)]))
            sigma_s.append(statistics.stdev(
                [float(s) for s in e_s[:, i] if not torch.isnan(s)]))
            sigma_r.append(statistics.stdev(
                [float(r) for r in e_r[:, i] if not torch.isnan(r)]))
        else:
            sigma_x.append(
                [float(x * 0) for x in e_x[:, i] if not torch.isnan(x)][0])
            sigma_y.append(
                [float(y * 0) for y in e_y[:, i] if not torch.isnan(y)][0])
            sigma_s.append(
                [float(s * 0) for s in e_s[:, i] if not torch.isnan(s)][0])
            sigma_r.append(
                [float(r * 0) for r in e_r[:, i] if not torch.isnan(r)][0])

    m = m.transpose(0, 1)
    m[m > 0] = 1
    fes = []
    for element in m:
        f_e = 0
        for e in range(1, 4):
            if element[e] != element[e - 1]:
                f_e += 1
        fes.append(f_e / 3)

    return torch.tensor(sigma_x).squeeze(), torch.tensor(sigma_y).squeeze(), \
           torch.tensor(sigma_s).squeeze(), torch.tensor(sigma_r).squeeze(), torch.
    tensor(fes)
```

Listing B.9: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 5

```
...
if __name__ == '__main__':
    for m in tqdm(range(len(methods))):
        rpn, roi = methods[m]
        if rpn == 'soft':
            rpn_thr = 'score_thres_0.01'
        else:
            rpn_thr = 'default'
        if roi == 'soft':
            roi_thr = 'score_thres_0.1'
        else:
            roi_thr = 'default'
        res_d_file = '/home/user/engeli/stabilizing-nms/fasterrcnn/coco_results/accuracy/'
            \'coco_resnet50_rpn_{}_{}_roi_{}_{}_x_default_y_default_rot_default.h5'.format(
                rpn, rpn_thr, roi, roi_thr)
        res_x_file = '/home/user/engeli/stabilizing-nms/fasterrcnn/coco_results/accuracy/'
            \'coco_resnet50_rpn_{}_{}_roi_{}_{}_x_20_y_default_rot_default.h5'.format(
                rpn, rpn_thr, roi, roi_thr)
        res_y_file = '/home/user/engeli/stabilizing-nms/fasterrcnn/coco_results/accuracy/'
            \'coco_resnet50_rpn_{}_{}_roi_{}_{}_x_default_y_20_rot_default.h5'.format(
                rpn, rpn_thr, roi, roi_thr)
        res_r_file = '/home/user/engeli/stabilizing-nms/fasterrcnn/coco_results/accuracy/' \
                        'coco_resnet50_rpn_{}_{}_roi_{}_{}_x_default_y_default_rot_5.h5'.format(
                rpn, rpn_thr, roi, roi_thr)
        _, _, _, results_d = dd.io.load(res_d_file)
        _, _, _, results_x = dd.io.load(res_x_file)
        _, _, _, results_y = dd.io.load(res_y_file)
        _, _, _, results_r = dd.io.load(res_r_file)
        stability_results = dict()
        frames = []
        N = 0
        results_d = {d['image_id']: d for d in results_d if d['aRng'] == [0, 10000000000.0]}
        results_x = {x['image_id']: x for x in results_x if x['aRng'] == [0, 10000000000.0]}
        results_y = {y['image_id']: y for y in results_y if y['aRng'] == [0, 10000000000.0]}
        results_r = {r['image_id']: r for r in results_r if r['aRng'] == [0, 10000000000.0]}
        indices_d = list(results_d.keys())
        indices_x = list(results_x.keys())
        indices_y = list(results_y.keys())
        indices_r = list(results_r.keys())
        lead_indices = list(set().union(indices_d, indices_x, indices_y, indices_r))
```

Listing B.10: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 6

```
...for i in tqdm(range(len(lead_indices))):
            frame = dict()
            image_id = lead_indices[i]
            frame['image_id'] = image_id
            if lead_indices[i] in indices_d:
                len_d = len(results_d[lead_indices[i]]['dtBoxes'])
                if len_d > 0:
                    boxes_d_dt = convert_to_xyxy(torch.stack(
                        results_d[lead_indices[i]]['dtBoxes']))
                    boxes_d_gt = convert_to_xyxy(torch.tensor(
                        results_d[lead_indices[i]]['gtBoxes']))
                else:
                    boxes_d_dt, boxes_d_gt = torch.zeros(1, 4),torch.zeros(1, 4)
            else:
                len_d = 0
            if lead_indices[i] in indices_x:
                len_x = len(results_x[lead_indices[i]]['dtBoxes'])
                if len_x > 0:
                    boxes_x_dt = convert_to_xyxy(torch.stack(
                        results_x[lead_indices[i]]['dtBoxes']))
                    boxes_x_gt = convert_to_xyxy(torch.tensor(
                        results_x[lead_indices[i]]['gtBoxes']))
                else:
                    boxes_d_dt, boxes_d_gt = torch.zeros(1, 4),torch.zeros(1, 4)
            else:
                len_x = 0
            if lead_indices[i] in indices_y:
                len_y = len(results_y[lead_indices[i]]['dtBoxes'])
                if len_y > 0:
                    boxes_y_dt = convert_to_xyxy(torch.stack(
                        results_y[lead_indices[i]]['dtBoxes']))
                    boxes_y_gt = convert_to_xyxy(torch.tensor(
                        results_y[lead_indices[i]]['gtBoxes']))
                else:
                    boxes_d_dt, boxes_d_gt = torch.zeros(1, 4),torch.zeros(1, 4)
            else:
                len_y = 0
            if lead_indices[i] in indices_r:
                len_r = len(results_r[lead_indices[i]]['dtBoxes'])
                if len_r > 0:
                    boxes_r_dt = convert_to_xyxy(torch.stack(
                        results_r[lead_indices[i]]['dtBoxes']))
                    boxes_r_gt = convert_to_xyxy(torch.tensor(
                        results_r[lead_indices[i]]['gtBoxes']))
                else:
                    boxes_d_dt, boxes_d_gt = torch.zeros(1, 4),torch.zeros(1, 4)
            else:
                len_r = 0
            frame['default_gt_boxes'] = boxes_d_gt
            frame['x_gt_boxes'] = boxes_x_gt
            frame['y_gt_boxes'] = boxes_y_gt
            frame['r_gt_boxes'] = boxes_r_gt
            if len_d == len_x == len_y == len_r == 0:
                continue
```

Listing B.11: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 7

```
...
