UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Civile, Edile e Ambientale - ICEA

Corso di Laurea Magistrale in Mathematical Engineering for Engineering and Science

Negative Ion Beam profile estimation on STRIKE calorimeter by means of Machine Learning method

Supervisor
Dr. Nicolo' Marconato
UNIVERSITÀ DEGLI STUDI DI PADOVA

Co-supervisor
Dr. Rita Sabrina Delogu
CONSORZIO RFX

Master Candidate
Kenji Urazaki Junior

Student ID
2013056

Academic Year
2022-2023

To my grandparents, whose memories remain with me always.

Abstract

This thesis addresses the challenge of estimating the parameters of beamlets from thermal images in the STRIKE diagnostic calorimeter, a crucial component for the neutral beam source SPIDER. The latter is a clone of the main part of the MITICA Neutral Beam Injector, which is the full-scale prototype of the ITER tokamak additional injection power system. SPIDER produces up to 1280 beamlets of H-/D-, which are collected by STRIKE during 10-second shots. STRIKE is observed via thermal cameras, and the thermal pattern given by every single beamlet has been experimentally proven to be approximated as a 2D Gaussian curve. Traditional methods for fitting beamlets are insufficient due to their time-consuming nature, and previously developed rapid methods are not feasible for the new operation conditions of SPIDER.

This study tests two machine learning techniques, applying both unsupervised and supervised learning for fast and efficient beamlet parameter estimation. An unsupervised Gaussian Mixture Model (GMM) and a supervised deep learning model, YOLO (You Only Look Once), were trained on synthetic images to detect and localize Gaussian approximations of the beamlets. The YOLO model, in particular, demonstrated superior performance, accurately identifying beamlets with tight bounding boxes even in cases of significant overlap. Refinement techniques for YOLO as PX modifier and Ensemble were explored but didn't yield better results.

Challenges remain in correctly estimating the amplitude of overlapping Gaussians. Therefore, the thesis emphasizes the need for future work in disentangling overlapped Gaussians and extending model training to experimental STRIKE data. Besides, the methods developed in this thesis offer a promising approach for characterizing SPIDER beam profiles: YOLO for detecting and characterizing the beamlets with fast predictions of less than one second usually, and GMM as a support method to label future experimental data.

Contents

Listing of figures

Listing of tables

Listing of acronyms

CNN . . . . . Convolutional Neural Network

DS . . . . . Dataset

FFT . . . . . Fast Fourier transform

GMM . . . . . Gaussian mixture model

HWHM . . . . . Half width at half maximum

IR . . . . . Infrared

MLP . . . . . Multilayer perceptron, same as NN

NBTF . . . . . Neutral Beam Test Facility

NN . . . . . Neural Network

PCA . . . . . Principal Component Analysis

SPIDER . . . . . Source for Production of Ion of Deuterium Extracted from RF plasma

STRIKE . . . . . Short-Time Retractable Instrumented Kalorimeter Experiment

YOLO . . . . . You Only Look Once model

1 Introduction

The ITER (International Thermonuclear Experimental Reactor) tokamak is an international project to prove the possibility of producing energy by nuclear fusion of deuterium and tritium nuclei on a reactor scale. Additional heating systems are required to heat up the plasma to the necessary temperature for fusion, like injecting neutral beams.

A diagnostic calorimeter, STRIKE (Short-Time Retractable Instrumented Kalorimeter Experiment), made of 1D-CFC tiles, has been developed to study the neutral particle beam generated on SPIDER (Source for Production of Ion of Deuterium Extracted from RF plasma). The beam impinges on the calorimeter, and thermal cameras provide images of the tile's temperature profile on the rear side.

This thesis proposes estimating the beamlets' parameters from the temperature profile via machine learning. The model's input is the image taken from a specific time of the STRIKE operation in which the identification of the individual beamlets as approximated 2D Gaussians is possible. The longer the time in the operation, the lower the local information about each beamlet hitting the tile as the heat transfers in the direction of length and width and the individual peaks of temperature start to merge.

Traditional methods to fit the beamlets are generally time-consuming, exceeding the required response time for the operation of SPIDER. Furthermore, the beamlets are expected to present divergence and deflection, increasing the difficulty of successfully identifying each one. The first campaign of SPIDER had five beamlets impingement per tile of the STRIKE diagnostics. After a retrofit of the equipment, the expectation is the total operation with eighty beamlets per

tile for the end of 2023, so a new method of fitting the 2D Gaussians is needed, as the previously developed ones have limitations regarding the high number of entities in the frames.

All images used in training and testing the algorithms in this thesis were artificially generated. Two datasets were used, identified as DS-I and DS-II. The first one, with higher variance in the beamlets' parameters, aimed to stress-test the detection methods; the second was as similar as possible to the future temperature profiles in STRIKE. Both unsupervised and supervised machine learning techniques were used to estimate the Gaussian beamlet parameters from the thermal images.

The unsupervised Gaussian Mixture Model (GMM) was tested to cluster the data points and fit Gaussians representing each beamlet. Initial Gaussian center estimates from peak detection facilitate the iterative fitting. The choice of the GMM was direct, as the beamlets are approximated as Gaussians, so the final thermal images would be a sum of the distributions.

The supervised deep learning model, YOLO (You Only Look Once), was also trained on labeled synthetic images to detect and localize each Gaussian beamlet. This is the first time YOLO has been applied to characterizing the SPIDER beamlets. YOLO was selected due to its state-of-the-art object detection capabilities, ability to handle overlapping objects, and fast predictions. After training, YOLO rapidly analyzes new images, predicting bounding boxes that identify the Gaussian beamlets. The box locations and dimensions provide the necessary parameters to characterize each beamlet.

Both unsupervised GMM and supervised YOLO strategies were evaluated on the synthetic test data. Their estimation performance was compared through error metrics such as beamlet center positions, standard deviations, and amplitudes.

The YOLO model demonstrated the most robust performance in detecting and localizing the individual Gaussians on the synthetic test images. After training on labeled data, YOLO accurately identified the beamlets and bounded them with tight bounding boxes even when significant overlap was present. Its error rates for predicting Gaussian centers and standard deviations were substantially lower than the unsupervised GMM approach.

Additionally, two refinement techniques, PX modifier and PX modifier 2, were introduced to improve YOLO's localization accuracy further. These methods analyzed the predicted Gaussian centers and employed the probability density function to update the estimated standard deviations. However, they were found to perform inconsistently due to sensitivity to overlapping Gaussians.

Furthermore, all methods struggled to correctly estimate Gaussians' amplitude or intensity value when heavy overlap causes them to merge. The overlapping causes the peak brightness to

exceed the individual components' true amplitudes. Another critical limitation in the models was the synthetic data, which may not fully capture the complexity of experimental STRIKE images.

To bridge these gaps, future work should focus on disentangling overlapped Gaussians to improve amplitude characterization. Training and evaluating models should also be extended to use future STRIKE data. Finally, the machine learning techniques in this work could have broader applications in nuclear fusion and other physics experiments needing rapid image data analysis.

In conclusion, the supervised YOLO model provides a promising solution for characterizing the 80 beamlets expected in the upgraded STRIKE diagnostic. Although not reaching astonishing performance, as the GMM is an unsupervised learning model, it may be used in support of labeling of new images. YOLO can become a valuable tool for rapidly analyzing beam profiles during SPIDER operation with further refinement to address amplitude estimation. This will facilitate accelerated experiment optimization to meet the heating and current drive requirements for ITER.

The thesis is organized into four chapters. The chapter on Literature overview introduces the problem, provides background context, and presents relevant literature on beamlet characterization, machine learning techniques, and prior work applying neural networks to this problem. The chapter on Methods presents the methodology, including the synthetic data generation, application of the machine learning models, training procedures, and performance evaluation metrics. The chapter on Results analyzes the results, evaluates model performance on the test data, and compares the strengths and limitations of the approaches. Finally, the conclusion chapter summarizes the essential findings.

2

Literature overview

This literature review summarizes past research on rapidly characterizing particle beamlets using thermal imaging data from the SPIDER experiment. It also gives a general context of the subjects related to this thesis.

The goal is to provide context on the challenges and gaps this thesis aims to address. First, it outlines the general context of fusion and related topics. Finally, review previous methods proposed to estimate beamlet parameters from IR (infrared) camera images, including transfer function models, neural networks, and convolutional neural networks.

It highlights critical identified difficulties, such as scaling existing approaches to accommodate more beamlets. It previews this thesis's objectives and proposed techniques to overcome these limitations through innovative deep-learning solutions. Specifically, this review discusses the need for reliable, efficient estimation of parameters for Gaussian fits to approximately 80 beamlet profiles per tile.

Finally, a brief explanation of the methods used is exposed by describing crucial concepts of Machine Learning, the Gaussian mixture model (GMM), Neural Networks (NNs), Convolutional Neural Networks (CNNs), and the YOLO architecture.

Therefore, it provides an orientation on how this thesis builds upon and extends past literature to advance beamlet characterization capabilities for diagnostics of the future operation of SPIDER.

2.1 Fusion

The production of energy is a big challenge as it needs to fulfill the increasing energy demand while managing the sustainability and cost-effectiveness of the process. One of the promising pathways is energy production using nuclear fusion. Compared to the nuclear fission used today, it does not present the same magnitude of problems with the long lifetime of radioactive waste.

Fusion combines light atomic nuclei to form a heavier nucleus, releasing much energy. The most efficient fusion reaction in the laboratory setting is between two hydrogen isotopes, deuterium (D) and tritium (T), to form heavier helium atoms with the release of energy. The DT fusion produces the highest energy gain at the "lowest" temperature.

The fulfillment of three conditions are needed to reach nuclear fusion: very high temperature (around 150 million degrees Celsius); sufficient plasma particle density, which increases the likelihood of collisions between particles; and sufficient confinement time, as the plasma tends to expand and it is needed to restrict it in a defined volume [1]. The deuterium isotopes can be extracted from seawater, guaranteeing its supply for thousands of years, and the tritium isotope production is feasible from lithium [2].

The electrons are separated from the nuclei in a gas at extreme temperatures. Hence, the gas becomes a plasma and can be confined and controlled using powerful magnetic fields in different designs. One of the designed reactors is the Tokamak reactor [1].

2.1.1 Tokamak

The Tokamak is an experimental nuclear fusion reactor designed to produce energy from fusion reactions [3]. Its key objective is to confine plasma at extremely high temperatures to enable fusion between hydrogen isotopes [3], [4].

The Tokamak was first developed in the late 1950s by Soviet research programs, and its name comes from the Russian acronym for "toroidal chamber with magnetic coils." It has since been adopted worldwide as the most promising magnetic confinement configuration [3].

The Tokamak confines the plasma using powerful magnetic fields generated by coils surrounding a torus-shaped vacuum chamber. The charged plasma particles are controlled and shaped by the magnetic fields enclosing the hot plasma away from the reactor walls [3].

The Tokamak chamber is first evacuated and then filled with hydrogen gas during operation. Through electrical currents in the vessel, the gas ionizes into plasma, and as the particles become

energized and collide, the plasma heats up. Additional heating methods raise the temperature to 150-300 million °C to initiate fusion reactions [4].

The energy released heats the reactor walls and might be used to produce steam to drive turbines to generate electricity, similar to conventional thermal power plants [3].

2.1.2 ITER

ITER (International Thermonuclear Experimental Reactor) is an international scientific collaboration to demonstrate the scientific and technological feasibility of fusion energy. Thirty-five nations are collaborating to construct ITER in southern France, and the world's largest tokamak assembly is underway. ITER will have the largest Tokamak, with a plasma chamber volume ten times greater than current devices [1], [3].

The key objective of ITER is to achieve a self-sustaining burning plasma that produces net energy gain from the fusion process, 500 MW of fusion power from 50 MW of input heating power. To accomplish this, ITER will heat hydrogen isotopes, deuterium-tritium, to temperatures over 150 million °C and confine the plasma within a Tokamak reactor. It will test integrated systems and components essential for future fusion power plants, including tritium breeding module, remote maintenance, diagnostics, and safety [1].

The additional heating systems for ITER are two neutral beam injectors, with a third one possible if an upgrade is necessary during operation [5]. ITER requires a significantly high injected particle energy of 1 MeV for effective plasma heating. This ion source must deliver a total extracted ion current of 40 A and provide an injected heating power of 16.5 MW to the ITER plasma [6].

2.1.3 Neutral beams

Injecting a particle beam into the reactor provides the plasma with additional heating. The injected fast neutral hydrogen atoms become ionized through collisions with ions and electrons in the plasma. The magnetic field then confines these newly ionized particles. The injected ions gradually transfer their kinetic energy to the electrons and ions constituting the plasma via successive collisions, heating it [7]. Acceleration of the injected particles and the formation of the beams occurs before entering the plasma, i.e., outside of the reactor. They must penetrate the plasma enveloped by its strong magnetic fields [8].

The needed beam is composed of high-energy neutral particles as the inexistence of charge permits the particles to freely enter the strong magnetic fields around the plasma. This beam

production and acceleration process is the focus of the NBTF Project in Italy, Padua [9].

2.2 NBTF Project - Neutral Beam Test Facility

The key objective of the NBTF Project is to demonstrate a neutral beam injector capable of delivering 16.5 MW of 1 MeV power for 1-hour pulses, as ITER needs. To accomplish this, the NBTF Project is developing two prototypes at Padua - SPIDER, which tests the negative ion source, and MITICA, which tests the full-scale 1 MV neutral beam injector [9].

SPIDER and MITICA allow experimental optimization of the physics and technology for negative ion production, extraction, acceleration, and neutralization at the required power levels. The knowledge gained from testing these prototypes enables the NBTF Project to finalize the neutral beam injector design for ITER [9].

2.2.1 RFX

RFX Group is a research consortium established in 1996 consisting of five Italian partners – CNR (National Research Council), ENEA (Italian National Agency for New Technologies, Energy and Sustainable Economic Development), INFN (National Institute for Nuclear Physics, added to the consortium in 2005), University of Padua, and Acciaierie Venete S.p.A (steel producer) [10]. The RFX Group aims to expand understanding of fusion-relevant plasma physics and push technological boundaries in research facilities [11].

The RFX Group pursues this goal by operating experiments like RFX-mod2 and participating in major fusion projects like ITER. RFX houses test facilities like SPIDER and MITICA prototypes in the NBTF experiment to support ITER-neutral beam development. With multidisciplinary expertise and international collaboration, the RFX Group provides research, training, and innovation to progress fusion [11].

2.2.2 MITICA

The key objective of MITICA (Megavolt ITER Injector & Concept Advancement) is to demonstrate the operation of a 1 MV neutral beam injector capable of meeting ITER's injected heating power requirements. MITICA aims to produce a neutral particle beam with an energy of 1 MeV at a power of 16.5 MW sustained for 1-hour pulse lengths [8].

To achieve this, MITICA utilizes a negative ion beam source and a 5-stage accelerator to generate and accelerate negative ions. The neutralization of the ions occurs before injection,

and in the 1:1 scale prototype, they will hit the target calorimeter. The results from MITICA will validate the injector design and performance for integration on ITER [8].

2.2.3 SPIDER

The SPIDER (Source for Production of Ion of Deuterium Extracted from RF plasma) is a prototype negative ion source designed to test the feasibility of meeting the neutral particle beam injection (NBI) requirements that will heat the ITER tokamak. SPIDER is the world's most powerful negative ion source and has operated since 2018 [12]. Optimization of SPIDER operational variables allows testing of different operating regimes to characterize the beam optics as uniformity, divergence, and current [12].

SPIDER utilizes RF power to produce a dense plasma in the source from which negative ions are extracted. An extraction and acceleration system follows the ion source, consisting of three electrode grids with 1280 holes each. The plasma is extracted and accelerated through these grids using a potential difference [12]. The source and diagnostics are enveloped by a vacuum chamber formed by two cylindrical sections 4 meters in diameter and 6 meters in length [13].

Key design parameters for SPIDER are [12]:

- Beam energy of 100 keV
- Production of negative ions of Deuterium and Hydrogen
- RF system power of 800 kW
- Pulse duration of 1 hour
- Internal source pressure of 0.3 P

SPIDER provides critical data to finalize the design of the ITER ion source to meet its heating and current drive requirements. Different systems can measure the accelerated ion beam; one is the STRIKE diagnostic calorimeter, and the other is the Beam Dump calorimeter [13]. The first one is the main diagnostics for the SPIDER experiment; the latter is a water-cooled calorimeter that can absorb all the power of the beams in a steady state and has calorimetric measurements performed. The advantage of the Beam Dump is that it can operate for long pulses at full power, but as a disadvantage, it does not have a high resolution as STRIKE as it is limited to a few centimeters [13].

The SPIDER with the beamlet impingement structure and the positioning of the IR cameras used to capture images from STRIKE is presented in figure 2.1.

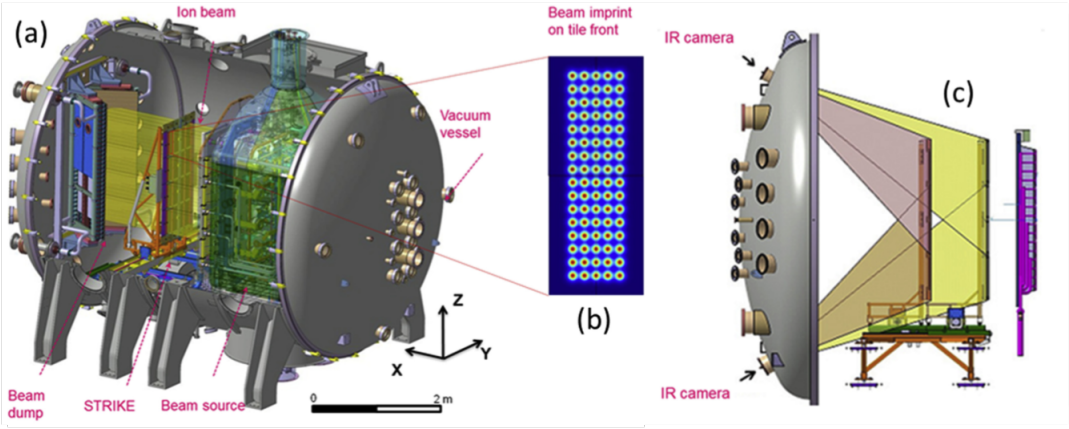


Figure 2.1: (a) SPIDER, (b) Tile impingement - 80 beamlets, (c) Positioning of IR cameras. Extracted and adapted from Delogu et al. (2019) [14].

STRIKE

STRIKE (Short-Time Retractable Instrumented Kalorimeter Experiment) is a high-resolution calorimeter of 16 carbon fiber composite (CFC) tiles mounted on a metal structure. The organization of the tiles follows a 4 x 4 structure. The calorimeter allows measurements with a precision of about 3 mm. Still, as it is not cooled, it only characterizes the impulses in a very short window of a few seconds [13]. The key objectives of STRIKE are to measure the uniformity, current, deflection, and divergence of the SPIDER beams, and it is considered the central diagnostic system for the SPIDER experiment [13].

The unidirectional carbon fibers are oriented in the same direction as the beam, so the heat conduction in the depth direction is significantly higher than in the height and width directions; the ratios of the tiles used were 20 in this aspect [14]. The main measurements of the calorimeter are high-precision 2D temperature maps formed by the beams hitting its surface and collected by two IR (infrared) cameras.

The anisotropic conductivity property of the tile is necessary since the IR cameras used for capturing the data for the calorimeter are positioned at the back of the surface in which the beams hit the tiles. Hence, transferring the thermal pattern to the back must occur with distortion as small as possible [15] , [14].

The front-facing optical measurement approach faces challenges from two sources. First, the energized gas layer between the beam and the calorimeter emits light, interfering with observation. Second, the beam's heat causes the material to transition from solid to gas on the calorimeter's surface, releasing obscuring particles [14].

The IR camera possesses a 640 × 480-pixel resolution and can identify temperature variances as low as <30 mK, and it transmits full-frame 16-bit data at up to 50 Hz [16]. The axial positioning accuracy is +/- 5 mm, and the final error in estimating the beamlet pattern is lower or equal to +/- 5% [2].

Besides the two IR cameras to measure the temperature changes, STRIKE has two other sets of direct diagnostics, which are the thermocouples to measure the local temperature to calibrate the signal from the IR cameras and current sensors that measure the current of the beams [2].

In the full operation of SPIDER, 80 beamlets organized in a 16 x 5 format should hit each STRIKE tile, as the calorimeter is composed of 16 tiles, and the equipment will operate with 1280 beamlets [14]. The analysis to determine the parameters of the energy beam and energy flux profile entails solving a complex inverse non-linear problem, which is mathematically challenging due to its ill-posed nature. Essential data for this analysis includes the non-linear attributes of the tiles and the two-dimensional temperature distribution captured on the calorimeter's surface [16].

Figure 2.2 presents the STRIKE calorimeter, a close-up of the CFC tiles, and the IR camera capture from the first campaign of SPIDER.

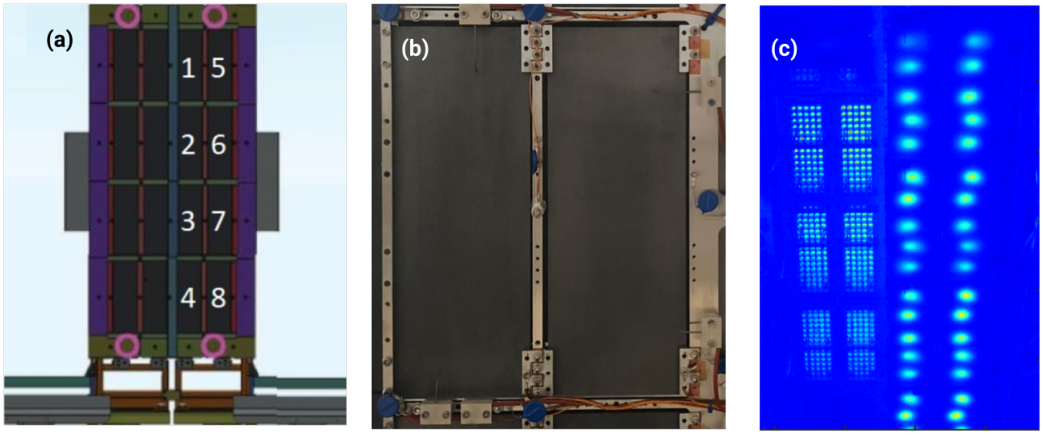


Figure 2.2: (a) STRIKE, (b) Two CFC tiles, (c) Image capture by IR camera on the first campaign - The right side is the rear side of the tiles. Extracted and adapted from Canocchi (2019) [2].

2.3 Beams characterization

2D Gaussians approximate each beamlet in the temperature profile representation on STRIKE. They have their positions, standard deviation, and amplitude influenced by SPIDER's operating and construction conditions. The equation 2.1 presents a Gaussian distribution probability density generally used in the fittings considering zero angulation and normalized amplitude.

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{\left(-\frac{1}{2}\left[\frac{(x-\mu_x)^2}{\sigma_x^2}+\frac{(y-\mu_y)^2}{\sigma_y^2}\right]\right)} \tag{2.1}$$

where σ_i represents the standard deviation of the Gaussian in the i-th direction and μ_i the center coordinate in i-th direction.

The evaluation of the beamlet formation involves three essential characteristics, which are discussed in the work of Canocchi (2019) [2]. The beamlet divergence, deflection, and current can be estimated using the parameters of the approximated Gaussians.

Beamlet divergence measures how distributed the beam particles' velocities in the plane transverse to the direction of the beam propagation. Its value relates to the standard deviation of the approximated Gaussian of the beamlet. Deflection is the difference in the position of the center of the beamlet from the expected one divided by the distance of STRIKE to the last grid on the ion source. It is possible to estimate current by computing the total energy deposited on each tile by knowing the temperature increase and the specific heat, density, and volume of the material [2].

The operation of the SPIDER in the first campaign involved a mask of five holes per tile in front of the STRIKE calorimeter due to pressure issues [16]. A remarkable characteristic of the beamlet shape in the first campaign, which only had five beamlets hitting each STRIKE's tile, was that it had an ellipse form, flattened in the x-direction [2].

Grids extraction and acceleration tensions, current in the filter, and the pressure in the plasma are the operating conditions of SPIDER that can change the beamlet characteristics as discussed in the work of Canocchi (2019) [2]. The artificially generated images used in this current thesis accounted for these variations in beamlet characteristics.

The final expected values of the parameters used in the synthesis were based on the simulations of Pimazzoni (2018) [17] as there is no experimental data of the full operating conditions of 80 beamlets per tile on STRIKE. The database was generated and provided by Dr. Delogu and Dr. Pimazzoni to the author and reflects the expectation of the results of future experimental images.

2.3.1 Previous works in fast characterizing STRIKE

The STRIKE measurements performed by the IR camera must be analyzed to understand a series of beamlet characteristics. The work of the identification of the profile of the beams needs to be performed within a few minutes as it is a requirement to understand the beamlets' conditions in operating time [16], [14].

Various methods mentioned in the literature can estimate the impinging flux with minimal error. However, they share a common disadvantage of substantial computational time required to achieve convergence for each 2D thermal pattern examined [16].

The RFX team studied different approaches to reduce this time of estimation during operation, each tackling a specific problem of characterizing the IR frames. The common challenge is identifying the beamlets in the tile. However, all dealt with fewer beamlets hitting each tile, generally only five of them, a number lower than the 80 beams in the next full campaign of SPIDER.

In the works developed, there is an assumption that the input-output link of the temperature-heat profile is invertible [16]. This section briefly describes some of the studies with the leading results.

In 2015, Delogu et al. [18] presented a transfer function method to solve the problem of faster analysis of the STRIKE profile utilizing the fast Fourier transform algorithm. The previous methods used a multiple peak 2D fit of the beamlet thermal pattern, which had a high convergent time. The study performed tests in a small-scale version of the calorimeter (mini-STRIKE), and the tile was considered a system on the control-engineering view and so could be modeled using a transfer function. The energy flux of the beamlets profile was then estimated using the temperature profile via the inverse of this function. The energy fluxes profile approximation was a 2D Gaussian distribution constant in time. The Hubbert function approximated the temperature profiles, which are always larger and smoother than the energy profiles. The estimation results were good, but the number and locations of the beamlets in the study restricted the use to the future 80 beamlets per tile operation of STRIKE.