all_boxes = []
            d_boxes = []
            x_boxes = []
            y_boxes = []
            r_boxes = []
            indices = []
            gts = ['default_gt_boxes', 'x_gt_boxes', 'y_gt_boxes', 'r_gt_boxes']
            dts = ['default_boxes', 'x_boxes', 'y_boxes', 'r_boxes']
            for gt_id, g in enumerate(gts):
                if g == 'default_gt_boxes':
                    for box in frame[g]:
                        d_boxes.append(box)
                        indices.append(gt_id)
                elif g == 'x_gt_boxes':
                    for box in frame[g]:
                        x_boxes.append(box - torch.tensor([20, 0, 20, 0]))
                        indices.append(gt_id)
                elif g == 'y_gt_boxes':
                    for box in frame[g]:
                        y_boxes.append(box - torch.tensor([0, 20, 0, 20]))
                        indices.append(gt_id)
                elif g == 'r_gt_boxes':
                    _, r_gt = backward_transform(image_id, frame[g], frame[g])
                    for r in r_gt:
                        r_boxes.append(r)
                        indices.append(gt_id)
            if len(d_boxes) == len(x_boxes) == len(y_boxes) == len(r_boxes) == 1:
                if torch.sum(d_boxes[0]) == 0 and torch.sum(x_boxes[0]) == -40 \
                        and torch.sum(y_boxes[0]) == -40 and torch.sum(r_boxes[0]) == 0:
                    continue

            d_boxes = torch.stack(d_boxes)
            x_boxes = torch.stack(x_boxes)
            y_boxes = torch.stack(y_boxes)
            r_boxes = torch.stack(r_boxes)
            all_boxes = torch.cat([d_boxes, x_boxes, y_boxes, r_boxes])
```

Listing B.12: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 8

```
...
all_box_dict = {all_box_id + 1: {'box': box}
                            for all_box_id, box in enumerate(all_boxes) if
                            torch.sum(box) > 0}
            for k, v in all_box_dict.items():
                all_box_dict[k]['frame_index'] = int(indices[k - 1])
            for box_id, values in all_box_dict.items():
                box = values['box']
                frame_index = values['frame_index']
                remaining_ids = []
                remaining_boxes = []
                for r_box_id, r_values in all_box_dict.items():
                    if r_values['frame_index'] != frame_index:
                        remaining_ids.append(r_box_id)
                        remaining_boxes.append(r_values['box'])
                if len(remaining_boxes) == 0:
                    all_box_dict[box_id]['matches'] = np.array([0])
                    continue
                ious = box_ops.box_iou(box.unsqueeze(0),
                                       torch.stack(remaining_boxes))
                remaining_ids = np.array(remaining_ids)
                if torch.count_nonzero(ious) > 0:
                    all_box_dict[box_id]['matches'] = remaining_ids[
                        [int(i) for i in torch.where(ious > 0.01)[1]]]
                else:
                    all_box_dict[box_id]['matches'] = np.array([0])

            m = []
            for id, value in all_box_dict.items():
                n = [0, 0, 0, 0]
                n[value['frame_index']] = id
                for match in value['matches']:
                    if match == 0:
                        continue
                    n[all_box_dict[match]['frame_index']] = match
                if n not in m:
                    m.append(n)

            dt_boxes = torch.cat((boxes_d_dt, boxes_x_dt, boxes_y_dt, boxes_r_dt), dim
    =0)
            gt_boxes = torch.cat((boxes_d_gt, boxes_x_gt, boxes_y_gt, boxes_r_gt), dim
    =0)
```

Listing B.13: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 10

```
...
m = torch.tensor(m).transpose(0, 1)

            N += m.shape[1]
            pad = torch.zeros(1, 4)
            for index, dist_det in enumerate(m):
                dt_fr = []
                gt_fr = []
                for j in dist_det:
                    if j == 0:
                        dt_fr.append(pad)
                        gt_fr.append(pad)
                    else:
                        dt_fr.append(dt_boxes[int(j - 1)].unsqueeze(0))
                        gt_fr.append(gt_boxes[int(j - 1)].unsqueeze(0))
                frame[dts[index]] = torch.cat(dt_fr)
                frame[gts[index]] = torch.cat(gt_fr)

            if len(frame) > 1:
                frame['sigma_x'], frame['sigma_y'], frame['sigma_s'], frame['sigma_r'], frame['ef'] = get_sigmas(frame,

                                m)
            frames.append(frame)

        stability_results['N'] = N
        e_c = 0
        e_r = 0
        e_f = 0
        for frame in frames:
            if len(frame) > 1:
                e_f += float(torch.sum(frame['ef']))
                e_c += torch.sum(frame['sigma_x'] + frame['sigma_y'])
                e_r += torch.sum(frame['sigma_s'] + frame['sigma_r'])
        stability_results['E_f'] = float(e_f / N)
        stability_results['E_c'] = float(e_c / N)
        stability_results['E_r'] = float(e_r / N)
        stability_results['SE'] = stability_results['E_f'] + stability_results['E_c'] + stability_results['E_r']
        log_file = '/home/user/engeli/stabilizing-nms/fasterrcnn/coco_results/stability/coco_rpn_{}_{}_roi_{}_{}.log'.format(
            rpn, rpn_thr, roi, roi_thr)
        log = open(log_file, "a")
        orig_stdout = sys.stdout
        sys.stdout = log
        print(rpn, rpn_thr, roi, roi_thr)
        print(stability_results['N'])
        print(stability_results['E_f'])
        print(stability_results['E_c'])
        print(stability_results['E_r'])
        print(stability_results['SE'])
        log.close()
        sys.stdout = orig_stdout
        dd.io.save(log_file.replace('log', 'h5'), [frames, stability_results])
```

Listing B.14: OWN IMPLEMENTATION OF THE STABILITY EVALUATION PART 11

Appendix C

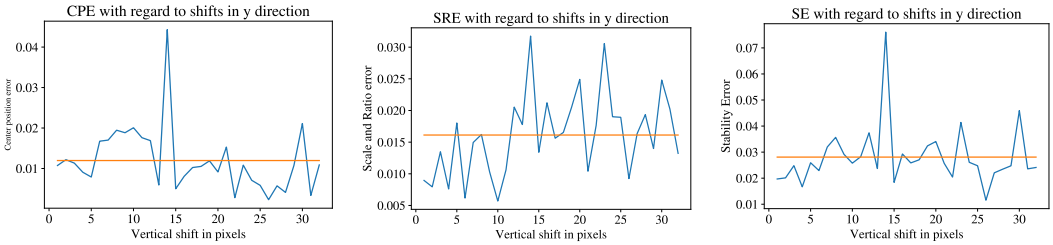
Extended Analysis

(a) Faster R-CNN center position error for $t_x \in P$

(b) Faster R-CNN scale and ratio error for $t_x \in P$

(c) Faster R-CNN stability error for $t_x \in P$

Figure C.1: FASTER R-CNN STABILITY ERROR BREAKDOWN FOR HORIZONTAL TRANSLATION. The stability error breakdown for the horizontal translational parameter. (a) shows the center position error (y-axis) over an increase of horizontal translation (x-axis) in pixels. (b) shows the scale and ratio error (y-axis) over an increase of horizontal translation (x-axis) in pixels. (c) shows the stability error (y-axis) over an increase of horizontal translation (x-axis) in pixels. The fragment error is constantly 0 and therefore not shown in the graph due to the missing contribution to the stability error. The horizontal orange line in all three graphs indicates the mean stability error for the respective curve.

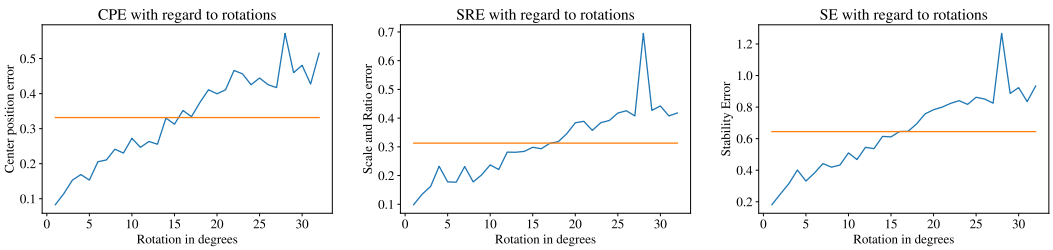


(a) Faster R-CNN center position error for $t_y \in P$

(b) Faster R-CNN scale and ratio error for $t_y \in P$

(c) Faster R-CNN stability error for $t_y \in P$

Figure C.2: FASTER R-CNN STABILITY ERROR BREAKDOWN FOR VERTICAL TRANSLATION. The stability error breakdown for the vertical translational parameter. (a) shows the center position error (y-axis) over an increase of vertical translation (x-axis) in pixels. (b) shows the scale and ratio error (y-axis) over an increase of vertical translation (x-axis) in pixels. (c) shows the stability error (y-axis) over an increase of vertical translation (x-axis) in pixels. The fragment error is constantly 0 and therefore not shown in the graph due to the missing contribution to the stability error. The horizontal orange line in all three graphs indicates the mean stability error for the respective curve.