Delogu et al. in 2019 [14] presented an alternative approach. The new method used Neural Networks to estimate the heat flux profile by pre-processing the image using FFT (Fast Fourier Transform) and PCA (Principal Component Analysis) for dimensionality reduction. Two general models were proposed for the stationary and non-stationary IR data. One model directly predicted the heat flux from the 2D temperature profile from the calorimeter IR cameras as input. The other model predicted the 2D temperature profile from the heat flux as input, so

the authors expected to solve the inverse problem inverting this later model. The direct prediction model proved more feasible for the stationary condition and was considered reliable for performing the task.

However, despite the excellent performance in the synthetic data, the model built by Delogu et al. (2019)[14] failed in the experimental one due to the high noise in the frames [16]. In a later development, it was concluded that using the parameters of the approximated Gaussians for the temperature and heat fluxes profiles resulted in better performance for the experimental data [16].

As the model from Delogu et al. (2019)[14] proved not to be suitable for the experimental data, in the study of Delogu et al. (2022) [16] a new set of NNs was proposed and tested. The model's inputs were the seven characterizing parameters from the fitted Gaussians in the 2D temperature profile (x and y center, x and y standard deviation, amplitude, and angulation). The outputs were the five parameters for the fitted Gaussians for the heat flux (excluding angulation and differentiation in x and y standard deviation).

Delogu et al. (2022) [16] tested different architectures of the NN (Neural Network), varying the number of outputs. The best model achieved was to predict each output parameter of the heat flux profile in different NNs, resulting in 25 NNs trained with just a single output. The final reconstructed heat flux using the best model type reached high performance with low errors in the standard deviation and amplitude estimations.

It is essential to highlight that the model from Delogu et al. (2022)[16] needed the parameters of the approximated Gaussians of the temperature as input. The work tackled fitting the profiles for five beamlets per tile. However, increasing to 80 beamlets per tile demands new tools to perform the estimations, which was the core objective of this thesis, as the results can later be used as input to the model from Delogu et al. (2022)[16] to estimate the heat flux profiles.

Steffinlongo (2019) [19] developed two NNs to characterize the heat flux from the temperature profile on STRIKE, operating with five beamlets per tile impingement. The difference between the two NNs was the output, with one predicting the heat profile directly and the other the characteristic parameters of the fitted Gaussians of the heat profile. The same preprocessing used in the work of Delogu et al. (2019) [14], a combination of FFT and PCA, was applied to reduce the input space while maintaining the main features.

In the study, Steffinlongo (2019) [19] concluded that the direct prediction of the heat profile from the temperature profile didn't reach good results as the estimated standard deviation and amplitudes presented high errors. The second method analyzed, the prediction of the parame-

ters of the approximated Gaussians of the heat flux, reached low errors in the reconstruction of the profile for synthetic data. However, it could not estimate the HWHM (Half-width at half maximum) of the Gaussians for the experimental data, as it did not have the parameter variance between beamlets in the same image on the training data. The synthetic data used in the current thesis incorporated this variation in the HWHM on the same image on the database, as recommended by Steffinlongo (2019)[19].

As discussed by Steffinlongo (2019)[19] and Delogu et al. (2019)[14], the pre-processing using FFT and PCA of the thermal image involved a choice in the number of principal components and frequencies to reduce the number of inputs while maintaining the main features of the signal. The manual choice of compression level is unnecessary when the image transformation is performed using CNN (Convolutional Neural Network) layers, which gives the same effect of dimensionality reduction with the extraction of main features. The final compression and feature extraction in the CNN layers results from the architecture of the NN chosen.

In the study of Lonigro (2019)[20], a CNN was trained to predict the parameters of a Gaussian given the input image. The synthetic Gaussian images used in the study presented variations in center position, standard deviation in the x and y direction, amplitude, and rotation, totalizing seven parameters. Each image showed a random number of randomly generated Gaussians in a 200 by 300 pixels black image, and the model output was the seven parameters characterizing each Gaussian present.

The training of an initial CNN to determine the centers yielded promising results for just one Gaussian in the image. At the same time, the second tested CNN to predict the centers and the standard deviations together did not reach the same performance. It even increased the error in the center estimation. The final model architecture that presented the best results for the prediction of one Gaussian was the one composed of three CNNs: one to predict the centers, the other one for the standard deviation, and the last one for the angulation. Training different CNN models with two Gaussians in the image did not yield good results [20].

The CNN models looking at the entire image from Lonigro (2019)[20] could not successfully identify more than one entity in the image, as the complexity of learning the image becomes increasingly higher as the combination of objects' location and shapes are exponential with its number. The author at first circumvented the problem using a close-up strategy, which required the estimation of the center of the Gaussian and a crop-size parameter to extract a subimage in which the trained CNN was used to estimate all parameters of the extracted entity. This method also presents limitations, as the crop size needs to be adequately chosen for each dataset and is not learned directly during the training step.

As the two Gaussians estimations already presented challenges, using the close-up strategy for a higher number of Gaussians was not feasible, so Lonigro (2019)[20] dealt with this problem of identifying many entities moving a sliding window in the image and applying the same best CNN from the estimation of up to two Gaussians. This last attempt resulted in good results but with misfires of some estimates and only considered ten Gaussians in the image randomly organized in the tests.

The method derived in the current thesis to localize and detect multiple objects in the image differs in approach. It does not utilize the sliding window technique but the YOLO – You Only Look Once – architecture. The YOLO architecture permits a more flexible range of object locations and shapes in the detection than a sliding window strategy, as it processes the whole image simultaneously [21].

Finally, Cannochi (2019)[2] performed different pre-process in the images of the STRIKE, such as background subtractions, image clipping, and perspective correction, as well as post-process, such as computation of the energy deposited on the tiles, calorimetric current, deflection, and divergence of the beams. Knowing the beamlet profiles produced in STRIKE allows many analyses of the operating state of SPIDER, which the studies of Cannochi (2019)[2] extensively discussed.

Fastly estimating the beamlets' characteristics may lead to a quicker experiment adjustment. Hence, this thesis's main objective is a reliable and efficient estimation of the parameters of the beamlets' approximated Gaussians considering the complete operation condition of 80 beamlets per tile in the STRIKE tiles. The post-process and pre-process algorithms and methods are out of the scope as it is already well developed in previous works and can be extended to the analysis of future conditions.

2.4 Machine Learning

Machine learning is a branch of artificial intelligence that aims to learn the representation of a problem through data. This section's formal description of this learning process was based on the book of Shalev-Shwartz, S. and Ben-David, S. (2014) [22] and considered a supervised learning framework.

It is important to note that this thesis applies both unsupervised learning, the Gaussian Mixture Model, and supervised learning, YOLO, to solve the problem of identifying the beamlets' approximated Gaussians.

2.4.1 Supervised Learning

Supervised learning is where the model is provided with labeled training data, aiming to learn a mapping from inputs to outputs. The learner in the basic supervised statistical learning setting has access to the following data:

- Domain set ($\mathcal{X}$): an arbitrary set of objects that need to be labeled;
- Label set ($\mathcal{Y}$): the possible labels of the objects, for example, 0 or 1;
- Training data ($\mathcal{S}$): The sequence of pairs $\mathcal{X} \times \mathcal{Y}$ the learner can access. Each input ($\mathcal{X}$) has a known label ($\mathcal{Y}$), often called a target, and it serves as the training set of the problem;
- Learner's output: h, a predictor.

The learner must use the training data to output a prediction rule or hypothesis, $h : \mathcal{X} \to \mathcal{Y}$. This hypothesis can predict the label of new objects in the domain set outside of the training data.

The learning algorithm learns through the minimization of the training error that it has access to. This error is called empirical error or empirical risk, so the whole process of outputting a prediction rule h that minimizes the error is called Empirical Risk Minimization (ERM) [22]. The equation for the computed empirical risk function for a particular prediction rule h in a training set of m samples is presented in equation 2.2. The equation 2.3 presents the best output of a learner given a set of hypotheses $\mathcal{H}$; for example, $\mathcal{H}$ can be the set of linear functions that relates x and y.

$$L_s(h) := \frac{|i \, \varepsilon \, [m] : h(x_i) \neq y_i|}{m} \tag{2.2}$$

$$\hat{h} := \underset{h \,\varepsilon\, \mathcal{H}}{argmin}\; L_S(h) \tag{2.3}$$

There are many different learning tasks that a machine learning model can tackle. Here, two are described with their suitable loss function to evaluate the quality of the predictor. The way to evaluate a predictor will change according to the learning objective, and the format of the loss function used is as important as the definition of the hypothesis class.

In Multiclass Classification, for example, the objective is to classify objects based on the training set of correctly classified ones. A predictor that identifies the color (label) (e.g., red, blue, yellow, green) is a Multiclass Classification problem, and the evaluation of this predictor could be based on the probability of getting a wrong classification result.

In Regression, the objective is to find a pattern between $\mathcal{X}$ and $\mathcal{Y}$ values, i.e., a function to predict the target $\mathcal{Y}$ using the domain set $\mathcal{X}$ as input. For example, a predictor that identifies the maximum stress ($\mathbb{R}$) of a structure based on the displacement in a point ($\mathbb{R}^2$) is a regression problem, and the sum of the squared differences between the true target and the predicted values could evaluate this predictor.

The loss functions (l) are defined as any function such that, given a set of hypothesis $\mathcal{H}$ and some domain $\mathcal{Z}$, it maps $\mathcal{H} \times \mathcal{Z}$ to nonnegative real numbers, $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_b$. The domain $\mathcal{Z}$ in the prediction problem is the product of the domain set and the labels, i.e., an element in the domain $\mathcal{Z}$ is the tuple z = (x, y) where $x \,\varepsilon \mathcal{X}$ and $y \,\varepsilon\, \mathcal{Y}$.

The empirical risk is the expected loss of a sample S = $(z_1, \ldots, z_m) \,\varepsilon\, \mathcal{Z}^m$, and its equation is presented in 2.4.

$$L_S(h) := \frac{1}{m} \sum_{i=1}^{m} l(h, z_i) \tag{2.4}$$

The typical loss functions used for classification and regression tasks are:

- 0-1 loss: normally used for classification problems:

$$l_{0-1}(h, (x, y)) := \begin{cases} 0 \text{ if } h(x) = y \\ 1 \text{ if } h(x) \neq y \end{cases} \tag{2.5}$$

- Square loss: normally used for regression problems:

$$l_{sq}(h, (x, y)) := (h(x) - y)^2 \tag{2.6}$$

The empirical risk is used as an estimate of the minimization of the true risk represented by function 2.7. So, minimizing the empirical risk might not guarantee the minimization of the true risk, a problem related to the generalization of the models.

$$L_{\mathcal{D}}(\mathbf{w}) = \mathop{\mathbb{E}}_{z \sim \mathcal{D}}[l(\mathbf{w}, z)] \tag{2.7}$$

where l is the loss function, which computes the error between the predictions and the true labels.

2.4.2 Unsupervised Learning

Unsupervised learning deals with unlabeled data, i.e., only the domain set ($\mathcal{X}$) is known. The algorithm explores the structure and patterns inherent in the data without explicitly being told the "right answer." Common unsupervised learning methods include:

- **Clustering**: The objective is to group similar data points based on specific criteria.
- **Dimensionality Reduction**: Aims to reduce the number of variables in consideration, making the data more accessible to visualize or compute.

The principle of minimizing a loss function applies to the unsupervised learning framework; for example, in the k-means clustering algorithm, the loss function minimized is the total distance of the data points to the centers of the k clusters.

2.4.3 Gaussian mixture model

A Gaussian mixture model (GMM) supposes that data points come from a combination of Gaussian distributions whose parameters are unknown. It is like an extension for the k-means clustering by modeling the covariances of the data along with the centers of the latent Gaussian components [23]. In the GMM model, the true assignment of each data point to a Gaussian is unknown, so this is an unsupervised learning problem.

As described in section 2.3, 2D Gaussians can approximate the beamlet temperature profile in the STRIKE calorimeter. Therefore, the final resulted profile in the tile can be represented by the sum of many Gaussian, and the problem is described as a Gaussian mixture model.

Given many beamlets, i.e., Gaussians, the problem is estimating each entity's parameters. This can be framed as an optimization problem - finding the parameters that maximize the likelihood given the data distribution.

The Gaussians in the mixture can have different degrees of freedom in the fitting regarding the covariance matrices. The sklearn.mixture module allows configuring different covariance settings for the Gaussian components. This controls whether the Gaussians have correlated or independent variances [23].

There are four covariance types [23]:

- Full: Each Gaussian has its general covariance matrix between the x and y dimensions, and it allows correlations between x and y variances;
- Tied: Same covariance structure as "Full," but all Gaussians share the same general covariance matrix;
- Diagonal: Each Gaussian has its diagonal covariance matrix, meaning no x and y variances correlation, i.e., the x and y variances are independent, and the Gaussian format has no angulation;
- Spherical: Each Gaussian has a single variance value, meaning circularly symmetric.

The 'full' setting is the most flexible, allowing each Gaussian to have differently shaped ellipses based on the covariance matrix. The 'spherical' setting forces circular Gaussians and is the strictest. Therefore, the appropriate setting depends on assumptions about the x and y variances correlations and constraints and possible simplifications for the problem.