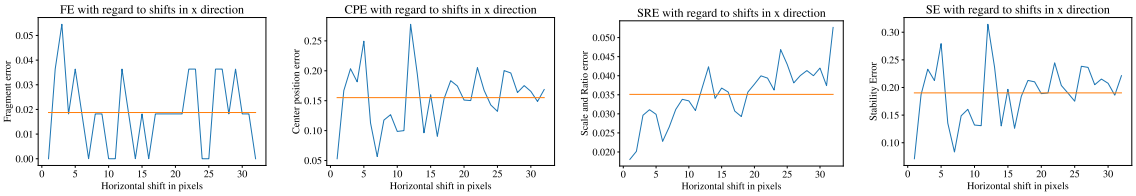


(a) Faster R-CNN center position error for $\theta \in P$

(b) Faster R-CNN scale and ratio error for $\theta \in P$

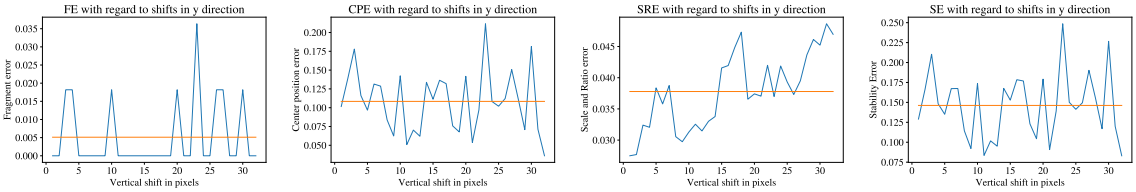
(c) Faster R-CNN stability error for $\theta \in P$

Figure C.3: FASTER R-CNN STABILITY ERROR BREAKDOWN FOR ROTATION. The stability error breakdown for the rotational parameter. (a) shows the center position error (y-axis) over an increase of rotation (x-axis) in degrees. (b) shows the scale and ratio error (y-axis) over an increase of rotation (x-axis) in degrees. (c) shows the stability error (y-axis) over an increase of rotation (x-axis) in degrees. The fragment error is constantly 0 and therefore not shown in the graph due to the missing contribution to the stability error. The horizontal orange line in all three graphs indicates the mean stability error for the respective curve.



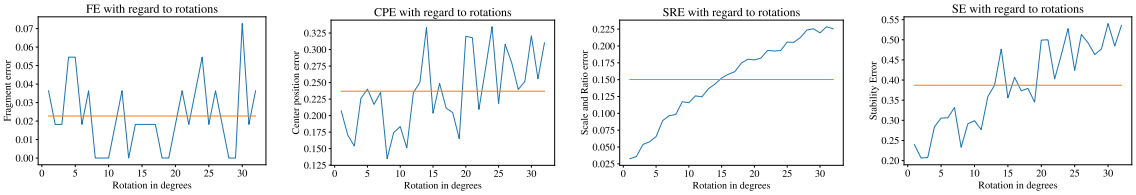
(a) MTCNN fragment error for $t_x \in P$ (b) MTCNN center position error for $t_x \in P$ (c) MTCNN scale and ratio error for $t_x \in P$ (d) MTCNN stability error for $t_x \in P$

Figure C.4: MTCNN STABILITY ERROR BREAKDOWN FOR HORIZONTAL TRANSLATION. The stability error breakdown for the horizontal translational parameter for MTCNN. (a) shows the fragment error (y-axis) over an increase of horizontal translation (x-axis) in pixels. (b) shows the center position error (y-axis) over an increase of horizontal translation (x-axis) in pixels. (c) shows the scale and ratio error (y-axis) over an increase of horizontal translation (x-axis) in pixels. (d) shows the stability error (y-axis) over an increase of horizontal translation (x-axis) in pixels. The horizontal orange line in all three graphs indicates the mean stability error for the respective curve.



(a) MTCNN fragment error for $t_y \in P$ (b) MTCNN center position error for $t_y \in P$ (c) MTCNN scale and ratio error for $t_y \in P$ (d) MTCNN stability error for $t_y \in P$

Figure C.5: MTCNN STABILITY ERROR BREAKDOWN FOR VERTICAL TRANSLATION. The stability error breakdown for the vertical translational parameter for MTCNN. (a) shows the fragment error (y-axis) over an increase of vertical translation (x-axis) in pixels. (b) shows the center position error (y-axis) over an increase of vertical translation (x-axis) in pixels. (c) shows the scale and ratio error (y-axis) over an increase of vertical translation (x-axis) in pixels. (d) shows the stability error (y-axis) over an increase of vertical translation (x-axis) in pixels. The horizontal orange line in all three graphs indicates the mean stability error for the respective curve.



(a) MTCNN fragment error for $\theta \in P$ (b) MTCNN center position error for $\theta \in P$ (c) MTCNN scale and ratio error for $\theta \in P$ (d) MTCNN stability error for $\theta \in P$

Figure C.6: MTCNN STABILITY ERROR BREAKDOWN FOR ROTATION. The stability error breakdown for the rotational parameter for MTCNN. (a) shows the fragment error (y-axis) over an increase of rotation (x-axis) in degrees. (b) shows the center position error (y-axis) over an increase of rotation (x-axis) in degrees. (c) shows the scale and ratio error (y-axis) over an increase of rotation (x-axis) in degrees. (d) shows the stability error (y-axis) over an increase of rotation (x-axis) in degrees. The horizontal orange line in all three graphs indicates the mean stability error for the respective curve.

Appendix D

Extended Results

NMS P-Net	NMS R-Net	NMS O-Net	AP	AP^{50}	AP^{f_1}	$AP^{50}_{f_1}$	AP^{f_2}	$AP^{50}_{f_2}$	AP^{f_3}	$AP^{50}_{f_3}$	AP^{f_4}	$AP^{50}_{f_4}$
Classic	Classic	Classic	26.73	47.35	28.00	47.40	22.60	47.20	28.20	**47.40**	28.10	47.40
Classic	Classic	Soft	4.93	8.93	5.00	9.00	4.20	8.60	5.40	9.10	5.10	9.00
Classic	Classic	Average	26.48	46.80	28.00	46.80	21.80	46.80	28.00	46.80	28.10	46.80
Classic	Classic	Avg. IoU	24.65	44.58	26.00	44.60	20.50	44.50	26.10	44.60	26.00	44.60
Classic	Soft	Classic	11.90	22.10	12.40	22.20	10.10	21.90	12.50	22.20	12.60	22.10
Classic	Soft	Soft	7.63	13.85	7.90	13.70	6.40	13.50	7.90	13.60	8.30	14.60
Classic	Soft	Average	11.85	22.00	12.30	22.10	10.00	21.80	12.50	22.10	12.60	22.00
Classic	Soft	Avg. IoU	11.43	21.60	11.90	21.70	9.80	21.40	12.10	21.80	11.90	21.50
Classic	Average	Classic	26.83	47.33	28.00	47.30	22.70	47.20	**28.50**	**47.40**	28.10	47.40
Classic	Average	Soft	4.93	8.95	5.10	9.10	4.30	8.90	5.30	8.90	5.00	8.90
Classic	Average	Average	26.55	46.80	28.00	46.80	21.90	46.70	28.10	46.80	28.20	46.90
Classic	Average	Avg. IoU	24.63	44.48	26.00	44.50	20.50	44.50	26.00	44.50	26.00	44.40
Classic	Avg. IoU	Classic	26.80	47.35	28.10	47.30	22.70	**47.30**	28.30	**47.40**	28.10	47.40
Classic	Avg. IoU	Soft	4.88	8.83	5.10	9.10	4.20	8.70	5.30	8.80	4.90	8.70
Classic	Avg. IoU	Average	26.58	46.95	28.10	47.50	21.90	46.70	28.10	46.80	28.20	46.80
Classic	Avg. IoU	Avg. IoU	24.60	44.63	26.00	45.20	20.40	44.40	26.00	44.50	26.00	44.40
Soft	Classic	Classic	9.85	17.00	10.50	16.80	8.10	16.70	10.50	17.70	10.30	16.80
Soft	Classic	Soft	3.95	7.13	4.30	7.10	3.50	7.30	3.80	7.00	4.20	7.10
Soft	Classic	Average	9.95	17.18	10.60	17.70	8.10	16.60	10.60	17.60	10.50	16.80
Soft	Classic	Avg. IoU	9.75	16.65	10.40	16.70	8.10	16.60	10.30	16.70	10.20	16.60
Soft	Soft	Classic	6.93	12.33	7.30	12.60	5.80	11.60	7.30	12.60	7.30	12.50
Soft	Soft	Soft	4.53	8.15	4.90	8.70	3.60	7.60	4.70	8.00	4.90	8.30
Soft	Soft	Average	7.00	12.38	7.30	12.60	5.70	11.60	7.40	12.60	7.60	12.70
Soft	Soft	Avg. IoU	6.85	12.25	7.30	12.50	5.60	11.50	7.20	12.50	7.30	12.50
Soft	Average	Classic	9.88	16.95	10.70	17.70	8.10	16.60	10.30	16.70	10.40	16.80
Soft	Average	Soft	3.98	7.15	4.10	7.10	3.50	7.20	3.90	7.10	4.40	7.20
Soft	Average	Average	9.85	16.70	10.60	16.70	8.10	16.60	10.40	16.70	10.30	16.80
Soft	Average	Avg. IoU	9.70	16.63	10.30	16.70	8.00	16.50	10.30	16.70	10.20	16.60
Soft	Avg. IoU	Classic	9.83	16.93	10.70	17.70	8.00	16.60	10.20	16.70	10.40	16.70
Soft	Avg. IoU	Soft	3.95	7.13	4.10	7.00	3.50	7.30	3.90	7.00	4.30	7.20
Soft	Avg. IoU	Average	9.88	16.93	10.70	17.70	8.00	16.60	10.40	16.70	10.40	16.70
Soft	Avg. IoU	Avg. IoU	9.70	16.65	10.30	16.70	8.00	16.50	10.30	16.70	10.20	16.70
Average	Classic	Classic	26.53	47.20	28.10	47.40	22.00	46.70	28.00	47.30	28.00	47.40
Average	Classic	Soft	5.05	9.00	5.00	8.90	4.60	9.00	5.40	9.00	5.20	9.10
Average	Classic	Average	26.50	46.73	28.10	46.80	22.00	46.70	28.00	46.60	27.90	46.80
Average	Classic	Avg. IoU	24.78	44.73	26.30	44.60	20.60	44.70	26.20	44.80	26.00	44.80
Average	Soft	Classic	11.70	21.40	12.40	21.30	9.90	21.10	12.50	22.00	12.00	21.20
Average	Soft	Soft	7.53	13.65	7.80	13.30	6.50	13.90	8.10	14.00	7.70	13.40
Average	Soft	Average	11.60	21.50	12.30	22.10	9.80	21.00	12.40	21.90	11.90	21.00
Average	Soft	Avg. IoU	11.15	20.88	11.80	20.90	9.40	20.40	12.00	21.60	11.40	20.60
Average	Average	Classic	26.83	47.33	28.30	47.40	22.70	47.20	28.10	**47.40**	28.20	47.30