The fitting of the parameters in GMM occurs using the expectation-maximization (EM) algorithm, which provides an iterative approach. First, initial Gaussian parameters are randomly set (e.g., centered on data points, from k-means algorithm, or user-defined). Then, for each

data point, the probability that it was generated from each Gaussian is calculated. The parameters are then updated to maximize the likelihood of the data under these Gaussian assignments. Repeating this process guarantees convergence to a local optimum [23].

Formally, the log-likelihood of a data point x^i given the latent variables, i.e., hidden variables, is expressed in equation 2.8. The objective of the EM algorithm is to maximize this likelihood w.r.t. the parameters θ (means, covariances) of the Gaussians [24].

$$l(\theta) = \sum_{i=1}^{n} \log \sum_{z^{(i)}} p\left(x^{(i)}; z^{(i)}; \theta\right) \tag{2.8}$$

where n is the number of independent samples, and z is the latent variable.

2.5 Neural Networks

Neural networks (NNs) are powerful modeling techniques for different learning tasks. It's vastly used in problems in that no explicit rules can be programmed, for example, image recognition. This section aims to formally define an NN and present the general toolset used to train this model. The author followed the descriptions given in the book of Shalev-Shwartz, S. and Ben-David, S. (2014) [22].

The NN has a series of interconnections (links) between neurons, generally present in large numbers. These connections are typically represented as lines and the neurons as graph nodes. Neurons have a primary function in transforming the input data into an output via a transfer function. Before information enters the next neuron, it passes through an interconnection with a weighted factor applied. Therefore, the input data of the neurons is the sum of information arriving through the edges. [22].

In the context of learning, the hypotheses class of NN are the predictors that share the same underlying graph structure but have different weights at the edges. The training process of an NN involves minimizing a loss function within the hypotheses class, which consists of parameters such as weights and bias. [22].

A feedforward NN is presented in figure 2.3. The model is generally structured in input, hidden, and output layers. The artificial neurons are fully connected, and information travels from the input to the output layer.

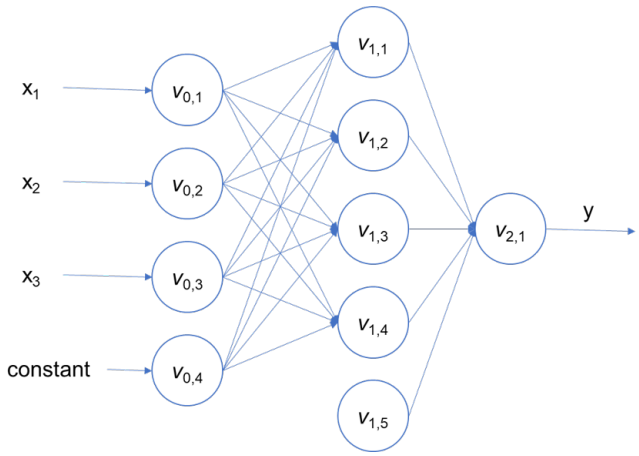


Figure 2.3: Feedforward Neural Network.

Because of its complicated structure and typically high amount of data necessary, the learning process of the hypothesis class of NN predictors $\mathcal{H}_{V,E,\sigma}$ is computationally demanding. The

Stochastic Gradient Descendent and Backpropagation algorithms are used in this process [22].

2.5.1 Feedforward Neural Network

A directed acyclic graph represents the feedforward NN. Nodes in the graph represent neurons, and edges represent interconnections between them. The nomenclatures and descriptions are based on the book of Shalev-Shwartz, S. and Ben-David, S. (2014) [22].

Every neuron is a scalar function, transforming the input data into an output one, $\sigma : \mathbb{R} \rightarrow \mathbb{R}$. It generally acts as a threshold function or, as it is called, an activation function. There are many possible functions; the most used one is the sigmoid presented in equation 2.9. Another one that may be used is the ReLu function, shown in 2.10 [22].

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{2.9}$$

$$\sigma(a) = max(0, a) \tag{2.10}$$

The structure can be divided into layers (V) to understand an NN better. Given that T is the number of layers, a disjoint subset of neurons from the NN. The nodes in one layer are connected to the next one, so every layer connects a node in V_{t-1} to nodes in V_t, where t ε [T]. An edge E represents the connection, each with a weight function $w : E \rightarrow \mathbb{R}$.

The first layer, V_0, is the input layer, containing n+1 neurons. The n is the size of the input space, and the added neuron is a constant bias value that always has the value 1. The $v_{t,i}$ represents the i^{th} neuron in the t^{th} layer, and the output of this neuron for the input x is denoted by $o_{t,i}(x)$. The calculation proceeds layer by layer, i.e., the outputs of the layers t are the inputs for the layer $t + 1$.

A final expression of the input in the $t + 1$ layer can be constructed using the outputs of the t layer and the weights. Given a j neuron in the t+1 layer, $v_{t+1,j}$ ε V_{t+1}, the input $a_{t+1,j}(x)$ when the NN is fed with the input vector x is presented in equation 2.11. The output of the j neuron is shown in equation 2.12.

$$a_{t+1,j}(x) = \sum_{r:(v_{t,r},\ v_{t+1,j}\ \varepsilon\ E)} w((v_{t,r}, v_{t+1,j}))o_{t,r}(x) \tag{2.11}$$

$$o_{t+1,j}(x) = \sigma(a_{t+1,j}(x)) \tag{2.12}$$

The input to the neuron $v_{t+1,j}$ is the sum of outputs of the neurons in V_t connected to it. Besides the summation, the outputs of the neurons in V_t are weighted by the weight function w assigned over the connecting edge to the neuron $v_{t+1,j}$. The final output of this neuron is only the application of the activation function σ in its input.

The layers $V_1, \ldots, V_{T-1}$ are called hidden layers, and the final one, V_T, is the output layer. It can have a single or more neuron depending on the type of target that the model is predicting. The hidden layers $t+1$ can have neurons not connected to the previous layers, outputting a constant.

Some parameters to describe a feedforward NN are:

- The value T is generally called the depth of the network (excluding the input layer V_0).
- The size of the network, $|V|$, represents the number of neurons in it.
- The width of the network, $max_t|V_t|$, represents the maximum number of neurons in a layer.

The feedforward NN's structure of depth 2, size 10, and width 5 is presented in figure 2.3.

2.5.2 Training of a Neural Network

The training description in this section will follow the training of a feedforward NN, which is more straightforward to visualize than the other architectures presented in later sections. The generalization of different architectures can be done. New implementations in the evaluations of loss and gradients in the training process exist, but all follow the same general idea presented here. The description of the SGD and backpropagation algorithms are based on the book of Shalev-Shwartz, S. and Ben-David, S. (2014) [22] and Goodfellow, I.; Bengio, Y. and Courville, A. (2016) [25].

Once specified by the parameters (V, E, σ, w), the feedforward NN is a function as $h_{V,E,\sigma,w} : \mathbb{R}^{|V_0-1|} \to \mathbb{R}^{V_T}$, i.e., an input returns an output. Any one of these functions can be a hypothesis class for learning.

The usual hypotheses class of neural network predictors is all functions $h_{V,E,\sigma,w}$ in which the triplet (V, E, σ) are fixed. This triplet is called the architecture of the network, and the weights over the edges are the parameters defining a hypothesis in the class. The hypotheses class is presented in equation 2.13.

$$\mathcal{H}_{V,E,\sigma} = h_{V,E,\sigma,w} : w \text{ is a mapping from } E \text{ to } \mathbb{R} \tag{2.13}$$

SGD

The Stochastic Gradient Descent algorithm (SGD) is a heuristic applied in the training of NNs. It minimizes the loss function $L_{\mathcal{S}}(\mathbf{w})$ to find the hypothesis in $\mathcal{H}_{V,E,\sigma}$ tuning the weights over the edges.

Supposing an NN has n inputs and k outputs, and given E is a finite set, i.e., a finite number of edges, it's possible to represent the weights as a vector $\mathbf{w}\ \varepsilon\ \mathbb{R}^{|E|}$. This NN can be denoted as $h_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}^k$, and given a target $\mathbf{y}\ \varepsilon \mathcal{Y}$, the prediction loss is denoted as $\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$. The loss function assumed for the derivations is the squared loss, as presented in equation 2.14. The risk function of the network over the training domain is shown in equation 2.15.

$$\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = \frac{1}{2}\|h_{\mathbf{w}}(\mathbf{x}) - \mathbf{y}\|^2 \tag{2.14}$$

$$L_{\mathcal{S}}(\mathbf{w}) = \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\ \varepsilon\ \mathcal{S}}[\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})] \tag{2.15}$$

The SGD is an extended version of the gradient descendent (GD) procedure. Gradient descendent is an iterative optimization procedure in which each step of the solution of an equality of a function is improved by taking a step in the direction of the negative gradient from the current point. SGD simplifies gradient computation by taking steps in a random direction approximating the expected negative gradient [22]. This makes the algorithm less prone to getting stuck in local minima and reduces computational cost.

Formally, given a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ at the point $\mathbf{w}$, the gradient $\nabla f(\mathbf{w})$ is computed as the vector of partial derivatives of f as in equation 2.16.

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, ..., \frac{\partial f(\mathbf{w})}{\partial w_d}\right) \tag{2.16}$$

The value of $\mathbf{w}$ is updated in every step according to a step parameter η (learning rate), the current point, and the gradient computed. Hence, the update step equation is presented in 2.17. The algorithm starts from an initial random value of $\mathbf{w}$, for example, $\mathbf{w}^1 = 0$. This updated value of $\mathbf{w}$ decreases the value of the function going in the opposite direction of the maximum increase of f around $\mathbf{w}^t$.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla f(\mathbf{w}^t) \tag{2.17}$$

The GD applied to the empirical risk involves the evaluation of "m" times the function's

gradient; hence, the operation's computational cost scales $O(m)$ with the number of training samples [25]. The equation of a final GD for the empirical risk, equation 2.4, is presented in 2.18.

$$\nabla_w L_S(h) := \frac{1}{m} \sum_{i=1}^{m} \nabla_w \Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) \tag{2.18}$$

The difference in the SGD algorithm is that the updating direction may not be precisely the opposite of the gradient of the empirical risk function; hence, it does not use the entire training dataset to evaluate the gradient. This is helpful because machine learning needs extensive training data to generalize effectively. Consequently, such large datasets also demand significantly more computational resources to compute the gradients of the model in the training [25].

In the iterations using SGD, only a minibatch of n samples from the training dataset, B = $x^{(1)}, \ldots, x^{(n)}$, is considered. The final equation for calculating the gradient using SGD is presented in 2.19. Using the compute gradient *grad*, the estimated new $\mathbf{w}$ is evaluated as in equation 2.17 [25]

$$grad := \frac{1}{n} \sum_{i=1}^{n} \nabla_w \Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) \tag{2.19}$$

There are important details when applying SGD to the NNs, including possible enhancements to the GD algorithm. These enhancements are not considered as a discussion scope in this thesis. The backpropagation algorithm is the solution for computing the gradient for the loss for the entire NN.

Backpropagation

Backpropagation allows NNs to learn from data by efficiently computing each parameter's impact on the final predictions during training. Backpropagation computes the gradient of the NN's loss function, exploiting the model's construction in layers and working backward to correct each parameter from the final loss. In the following brief explanation, the mechanism of computing the gradient of the loss function on an example $(\mathbf{x}, \mathbf{y})$ w.r.t. the vector $\mathbf{w}$ is presented.

The minima search in this section will be described using the sigmoid function, equation 2.9, and the loss function as the squared loss, but any derivation using a differentiable scalar function as activation and a differentiable function for loss similarly holds [22]. All derivations

and definitions were based on the book of Shalev-Shwartz, S. and Ben-David, S. (2014) [22].

An essential definition for understanding the backpropagation algorithm is the Jacobian of $\mathbf{f}$ at $\mathbf{w}\ \varepsilon\ \mathbb{R}^n$, given that f is a function with multiple m outputs $\mathbf{f}$: $\mathbb{R}^n \rightarrow \mathbb{R}^m$. The Jacobian of $\mathbf{f}$ at $\mathbf{w}\ \varepsilon\ \mathbb{R}^n$ denoted as $J_{\mathbf{w}}(\mathbf{f})$, is an m $\times$ n matrix of partial derivatives in which the i^{th} row is related to the i^{th} output variable and the j^{th} column is associated with the j^{th} variable at $\mathbf{w}$. Given f_i: $\mathbb{R}^n \rightarrow \mathbb{R}$ the i^{th} output function, the element at i, j of the Jacobian matrix is presented in 2.20.

$$J_{i,j} = \frac{\partial f_i}{\partial w_j} \tag{2.20}$$

The chain rule propriety for the Jacobian given two functions $\mathbf{f}$: $\mathbb{R}^n \rightarrow \mathbb{R}^m$ and $\mathbf{g}$: $\mathbb{R}^k \rightarrow \mathbb{R}^n$ and their composition (f $\circ$ g): $\mathbb{R}^k \rightarrow \mathbb{R}^m$ is presented in equation 2.21.

$$J_{\mathbf{w}}(f \circ g) = J_{g(\mathbf{w})}(f) J_{\mathbf{w}}(g) \tag{2.21}$$

The backpropagation exploits the decomposition of the neural network in layers to compute the gradient. For every layer t in the NN, denote as $V_t = v_{t,1}, \ldots, v_{t,k_t}$ the set of $k_t = |V_t|$ neurons and as $W_t\ \varepsilon\ \mathbb{R}^{k_{t+1},k_t}$ the weight matrix, which gives the values of the connections of layer V_t to V_{t+1}. If the edge E exists, the $W_{t,i,j}$ is the weight of the edge $(v_{t,j}, v_{t+1,i})$, and when the edge is not present, the weight is set to 0.