Average	Average	Soft	5.03	8.90	5.40	9.10	4.50	9.00	5.10	8.80	5.10	8.70
Average	Average	Average	26.53	46.78	28.10	46.80	21.90	46.80	28.20	46.80	27.90	46.70
Average	Average	Avg. IoU	24.60	44.58	25.90	44.30	20.50	44.50	26.20	44.90	25.80	44.60
Average	Avg. IoU	Classic	**26.88**	47.30	28.30	47.40	**22.80**	47.20	28.20	**47.40**	28.20	47.20
Average	Avg. IoU	Soft	5.08	8.88	5.30	9.00	4.50	8.80	5.20	8.90	5.30	8.80
Average	Avg. IoU	Average	26.58	46.70	28.10	46.70	22.00	46.70	28.20	46.80	28.00	46.60
Average	Avg. IoU	Avg. IoU	24.70	44.50	25.90	44.20	20.70	44.60	26.20	44.70	26.00	44.50
Avg. IoU	Classic	Classic	26.78	47.35	28.20	**48.30**	22.40	46.40	28.30	**47.40**	28.20	47.30
Avg. IoU	Classic	Soft	4.95	9.05	5.10	9.20	4.20	8.80	5.40	9.20	5.10	9.00
Avg. IoU	Classic	Average	26.53	46.83	28.20	47.80	21.80	46.00	28.20	46.80	27.90	46.70
Avg. IoU	Classic	Avg. IoU	24.80	44.93	26.30	45.70	20.60	44.70	26.20	44.70	26.10	44.60
Avg. IoU	Soft	Classic	11.25	21.03	11.70	21.00	9.60	20.90	11.80	21.10	11.90	21.10
Avg. IoU	Soft	Soft	7.13	13.00	7.30	12.70	6.30	13.50	7.50	13.00	7.40	12.80
Avg. IoU	Soft	Average	11.20	20.90	11.60	20.90	9.40	20.70	12.00	21.00	11.80	21.00
Avg. IoU	Soft	Avg. IoU	10.83	20.55	11.20	20.60	9.10	20.30	11.60	20.60	11.40	20.70
Avg. IoU	Average	Classic	26.83	47.33	28.30	48.20	22.50	46.40	28.30	**47.40**	28.20	47.30
Avg. IoU	Average	Soft	4.98	8.88	5.10	9.10	4.30	8.80	5.30	8.90	5.20	8.70
Avg. IoU	Average	Average	26.60	46.95	28.30	47.70	22.00	46.70	28.20	46.80	27.90	46.60
Avg. IoU	Average	Avg. IoU	24.68	44.73	26.10	45.50	20.50	44.50	26.20	44.60	25.90	44.30
Avg. IoU	Avg. IoU	Classic	**26.88**	**47.73**	**28.40**	48.20	22.60	47.20	28.20	**47.40**	**28.30**	**48.10**
Avg. IoU	Avg. IoU	Soft	4.80	8.83	5.10	9.20	4.00	8.70	5.20	8.80	4.90	8.60
Avg. IoU	Avg. IoU	Average	26.63	47.15	28.30	47.70	22.00	46.80	28.10	46.70	28.10	47.40
Avg. IoU	Avg. IoU	Avg. IoU	24.68	44.90	26.10	45.40	20.60	44.60	26.00	44.50	26.00	45.10

Table D.1: MTCNN ACCURACY EVALUATION RESULTS. Accuracy evaluation results for the WIDER Face dataset with MTCNN

NMS P-Net	NMS R-Net	NMS O-Net	AP	AP^{50}	SE	FE	CPE	SRE	Number of trajectories
Classic	Classic	Classic	26.73	47.35	0.1853	0.0639	0.0485	0.0729	2937
Classic	Classic	Soft	4.93	8.93	0.4096	0.3035	0.0434	0.0626	2428
Classic	Classic	Average	26.48	46.80	0.1637	0.0637	0.0345	0.0654	2939
Classic	Classic	Avg. IoU	24.65	44.58	0.1696	0.0636	0.0378	0.0681	2938
Classic	Soft	Classic	11.90	22.10	0.5480	0.4243	0.0559	0.0679	2108
Classic	Soft	Soft	7.63	13.85	0.5543	0.4293	0.0566	0.0684	2085
Classic	Soft	Average	11.85	22.00	0.5439	0.4231	0.0542	0.0667	2106
Classic	Soft	Avg. IoU	11.43	21.60	0.5445	0.4228	0.0545	0.0672	2102
Classic	Average	Classic	26.83	47.33	0.1895	0.0655	0.0502	0.0737	2940
Classic	Average	Soft	4.93	8.95	0.4169	0.3081	0.0456	0.0632	2417
Classic	Average	Average	26.55	46.80	0.1652	0.0647	0.0350	0.0655	2938
Classic	Average	Avg. IoU	24.63	44.48	0.1720	0.0647	0.0387	0.0686	2940
Classic	Avg. IoU	Classic	26.80	47.35	0.1901	0.0659	0.0505	0.0737	**2944**
Classic	Avg. IoU	Soft	4.88	8.83	0.4192	0.3072	0.0463	0.0657	2409
Classic	Avg. IoU	Average	26.58	46.95	0.1660	0.0646	0.0353	0.0661	2942
Classic	Avg. IoU	Avg. IoU	24.60	44.63	0.1724	0.0647	0.0388	0.0689	2942
Soft	Classic	Classic	9.85	17.00	0.5352	0.4470	0.0360	0.0522	1704
Soft	Classic	Soft	3.95	7.13	0.5233	0.4469	**0.0309**	**0.0456**	1333
Soft	Classic	Average	9.95	17.18	0.5293	0.4468	0.0328	0.0497	1727
Soft	Classic	Avg. IoU	9.75	16.65	0.5360	0.4544	0.0325	0.0491	1724
Soft	Soft	Classic	6.93	12.33	0.5776	0.4777	0.0457	0.0543	1388
Soft	Soft	Soft	4.53	8.15	0.5690	0.4725	0.0444	0.0521	1334
Soft	Soft	Average	7.00	12.38	0.5771	0.4788	0.0444	0.0539	1409
Soft	Soft	Avg. IoU	6.85	12.25	0.5784	0.4812	0.0442	0.0530	1404
Soft	Average	Classic	9.88	16.95	0.5365	0.4500	0.0357	0.0509	1712
Soft	Average	Soft	3.98	7.15	0.5290	0.4492	0.0323	0.0475	1323
Soft	Average	Average	9.85	16.70	0.5274	0.4436	0.0333	0.0505	1702
Soft	Average	Avg. IoU	9.70	16.63	0.5342	0.4513	0.0331	0.0497	1712
Soft	Avg. IoU	Classic	9.83	16.93	0.5382	0.4501	0.0362	0.0520	1710
Soft	Avg. IoU	Soft	3.95	7.13	0.5347	0.4556	0.0322	0.0469	1322
Soft	Avg. IoU	Average	9.88	16.93	0.5369	0.4534	0.0336	0.0499	1718
Soft	Avg. IoU	Avg. IoU	9.70	16.65	0.5307	0.4475	0.0330	0.0502	1705
Average	Classic	Classic	26.53	47.20	0.1864	0.0646	0.0484	0.0733	2925
Average	Classic	Soft	5.05	9.00	0.4199	0.3119	0.0441	0.0639	2428
Average	Classic	Average	26.50	46.73	**0.1629**	0.0637	0.0339	0.0653	2925
Average	Classic	Avg. IoU	24.78	44.73	0.1696	**0.0636**	0.0375	0.0685	2924
Average	Soft	Classic	11.70	21.40	0.5506	0.4264	0.0570	0.0672	2125
Average	Soft	Soft	7.53	13.65	0.5562	0.4329	0.0561	0.0672	2106
Average	Soft	Average	11.60	21.50	0.5477	0.4248	0.0560	0.0669	2114
Average	Soft	Avg. IoU	11.15	20.88	0.5476	0.4252	0.0555	0.0669	2119
Average	Average	Classic	26.83	47.33	0.1902	0.0667	0.0497	0.0738	2933
Average	Average	Soft	5.03	8.90	0.4248	0.3131	0.0466	0.0651	2418
Average	Average	Average	26.53	46.78	0.1663	0.0659	0.0347	0.0656	2932
Average	Average	Avg. IoU	24.60	44.58	0.1730	0.0657	0.0386	0.0687	2933
Average	Avg. IoU	Classic	**26.88**	47.30	0.1916	0.0674	0.0502	0.0740	2939
Average	Avg. IoU	Soft	5.08	8.88	0.4131	0.3004	0.0468	0.0659	2406
Average	Avg. IoU	Average	26.58	46.70	0.1669	0.0663	0.0349	0.0657	2936
Average	Avg. IoU	Avg. IoU	24.70	44.50	0.1727	0.0655	0.0385	0.0687	2936
Avg. IoU	Classic	Classic	26.78	47.35	0.1893	0.0681	0.0485	0.0726	2932
Avg. IoU	Classic	Soft	4.95	9.05	0.4149	0.3053	0.0452	0.0643	2417
Avg. IoU	Classic	Average	26.53	46.83	0.1662	0.0665	0.0343	0.0654	2932
Avg. IoU	Classic	Avg. IoU	24.80	44.93	0.1733	0.0666	0.0383	0.0684	2932
Avg. IoU	Soft	Classic	11.25	21.03	0.5660	0.4382	0.0590	0.0689	2082
Avg. IoU	Soft	Soft	7.13	13.00	0.5617	0.4369	0.0576	0.0671	2072
Avg. IoU	Soft	Average	11.20	20.90	0.5579	0.4326	0.0573	0.0679	2078
Avg. IoU	Soft	Avg. IoU	10.83	20.55	0.5576	0.4328	0.0571	0.0677	2081
Avg. IoU	Average	Classic	26.83	47.33	0.1941	0.0700	0.0505	0.0736	2939
Avg. IoU	Average	Soft	4.98	8.88	0.4241	0.3135	0.0458	0.0648	2440
Avg. IoU	Average	Average	26.60	46.95	0.1698	0.0684	0.0353	0.0662	2939
Avg. IoU	Average	Avg. IoU	24.68	44.73	0.1762	0.0682	0.0391	0.0689	2939
Avg. IoU	Avg. IoU	Classic	**26.88**	**47.73**	0.1935	0.0683	0.0512	0.0741	2940
Avg. IoU	Avg. IoU	Soft	4.80	8.83	0.4328	0.3204	0.0465	0.0659	2443
Avg. IoU	Avg. IoU	Average	26.63	47.15	0.1684	0.0666	0.0355	0.0663	2939
Avg. IoU	Avg. IoU	Avg. IoU	24.68	44.90	0.1752	0.0663	0.0395	0.0693	2939

Table D.2: MTCNN STABILITY EVALUATION RESULTS. Stability evaluation results for the WIDER Face dataset with MTCNN

List of Figures

List of Tables

List of Listings

Bibliography

Bodla, N., Singh, B., Chellappa, R., and Davis, L. S. (2017). Improving Object Detection With One Line of Code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569.