Computing the partial derivatives w.r.t. edges from V_{t-1} to V_t is computing them w.r.t. the elements in W_{t-1}. With all other weights fixed, the outputs of neurons in layer V_{t-1} become fixed numbers represented by the vector $\mathbf{o}_{t-1}$. A loss function for the subnetwork of layers V_t to V_T can be defined based on the output of the neurons in V_t. This loss function is denoted by $l_t : \mathbb{R}^{k_t} \rightarrow \mathbb{R}$ and can be written as in equation 2.22.

$$g_t(W_{t-1}) = l_t(\mathbf{o}_t) = l_t(\sigma(\mathbf{a}_t)) = l_t(\sigma(W_{t-1}\mathbf{o}_{t-1})) \tag{2.22}$$

The previous results are written in a transpose form, reducing the weight matrix to a vector for convenience. Let $\mathbf{w}_{t-1}\ \varepsilon\ \mathbb{R}^{k_{t-1}k_t}$ be the reduced row vector by concatenating the rows of W_{t-1}. The output vector is transformed to a $k_t \times (k_{t-1}k_t)$ matrix O_{t-1}, whose values in the diagonal are the transposed $\mathbf{o}_{t-1}$ vector. This rearrangement modifies the input of the neurons in the layer V_t as $W_{t-1}\mathbf{o}_{t-1} = O_{t-1}\mathbf{w}_{t-1}$, and the loss function of the layer becomes equation

2.23.

$$g_t(\mathbf{w}_{t-1}) = l_t(\sigma(O_{t-1}\mathbf{w}_{t-1})) \tag{2.23}$$

Besides the chain rule in terms of Jacobian, two results are used to compute the Jacobian of the loss function:

- Given $\mathbf{f}(\mathbf{w}) = A\mathbf{w}$ for $A \varepsilon \mathbb{R}^{m,n}$. $J_{\mathbf{w}}(\mathbf{f}) = A$
- For every n, given the notation σ to the function $\mathbb{R}^n$ to $\mathbb{R}^n$, that applies the sigmoid function element-wise. Therefore denoting $\alpha = \sigma(\theta)$, for every i $\alpha_i = \sigma(\theta_i) = \frac{1}{1+exp(-\theta_i)}$. The Jacobian $J_\theta(\sigma)$ is a diagonal matrix with entry $\sigma'(\theta_i)$. The derivation of the scalar sigmoid function is presented in equation 2.24.

$$\sigma'(\theta_i) = \frac{1}{(1 + exp(\theta_i))(1 + exp(-\theta_i))} \tag{2.24}$$

The Jacobian of the loss function in the layer V_t is computed using the chain rule and presented in equation 2.25.

$$J_{\mathbf{w}_{t-1}}(g_t) = J_{\sigma(O_{t-1}\mathbf{w}_{t-1})}(l_t) diag(\sigma'(O_{t-1}\mathbf{w}_{t-1}))O_{t-1} \tag{2.25}$$

In terms of output and input values of the layer, the equation is simplified to 2.26.

$$J_{\mathbf{w}_{t-1}}(g_t) = J_{\mathbf{o}_t}(l_t) diag(\sigma'(\mathbf{a}_t))O_{t-1} \tag{2.26}$$

Denoting $\boldsymbol{\delta}_t = J_{\mathbf{o}_t}(l_t)$, the Jacobian can be rewritten as equation 2.27.

$$J_{\mathbf{w}_{t-1}}(g_t) = (\delta_{t,1}\sigma'(a_{t,1})\mathbf{o}_{t-1}^T, ..., \delta_{t,k_t}\sigma'(a_{t,k_t})\mathbf{o}_{t-1}^T) \tag{2.27}$$

The gradient of l_t at $\mathbf{o}_t$, $\boldsymbol{\delta}_t$, is calculated recursively. Starting from the last layer, V_T, the loss is computed using the NN output $\mathbf{u}$ and the true labels $\mathbf{y}$ as $l_T(\mathbf{u}) = \Delta(\mathbf{u}, \mathbf{y})$. Assuming the loss function is $\Delta(\mathbf{u}, \mathbf{y}) = \frac{1}{2}||\mathbf{u} - \mathbf{y}||^2$, $J_{\mathbf{u}}(l_T) = (\mathbf{u} - \mathbf{y})$. In the last layer, $\delta_T = J_{\mathbf{o}_T}(l_T) = (\mathbf{o}_T - \mathbf{y})$. Noting equation 2.28 and using the chain rule, the Jacobian of the loss function on the layer V_t can be written using the loss function on the layer V_{t+1}, and the final result is expressed in equation 2.29.

$$l_t(\mathbf{u}) = l_{t+1}(\sigma(\mathbf{W}_t u)) \tag{2.28}$$

$$\delta_t = \delta_{t+1} diag(\sigma'(\mathbf{a}_{t+1}))\, W_t \tag{2.29}$$

This result for the Jacobian and the definition of the computation of the vectors a_t, o_t in the NN fully define how to compute the elements of equation 2.27. Therefore, the Backpropagation algorithm pseudo code is the following, extracted from [22], which can be divided into two macro steps: forward and backward. The first is the computation of all inputs and outputs of neurons run from the input to the output layer; the second is the computation of all Jacobians from the output to the input layer.

Backpropagation for NNs
input: an example (**x**, **y**), weight vector **w**, $\mathcal{H}_{V,E,\sigma}$
forward:

set $\mathbf{o}_0 = \mathbf{x}$

for t from 1 to T:

for i from 1 to k_t

set $a_{t,i} = \sum_{j=1}^{kt-1} W_{t-1,i,j} o_{t-1,j}$

set $o_{t,i} = \sigma a_{t,i}$

backward:

set $\delta_T = \mathbf{o}_T$ - **y**

for t = T-1, T-2, ..., 1

for i = 1, ..., k_t

$\delta_{t,i} = \sum_{j=1}^{k_{t+1}} W_{t,j,i} \delta_{t+1,j} \sigma'(a_{t+1,j})$

output

each edge partial derivative of $(v_{t-1,j}, v_{t,i})$ is set to $\delta_{t,i} \sigma'(a_{t,i}) o_{t-1,j}$

The final algorithm of training a Neural Network, extracted from [22], is presented, considering the combination of the SGD and the backpropagation, using mini-batches and updating the weight parameters using n training examples in each iteration.

Training for Neural Networks
parameters: Number of iterations τ; step size sequence $\eta_1, \eta_2, \ldots, \eta_\tau$; regularization parameter $\lambda > 0$
input: $\mathcal{H}_{V,E,\sigma}$
initialize: choose $\mathbf{w}^{(1)}$ ε $\mathbb{R}^{|E|}$ at random s.t. $\mathbf{w}^{(1)}$ is close enough to 0
for i from 1 to τ:

sample ($\mathbf{x}$, $\mathbf{y}$) from training sample
calculate $\mathbf{v}_i$ = backpropagation($\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$, $\mathcal{H}_{V,E,\sigma}$)
update $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta_i(\mathbf{v}_i + \lambda \mathbf{w}^{(i)})$

output $\bar{w}$ = best performing $\mathbf{w}^{(i)}$ on validation set

Deep learning libraries such as TensorFlow let the user specify the number of samples the model must consider updating its weights. The common terminology in the libraries is "Batch size" for the number n of training examples presented before updating the weights and Epoch for the number of iterations the training set shows to the NN until the end of the learning process.

Overfitting and underfitting

It's possible to observe that the number of parameters in the NN predictors class can be considerable. This can lead to overfitting during the learning process, which is a challenge that must be addressed with this type of predictor.

Overfitting is a natural path that the learning may lead and can be described as a learner reaching a predictor that has an excellent performance in the training set but a very bad one in the true "world." This happens when the hypothesis chosen by the learner in the hypothesis class is the one that fits the training set very well and cannot be generalized to new samples. The validation set is used to identify the scenario of overfitting and escape from it.

Overfitting is detected in the training by monitoring the loss in a validation set. The validation set contains samples that the model does not access during training and is used to estimate the model's true loss. Different strategies can be used to separate the collected data into training and validation sets. Overfitting starts when the training error of the NN keeps declining in the learning process, but the validation error starts increasing.

There are some computationally cheap methods to avoid overfitting:

- The dropout method consists of a technique that randomly "removes" neurons during the training from the network. At each step, the network is trained with a subset of neurons, which reduces the number of weights and avoids the output depending "too much" on a single neuron. The dropout probability is uniform and independent between neurons and is a hyper-parameter that needs to be tuned.

- Regularizing the weights involves adding a term in the loss function as a penalization for having a high sum of the weights. An L1 or L2 term of the weights can be used, and the hyper-parameter λ can be tuned to determine the penalty's size.

- Monitoring the training and validation loss is valuable to identify the right moment to stop the learning process. This method to avoid overfitting is called early stopping, and it uses the validation error to decide to stop the training.

The opposite of overfitting is underfitting, which occurs when the hypothesis class is not complex enough to fit the problem satisfactorily or when the learning algorithm fails to find a suitable hypothesis. Underfitting due to insufficient model complexity can be mitigated by adding neurons or layers to the network architecture. When building NN models, selecting the best hypothesis class between the topologies trained is common. The selection is based on the validation error, and typically, there is a test set to estimate the final true risk of the fully defined NN model.

2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialization of NNs to treat grid-like topology data, e.g., images and time-series data. It is a type of neural network where convolution replaces the usual matrix multiplication in one or more layers. The primary material used in the description of this section is based on the book of Goodfellow, Ian; Bengio, Y.; and Courville, Aaron (2016) [25] and on the course presented by Andrew NG et al. on Coursera about Convolutional Neural Networks inside of the Deep learning specialization provided by DeepLearning.AI [26].

In the 1990s, AT&T's team crafted a CNN for checks reading. Post-2000, Microsoft leveraged CNNs for OCR and handwriting, broadening commercial utility. Specializing in grid-structured data, CNNs excel in two-dimensional images and permit scalable model growth [25].

Convolutional neural networks utilize three key concepts to enhance machine learning performance: sparse connectivity, parameter sharing, and equivariance. Through sparse interactions, CNN contrasts with traditional systems where outputs interact with all inputs. Each output interacts with a limited set of inputs in the Convolution layer, which uses small kernels for the connections [25]. Deep networks, with many Convolution layers stacked, allow deeper units to connect with vast input sections so richer representations can be learned indirectly. Stridden convolution and pooling further expand this reach[25].

Furthermore, the parameter sharing in these layers enhances memory use, computational cost, and statistical efficiency, and the convolution's equivariance property ensures output shifts equally to input alterations. Parameter sharing is efficient and logical as it identifies features like edges universally as they appear in many parts of the image [25].

2.6.1 Convolution and Convolution in NNs

Convolution, a mathematical operation on two real-valued function arguments, is exemplified through the equation 2.30 and commonly denoted as 2.31. In convolutional network terminology, the first argument, function x, is the input, and the second, function w, is the kernel. The output is often referred to as the feature map [25].

$$s(t) = \int x(a)w(t-a)\mathrm{d}a \tag{2.30}$$

$$s(t) = (x * w)(t) \tag{2.31}$$

When x and w are defined on integer t, the discrete convolution can be defined as presented in equation 2.32.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.32}$$

In machine learning, multidimensional arrays of data and parameters serve as input and kernel. The latter one has its values optimized by the learning algorithm. Convolutions on multiple axes employ a two-dimensional image I and kernel K, and its computation is presented in equation 2.33 [25].

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m,j-n). \tag{2.33}$$

Cross-correlation, presented in equation 2.34, diverges from convolution by not flipping the kernel and is often mislabeled and used in machine learning libraries and implementations [25].

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i+m,j+n)K(m,n) \tag{2.34}$$

The figure2.4 presents an example of convolution given an input of 3x3 and a kernel, or filter, with the size of 2x2. The image is not padded, p = 0, and the stride equals 1 in both directions, s = (1, 1). Padding and striding are presented in a later section; the first is related to adding values in the input borders to increase the locations where the kernel may be applied, and the second is the number of indexes the kernel is moved in each computation [26].

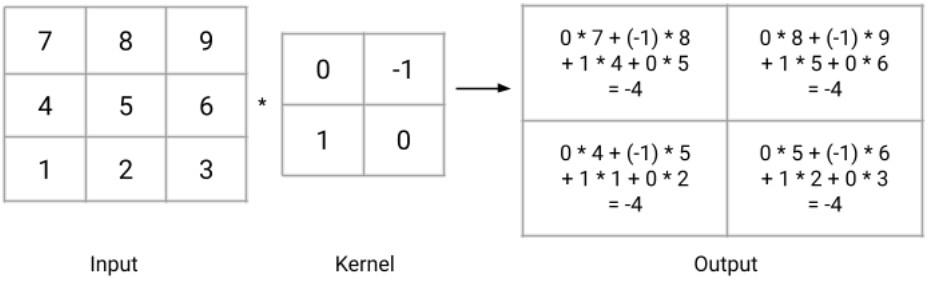


Figure 2.4: Convolution operation example. Kernel 2x2, input 3x3, s = (1, 1), and p = 0.

The convolution process in an NN can be expressed by a 4D kernel tensor, K, with elements $K_{i,j,k,l}$ indicating the connection between a unit in the output channel i and another in the input channel j. This connection considers an offset of k rows and l columns. Given the observed data I, they are represented by elements $I_{i,j,k}$, the input unit's value in channel i at row j and column k. The output, Z, is analogous to I. The value of Z is outputted by the convolution of K across I, without flipping K, as presented in equation 2.35 [25].

$$Z_{i,j,k} = \sum_{l,m,n} \left[I_{l,j+m-1,k+n-1} K_{i,l,m,n} \right] \tag{2.35}$$

In CNNs, 'convolution' embodies parallel operational applications. While one kernel discerns a feature type across spaces, a network layer aspires to capture diverse features from multiple regions. This is achieved in a single layer by stacking together many kernels, resulting in i output channels independent of the number of input channels.

2.6.2 Pooling

Convolutional Networks typically involve three stages. Initially, parallel convolutions produce linear activations. A nonlinear activation function then passes these activations. Finally, a pooling function modifies the output. This last pooling step is not necessarily applied, but almost all CNNs employ it as it speeds up computation and makes features more robust [26] [25].

A pooling layer summarizes the outputs of neurons in a local neighborhood by computing an aggregate statistic. Among these statistics options, max pooling identifies the maximum value in a rectangular local zone. Others include averaging within this local zone, the L2 norm, or a weighted average based on pixel distance [25]. A critical characteristic of the pooling layer is that it doesn't have weights to be adjusted in the training process, so sometimes the convolutional and pooling layers are considered just one convolutional layer [26]. Figure 2.5 presents a max pooling example using a filter of 2x2.

Pooling's strength lies in its ability to maintain invariance to slight input translations, meaning the pooled outputs remain mostly unchanged when the input is shifted slightly. If the presence of a feature is more important than its precise location, invariance to small translations can be an advantageous characteristic in the network [25]. When pooling over distinct convolution outputs, features can learn specific transformation invariance, for example, rotation.

The pooling function is essential as it adapts the network to varying input sizes and reduces

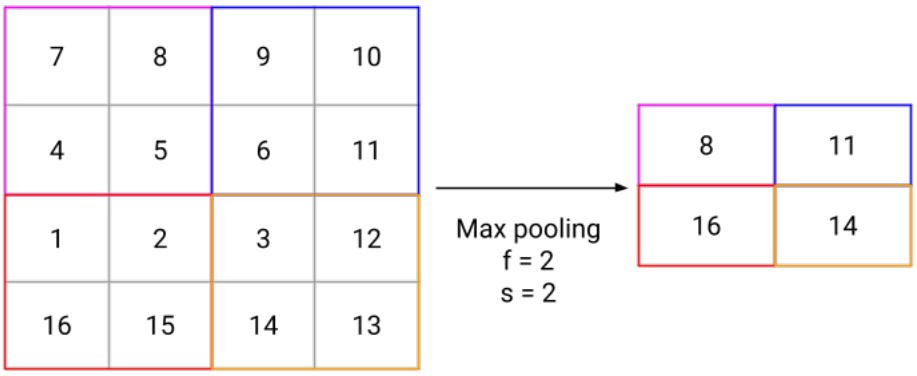


Figure 2.5: Pooling operation example with filter 2x2 and s = (2, 2).

computational cost when used to downsize the input dimension [25]. Generally, the images in CNNs are shrunk in size (width and height) and enlarged in channels using pooling layers [26].

2.6.3 Stride and padding

Zero-padding (or simply padding) in convolutional networks preserves the input I's size and enhances the network's capability, allowing output size control independently of the kernel by adding p pixels around the input. "Valid" convolution, where kernels are entirely inside the input as padding is not considered, provides an output of m - k + 1, i.e., the final outputs' dimensions are reduced compared to the input ones. "Same" convolution matches input-output sizes, while "full" convolution, visiting each pixel k times, offers m + k - 1 size. These types of padding control the convolutional layers' limit expanse, with an optimal padding level typically ranging between valid and full convolution [25].

The convolution can further downsample the input dimensions by applying a stride (s) greater than one with an option for different values for each direction. The stride value sets how many indexes the kernel is moved in each computation, skipping some positions, which reduces the computational cost but compromises the feature extraction. Figure 2.7 shows an example of the convolution computation using a stride of (1, 1) and (2, 2); it is essential to notice the effect of shrinkage of data when stride is used. The downsampled convolution formula

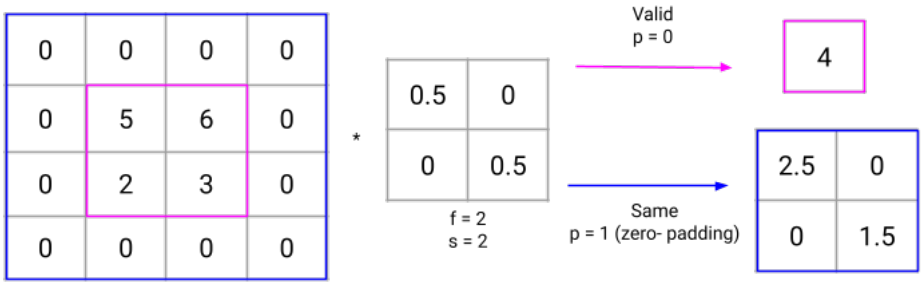


Figure 2.6: Valid and Same padding examples.

is presented in 2.36 considering the same stride value for row and column direction.

$$Z_{i,j,k} = c(K, I, s)_{i,j,k} = \sum_{l,m,n} \left[I_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right] \tag{2.36}$$

The final output size of the layer given the padding (p) and stride values (s), the kernel size (f), and the input size (n) can be computed using the relation presented in 2.37 [26].

$$j, k = \lfloor \frac{n + 2p - f}{s} + 1 \rfloor, \lfloor \frac{n + 2p - f}{s} + 1 \rfloor \tag{2.37}$$

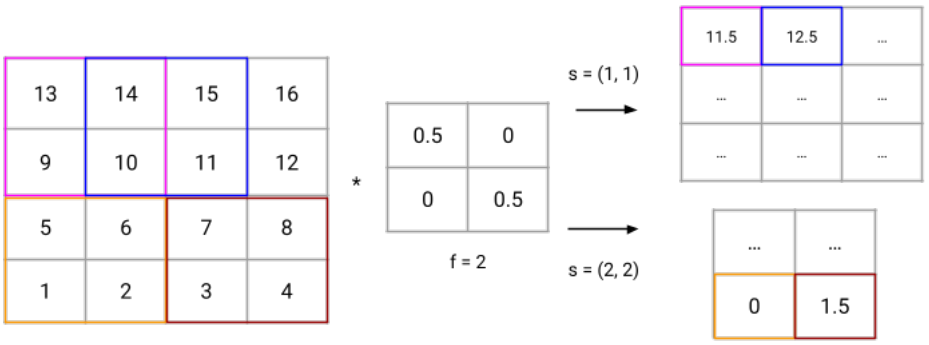


Figure 2.7: Stride example, s = (1, 1) and s = (2, 2).

2.6.4 Backpropagation in CNN

Three operations — convolution, backprop from output to weights, and backprop from output to inputs — are crucial for training deep convolutional networks [25].

The training aims to minimize a loss function $L_S(I, K)$. In forward propagation, convolution transforms the input image into output tensors Z used in L_S calculation. Backpropagation then calculates G, which is the gradient of the loss function with respect to the output tensor, defined by expression 2.38 [25]. A convolutional network that uses a strided convolution with kernel stack K on a multi-channel image I with stride s is considered.

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} L_S(I, K) \tag{2.38}$$

Training needs the computation of the derivatives with respect to the kernel weights using the expression 2.39 [25]. This is essential for updating the convolutional kernels during the learning process.

$$g(G, I, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} L_S(I, K) = \sum_{m,n} G_{i,m,n} I_{j,(m-1)\times s+k,(n-1)\times s+l} \tag{2.39}$$

If the layer isn't the lowest in the network, equation 2.40 computes the gradient concerning the input I for further backpropagation [25]. It is essential to update the weights of previous layers in the networks.

$$h(K, G, s)_{i,j,k} = \frac{\partial}{\partial I_{i,j,k}} L_S(I, K) \tag{2.40}$$

Considering the convolutional network with stride s and padding p, the expression for the gradient can be detailed as expression 2.41 [25].

$$h(K, G, s)_{i,j,k} = \sum_{\substack{l,m \\ \text{s.t.} \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{s.t.} \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n} \tag{2.41}$$

2.7 Computer vision

Convolutional networks can scale to large-sized NNs and have proven highly successful for two-dimensional image inputs. Over time, various designs or 'architectures' of these networks have emerged for computer vision tasks. This section explores some key architectures, including the specific model used in this thesis. The primary material used in this section is from the course presented by Andrew NG et al. on Coursera about Convolutional Neural Networks inside of the Deep Learning specialization provided by DeepLearning.AI [26].

Computer vision is a broad field that addresses various problems. Three significant areas within this domain that are relevant to the scope of this thesis are image classification, object detection, and semantic segmentation [26].

- Image Classification: This task involves assigning a label to an entire image based on its content. For instance, given a photo, the model determines whether it depicts a cat, dog, car, or other object. It doesn't provide the location or count of the objects;
- Object Detection: This is more detailed than image classification. It not only identifies the objects present in an image but also determines their boundaries. It can involve multiple objects of multiple classes in the image. For example, the model will highlight each car's location using bounding boxes in a picture with several cars;
- Semantic Segmentation: Every pixel in an image is labeled with a class. For example, in an autonomous car's camera image, the model will label pixels to indicate if they belong to the road, a pedestrian, a vehicle, the sky, a tree, etc.

2.7.1 Object detection: bounding boxes and IoU

Bounding box dimensions are relative to the image size. These boxes locate and encapsulate the detected objects, visually representing the detection [26]. The boxes can be located by their center or bottom left, and the height and width fully characterize them in the 2D image. Figure 2.8 presents a bounding box locating a statue in the picture; the box is shown with a green line, the center of the box is identified with the coordinates (x, y), and width and height are presented as well.

One metric often used when evaluating object detection algorithms is the Intersection Over Union (IoU). It serves a dual purpose as a measuring tool to assess the accuracy of object local-

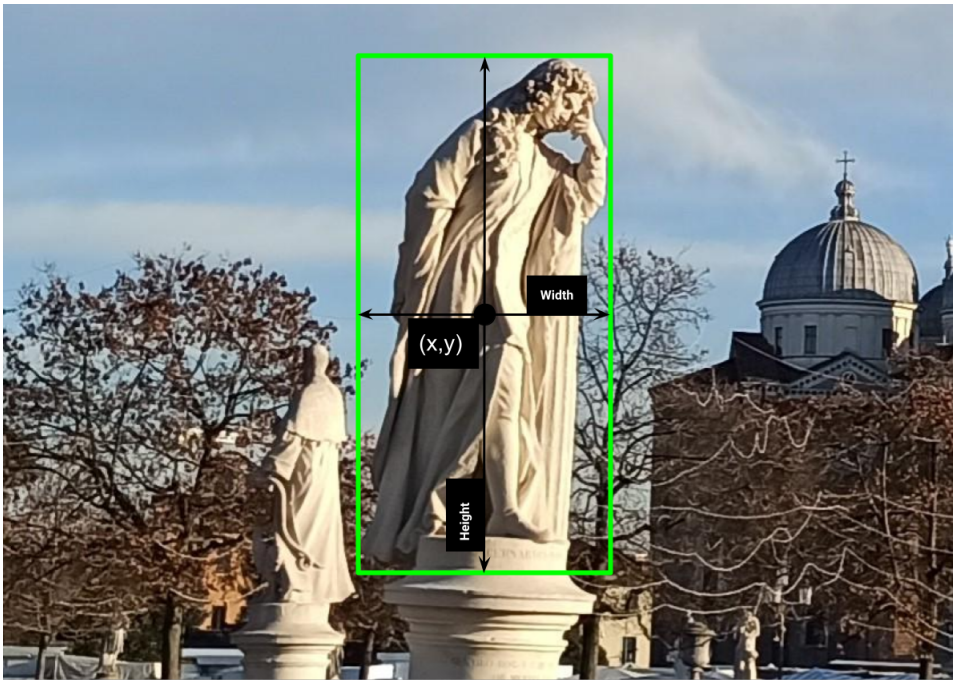


Figure 2.8: Bounding box example, locating a statue.

ization and can be used as a feature within the models to refine predictions. The formula to compute IoU is presented in 2.42, given two bounding boxes, A and B.

$$IoU(A, B) = \frac{\text{Area of Intersection}(A, B)}{\text{Area of Union}(A, B)} \tag{2.42}$$

Typically, an IoU value of 0.5 is considered standard for object detection. While stricter criteria can be applied, values lower than this are uncommon.

2.7.2 Case studies

In deep learning, case studies and published models are essential sources to start investigating the solution of a problem. For example, recent deep learning models for computer vision typically require vast resources and data to be trained, so utilizing the work previously done is crucial for the success of such a model. Some already trained networks for computer vision are available to be used, and using these pre-trained models constitutes an excellent head start to solve a problem [26]. Four architectures exemplify the progression in deep learning for computer vision: LeNet-5, VGG, ResNet, and Yolo.