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. (2020). Albumentations: Fast and Flexible Image Augmentations. *Information*, 11(2).

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338.

Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Gomaa, A., Abdelwahab, M. M., Abo-Zahhad, M., Minematsu, T., and Taniguchi, R.-i. (2019). Robust Vehicle Detection and Counting Algorithm Employing a Convolution Neural Network and Optical Flow. *Sensors*, 19(20):4588.

Gonzalez, R. C., Woods, R. E., et al. (2002). Digital image processing.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, Y., Zhang, X., Savvides, M., and Kitani, K. (2018). Softer-NMS: Rethinking Bounding Box Regression for Accurate Object Detection. *arXiv preprint arXiv:1809.08545*, 2:3.

Hoiem, D., Chodpathumwan, Y., and Dai, Q. (2012). Diagnosing error in object detectors. In *European conference on computer vision*, pages 340–353. Springer.

Hosang, J., Benenson, R., Dollár, P., and Schiele, B. (2015). What makes for effective detection proposals? *IEEE transactions on pattern analysis and machine intelligence*, 38(4):814–830.

Jaccard, P. (1901). Etude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017a). Feature Pyramid Networks for Object Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017b). Focal Loss for Dense Object Detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In *European conference on computer vision*, pages 740–755. Springer.

Ning, C., Zhou, H., Song, Y., and Tang, J. (2017). Inception Single Shot Multibox Detector for Object Detection. In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 549–554. IEEE.

Padilla, R., Netto, S. L., and da Silva, E. A. (2020). A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242. IEEE.

Ranjan, R., Patel, V. M., and Chellappa, R. (2017). HyperFace: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition. *IEEE transactions on pattern analysis and machine intelligence*, 41(1):121–135.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in neural information processing systems*, 28:91–99.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv preprint arXiv:1312.6229*.

Solovyev, R., Wang, W., and Gabruseva, T. (2021). Weighted boxes fusion: Ensembling boxes from different object detection models. *Image and Vision Computing*, 107:104117.

Sundararaman, R., De Almeida Braga, C., Marchand, E., and Pettre, J. (2021). Tracking Pedestrian Heads in Dense Crowd. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3865–3875.

Tan, Z., Nie, X., Qian, Q., Li, N., and Li, H. (2019). Learning to rank proposals for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8273–8281.

Tripathi, R., Singla, V., Najibi, M., Singh, B., Sharma, A., and Davis, L. (2020). ASAP-NMS: Accelerating Non-Maximum Suppression Using Spatially Aware Priors. *arXiv preprint arXiv:2007.09785*.

Tychsen-Smith, L. and Petersson, L. (2018). Improving Object Localization with Fitness NMS and Bounded IoU Loss. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6877–6885.

Wu, G. and Li, Y. (2021). Non-maximum suppression for object detection based on the chaotic whale optimization algorithm. *Journal of Visual Communication and Image Representation*, 74:102985.

Yang, S., Luo, P., Loy, C. C., and Tang, X. (2016). WIDER FACE: A Face Detection Benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, H. and Wang, N. (2016). On The Stability of Video Detection and Tracking. *arXiv preprint arXiv:1611.06467*.

Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, 23(10):1499–1503.

Zhou, H., Li, Z., Ning, C., and Tang, J. (2017). Cad: Scale Invariant Framework for Real-Time Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 760–768.

Zhu, H., Wei, H., Li, B., Yuan, X., and Kehtarnavaz, N. (2020). A Review of Video Object Detection: Datasets, Metrics and Methods. *Applied Sciences*, 10(21):7834.

Zou, Z., Shi, Z., Guo, Y., and Ye, J. (2019). Object Detection in 20 Years: A Survey. *arXiv preprint arXiv:1905.05055*.