LeNet-5

LeNet-5, presented in the seminal paper of LeCun et al., 1998 [27], as cited in [26], was a pioneering CNN designed explicitly for handwriting and character recognition, laying the groundwork for future CNNs in image recognition tasks. The model has around 60,000 parameters, making it relatively lightweight compared to later models. The network is presented in image 2.9 [26].

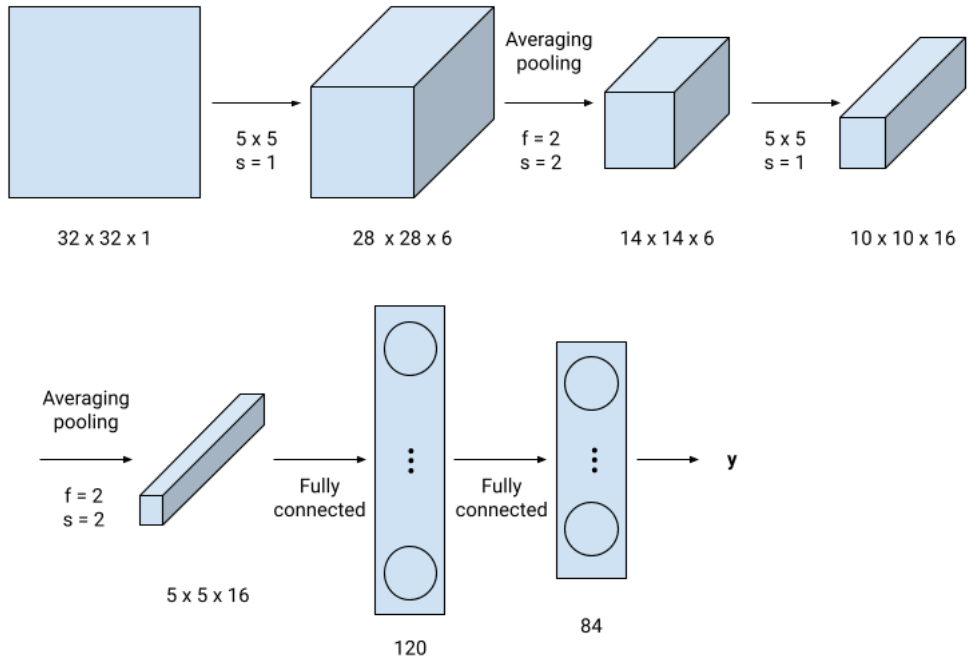


Figure 2.9: LeNet-5 architecture. Figure adapted from Ng, Katanforoosh, and Mourri (2023) [26].

The CNN can be described in tabular form, which presents the sequential layers, informing the number of filters or neurons and other important information about the architecture. The LetNet-5 is shown in table 2.1; it is possible to check that the numbering of the layers is sequential and the pooling and convolutional (CONV) layers are considered just one.

It is essential to notice how the convolution layers are stacked one after another, shrinking the image height and width and increasing the number of channels. After several convolutional layers, the network transitions to fully connected layers, which process the extracted features. The final layer uses a softmax activation to provide probabilities for each potential output. For instance, the network is designed to recognize digits, giving the likelihood of the image being

Layer	Number: filters/neurons	Filter size	Stride/padding	Output size
Input	-	-	-	32 x 32 x 1
Conv 1	6	5 x 5	s = 1, p = valid	28 x 28 x 6
Avg. Pooling 1	-	2 x 2	s = 2	14 x 14 x 6
Conv 2	16	5 x 5	s = 1, p = valid	10 x 10 x 16
Avg. Pooling 2	-	2 x 2	s = 2	5 x 5 x 16
Conv 3	120	5 x 5	s = 1, p = valid	120
Fully connected 1	120	-	-	84
Fully connected 2	84	-	-	10

Table 2.1: LetNet-5 architecture. Table adapted from Ng, Katanforoosh, and Mourri (2023) [26].

each digit from 0 to 9 [26].

VGG

VGG-16 is a deep convolutional neural network (CNN) designed for large-scale image recognition presented by K. Simonyan and A. Zisserman (2015) [28], as cited in [26]. The '16' in VGG-16 refers to the number of layers with trainable weights. There's also a VGG-19 variant and while it has more layers, its performance is comparable to VGG-16 without significant improvements [26]. The VGG-16 NN is presented in table 2.2.

The network doubles the number of filters at each layer, increasing its depth and ability to capture intricate details while decreasing the width and height of the data. The NN consistently uses 3x3 filters with a stride (s) of 1, implementing a max pooling afterward with a 2x2 filter and stride of 2, effectively reducing the data tensor size. The network has 138 million parameters to be trained [26]. The final two layers are fully connected and end in a final layer that uses a softmax activation to provide probabilities for each potential output [26].

ResNet

ResNet (Residual Network) is designed for large-scale image recognition. It's notably deep, presented by He et al. (2015) with a version containing 152 layers [29], as cited in [26]. The optimization algorithm can struggle with very deep architectures in traditional neural networks. Deeper networks can sometimes perform worse due to this difficulty in training. ResNet's design, with its residual blocks and skip connections, counters this problem, making very deep networks more effective [26].

Skip Connections are "shortcuts" or "residual connections." Instead of sending data straight

Layer	Number: filters/neurons	Filter size	Stride/padding	Output size
Input	-	-	-	224 x 224 x 3
2 x Convolution	64	3 x 3	s = 1, p = same	224 x 224 x 64
Max Pooling	-	2 x 2	s = 2	112 x 112 x 64
2 x Convolution	128	3 x 3	s = 1, p = same	112 x 112 x 128
Max Pooling	-	2 x 2	s = 2	56 x 56 x 128
3 x Convolution	256	3 x 3	s = 1, p = same	56 x 56 x 256
Max Pooling	-	2 x 2	s = 2	28 x 28 x 256
3 x Convolution	512	3 x 3	s = 1, p = same	28 x 28 x 512
Max Pooling	-	2 x 2	s = 2	14 x 14 x 512
3 x Convolution	512	3 x 3	s = 1, p = same	14 x 14 x 512
Max Pooling	-	2 x 2	s = 2	7 x 7 x 512
Fully connected 1	25088	-	-	4096
Fully connected 2	4096	-	-	4096
Fully connected 3	4096	-	-	1000

Table 2.2: VGG-16 architecture. Table adapted from Ng, Katanforoosh, and Mourri (2023) [26].

through every layer, ResNet occasionally lets data skip some layers. Specifically, the activation from one layer is added to a later layer's activation, usually before a ReLU activation function and after the linear transformation. This process allows the term $a^{[l]}$ to skip layers; the notation of superscript $[l]$ indicates the output belongs to the l^{th} layer [26].

The residual block diagram is presented in figure 2.10, and a toy example of a 4-layer residual NN is shown in image 2.11. The $a^{[l]}$ value that skips the $l+1$ layer in the figure is summed before the activation function of the layer $l+2$, the equation 2.43 extracted from [26] presents the computation of $a^{[l+2]}$ in the residual block.

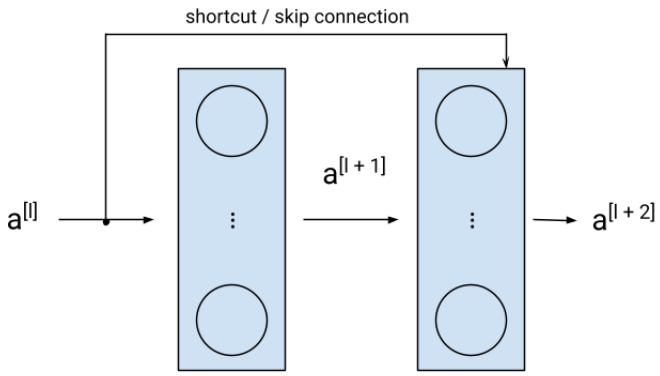


Figure 2.10: Residual block: shortcut or skip connection. Figure extracted from Ng, Katanforoosh, and Mourri (2023) [26].

$$a^{[l+1]} = \sigma\left(w^{[l+1]}a^{[l]} + b^{[l+1]}\right) \tag{2.43}$$

$$a^{[l+2]} = \sigma\left(w^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]}\right) \tag{2.44}$$

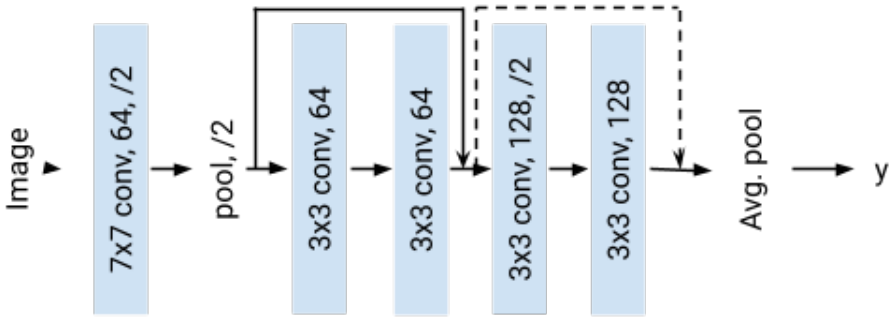


Figure 2.11: Residual Network. Figure adapted from Ng, Katanforoosh, and Mourri (2023) [26].

CNN to object detection

Traditional CNNs (Convolutional Neural Networks) usually provide a single output for an image, classifying it into a particular category. However, if identifying an object and determining its location in an image is needed, the standard approach of these CNNs falls short [26]. A potential solution is the "sliding window detection" method. The training of such CNN should use tightly cropped images of the object of interest. For instance, if identifying cars, use images cropped closely around cars. The goal is for the object to occupy most of the image.

The trained CNN is then applied to the various cropped portions of new images by sliding a window across the image. The CNN predicts the likelihood of the object's presence within each window. The object's exact location is identified by examining the CNN's predictions for each window. This method requires the CNN to evaluate numerous windowed portions of an image. Using larger strides or window sizes can speed up the process but may sacrifice accuracy as the object might not fit correctly in the window [26].

In 2014, Sermanet et al. [30], as cited in [26], introduced an approach to optimizing the sliding window method by processing the entire image in one go through the CNN as a convolution implementation of the sliding window. Although the process presented by Sermanet et al. was computationally effective, the disadvantage lay in the accuracy of the bounding box, as

the problem of a not perfectly matching window to the object is still present in the approach [26].

The YOLO model presented in the next section and used in this thesis overcomes both problems of computational efficiency and bounding box accuracy, as both properties are essential to the problem of characterizing the STRIKE calorimeter, as discussed in previous sections.

2.7.3 YOLO

YOLO, or "You Only Look Once," redefines object detection using a single neural network to predict bounding boxes and class probabilities straight from image pixels. As it discerns object sizes and shapes, it is a robust method. Remarkably, it learns very general representations of objects, offering superior performance compared to other methods, and is extremely fast in evaluations [21].

YOLO method reasons globally about images, enabling it to implicitly encode contextual information about class nuances and appearance. Unlike other techniques, such as sliding windows or region proposals, YOLO views the entire image during training and testing. While Fast R-CNN often misidentifies background patches as objects due to a lack of larger context vision, resulting in background errors, YOLO has fewer than half such mistakes. Notably, when YOLO trains on natural images and tests on artwork, it vastly outperforms top detectors like DPM and R-CNN, demonstrating a more robust generalization [21].

The model was first developed and proposed by Redmon et al. in 2015 [21], as cited in [26]. Many versions of the model were released with many improvements over time. Still, only the main concepts needed to grasp how versions of YOLO work are presented here, i.e., bounding boxes, anchor boxes, and non-max suppression. The general loss function and the architecture of the first version (YOLOv1) [21] and the version used in this thesis are also presented in more detail.

YOLO divides an image into a grid, often using finer grids like 19 x 19, and instead of separate models for classification and localization, YOLO employs a unified vector. This vector is assessed for each grid cell in the image, and its general form, given C possible object classes, is presented in equation 2.45 [26].

$$\text{Prediction} = [\text{confidence}, b_x, b_y, b_w, b_h, p_1, p_2, \ldots, p_C] \tag{2.45}$$

Where:

- $[b_x, b_y, b_w, b_h]$ are the bounding box attributes;

- confidence is confidence in the presence of an object;
- $p_1, p_2, \ldots, p_C$ are the class probabilities.

YOLO might be used to predict specific bounding boxes around detected objects. It's versatile and capable of recognizing various box ratios. The fundamental principle is that each object gets assigned to only one grid cell where the object's center lies. Using finer grids, YOLO minimizes the chances of having multiple object classes within a single cell [26]. The Anchor boxes concept to be described later enables YOLO to predict multiple objects within a single cell [31].

Unlike methods that might require running a separate algorithm for each grid cell, as discussed in the previous section, YOLO's architecture involves a single convolutional network performing computations for all cells simultaneously. This design choice enables YOLO to operate at remarkable speeds, making it apt for real-time object detection [26]. All the advantages described in speed, localization, and generalization made this model unique for the characterization of STRIKE.

Non-max suppression

A challenge in object detection using YOLO is that the same object can be detected multiple times, resulting in overlapping bounding boxes. Non-max suppression (NMS) is an algorithm designed to address this issue. Its primary function is to reduce the multiple identifications of the same object to a single bounding box. For example, by dividing an image into 19 x 19 boxes, each box could detect the same object but with varying confidence levels (probabilities) [26].

The NMS algorithm removes all low-probability predicted bounding boxes given a threshold. This threshold is chosen in the prediction step and can be a parameter to regulate the model prediction quality [26]. In the second step, the algorithm highlights the box with the highest probability among the remaining ones. All boxes with a high IoU overlap with this selected box (typically an overlap of 0.5 or more, but it can also be set in the prediction step) are suppressed. The idea is that these suppressed boxes are likely detecting the same object [26].

After the highest probability box has been selected and overlapping boxes suppressed, the process is repeated for the next highest probability until all boxes are either suppressed or identified as predictions [26]. The NMS algorithm gives the YOLO model two parameters that can be chosen in the prediction step: the threshold for the lowest probability and the acceptable IoU between bounding boxes.

Anchor boxes

The introduction of Anchor boxes occurred in YOLOv2 [31]. The YOLOv8 used in this thesis doesn't utilize Anchor boxes but a different approach to predict many scales inside a grid cell [32]. The description of Anchor boxes is maintained here. It is an easier way to understand how a computer vision model would detect overlapping objects of different scales in the same grid cell.

Traditionally, each grid cell predicted only one class of object. If multiple objects fall within the same grid cell, the system struggles to identify them separately. Anchor boxes are predefined shapes that can represent different types of objects. The detection system can modify its predictions using these boxes, considering multiple objects within a single grid cell [26]. For every grid cell, there's a prediction for each anchor box. Objects are paired with a grid cell (based on the object's midpoint) and an anchor box. The anchor box selected is the one that most closely matches the object's shape, determined by the Intersection over Union (IoU) metric [26].

For example, if two anchor boxes are used, each grid cell will predict two distinct bounding boxes. It's essential to note that sometimes, these bounding boxes might extend beyond the boundaries of the grid cell. Figure 2.12 presents the example of the prediction of the two anchor boxes in a grid cell. The anchor boxes are moved to the predicted centers in the image but not resized, and the prior centers do not necessarily need to be equal for all anchor boxes; the image only highlights how the shape of each anchor box might help identify different types of objects in the image.

The vector of prediction when two anchor boxes are applied is presented in equation 2.46 [26]. The subscript i denotes predictions associated with the i^{th} anchor box.

$$\begin{aligned} \text{Prediction} = [&\text{confidence}_1, b_{x1}, b_{y1}, b_{w1}, b_{h1}, p_{11}, p_{12}, \ldots, p_{1C}, \\ &\text{confidence}_2, b_{x2}, b_{y2}, b_{w2}, b_{h2}, p_{21}, p_{22}, \ldots, p_{2C}] \end{aligned} \tag{2.46}$$

Anchor boxes can be tailored to fit the objects in the dataset better. One method is manual selection, choosing common shapes in the data. Alternatively, a more data-driven approach involves using k-means clustering. This method groups common object shapes together, providing a clearer picture of the prevalent object shapes in the dataset.

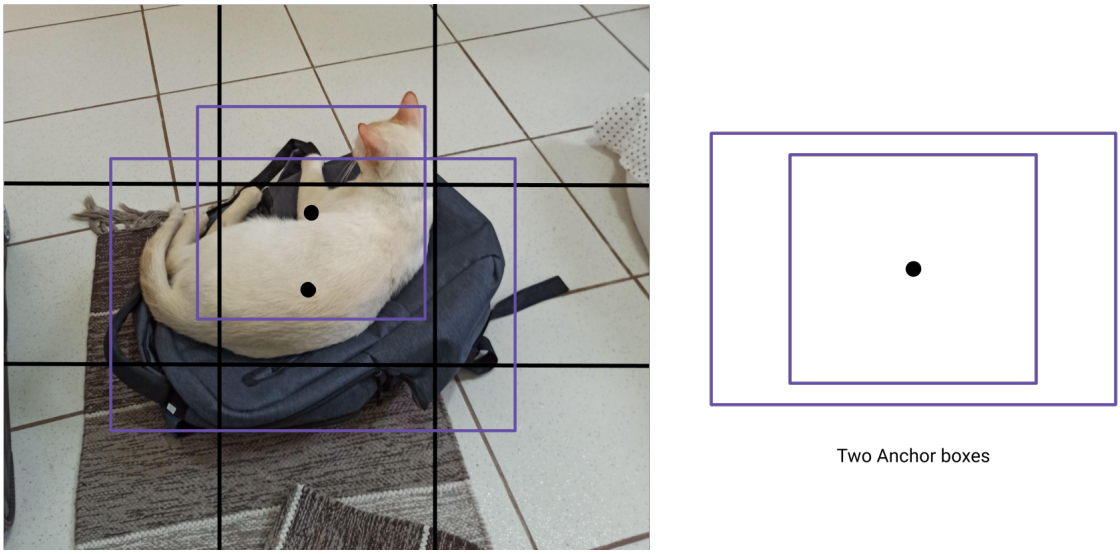


Figure 2.12: Two Anchor boxes represented considering objects with their center in the central grid cell. Figure created based on Ng, Katanforoosh, and Mourri (2023) [26]

Architecture YOLOv1

YOLOv1 comprises a single NN to extract relevant features from the images. The first version of YOLO was built using a CNN with 24 convolutional layers followed by two fully connected layers [21], presented in table 2.3.

The YOLO algorithm received many improvements over time, arriving at version 3 with its creators Redmon, J.; Farhadi, A. [33] and being developed up to version 8 by other researchers. This thesis used version 8, developed by Ultralytics [34]. References for all versions of YOLO can be accessed in [35].

Architecture YOLOv8

YOLOv8 architecture utilizes two major elements that are referred to as backbone and head. The backbone is the CNN responsible for the feature extraction of the images; the head is the NN responsible for the final prediction of classification and localization [32]. As the CNN of YOLOv8 is more complex, with many skip connections and split operations, it is more suitable to present it in a diagram format instead of the tabular form of the other NNs. The diagram of the YOLOv8 model, considering all five variations of size, is presented in figure 2.13. The table in the "details" part of the diagram describes the difference between the CNN sizes, from nano (n) to extra large (x).

Layer	Number: filters/neurons	Filter size	Stride/padding	Output size
Input	-	-	-	448 x 448 x 3
Convolution	64	7 x 7	s = 2, p = same	224 x 224 x 64
Max Pooling	-	2 x 2	s = 2	112 x 112 x 64
Convolution	192	3 x 3	s = 1, p = same	112 x 112 x 192
Max Pooling	-	2 x 2	s = 2	56 x 56 x 192
Convolution	128	1 x 1	s = 1, p = same	56 x 56 x 128
Convolution	256	3 x 3	s = 1, p = same	56 x 56 x 256
Convolution	256	1 x 1	s = 1, p = same	56 x 56 x 256
Convolution	512	3 x 3	s = 1, p = same	56 x 56 x 512
Max Pooling	-	2 x 2	s = 2	28 x 28 x 512
4 x Convolution	[256 (1 x 1), 512 (3 x 3)]	1 x 1, 3 x 3	s = 1, p = same	28 x 28 x 512
Convolution	512	1 x 1	s = 1, p = same	28 x 28 x 512
Convolution	1024	3 x 3	s = 1, p = same	28 x 28 x 1024
Max Pooling	-	2 x 2	s = 2	14 x 14 x 1024
2 x Convolution	[512 (1 x 1), 1024 (3 x 3)]	1 x 1, 3 x 3	s = 1, p = same	14 x 14 x 1024
Convolution	1024	3 x 3	s = 1, p = same	14 x 14 x 1024
Convolution	1024	3 x 3	s = 2, p = same	7 x 7 x 1024
Convolution	1024	3 x 3	s = 1, p = same	28 x 28 x 1024
Convolution	1024	3 x 3	s = 2, p = same	14 x 14 x 1024
2 x Convolution	1024	3 x 3	s = 1, p = same	7 x 7 x 1024
Fully connected	50176	-	-	4096
Fully connected	4096	-	-	7 x 7 x 30

Table 2.3: YOLOv1 architecture. Table adapted from the original paper of Redmon (2015) [21].

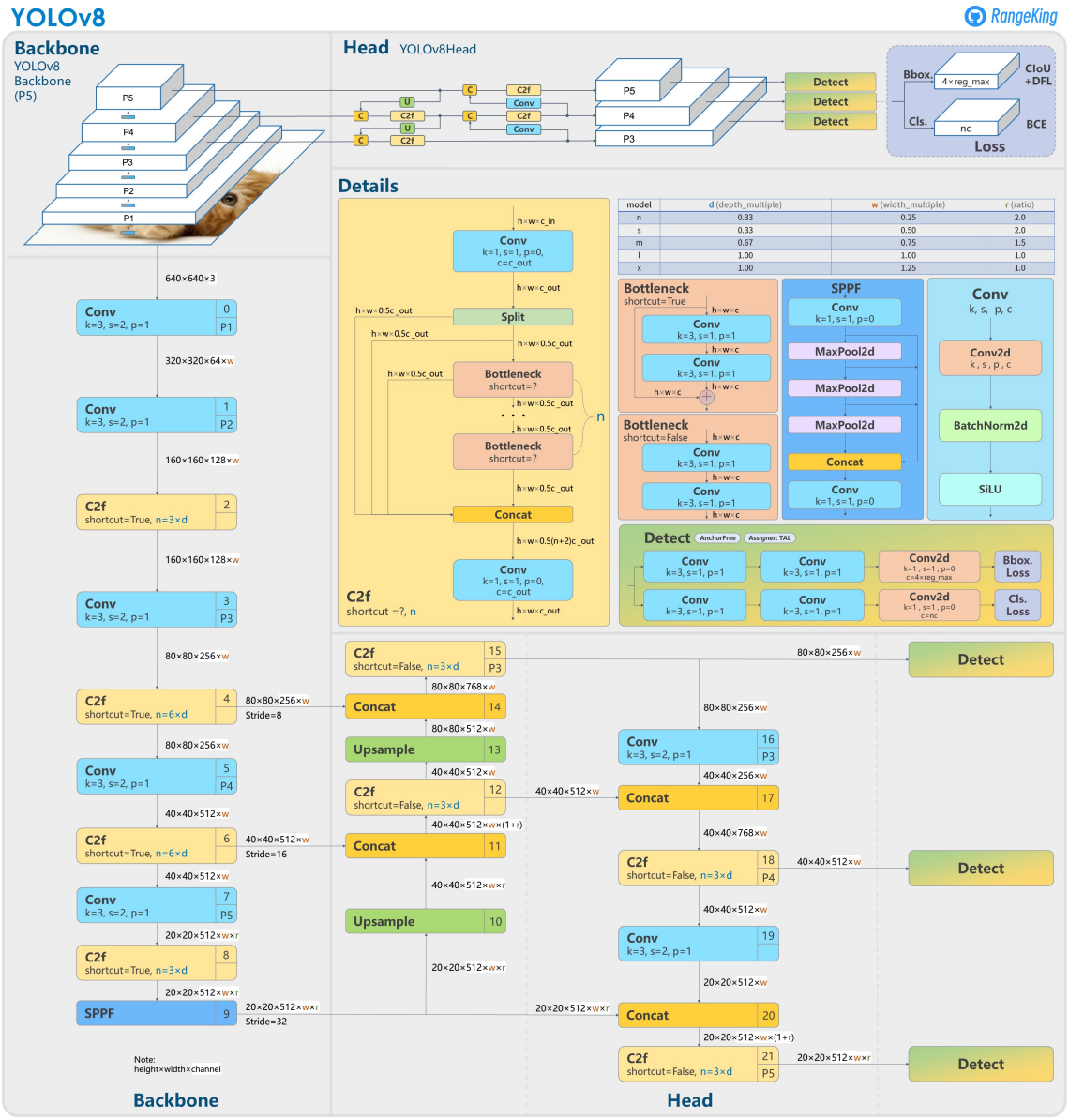


Figure 2.13: YOLOv8 model diagram, extracted from [32].

Loss function YOLOv1

The loss function described in the first version of YOLOv1 considered a combination of three types of losses related to the coordinates, objectness, and classification. Each loss is accounted for in the final error metric and is presented in equations 2.47 to 2.51, extracted from [21].

The S^2 value is the number of grid cells in the method, and B is the number of bounding boxes predicted for each grid cell. The indicator $1^{\text{obj}_{ij}}$ is used in the equations to penalize only if an object is present in the grid cell and the $1^{\text{noobj}_{ij}}$ when not. As many grid cells do not contain objects, the losses are weighted by the parameters λ_{coord} and λ_{noobj}, the latter one lower than the first one [21].

- The bounding box center loss:

$$L_{\text{center}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1^{\text{obj}_{ij}} \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right) \tag{2.47}$$

  where(x_i, y_i) is the true center coordinates of the object; and $(\hat{x}_i, \hat{y}_i)$, the predicted ones.

- The bounding box dimension loss and the square of the width and height are done to address the problem that the error in larger boxes should affect the metric less than the error in small ones:

$$L_{\text{dimension}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1^{\text{obj}_{ij}} \left(\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right) \tag{2.48}$$

  where (w_i, h_i) is the true width and height of the bounding box, respectively, and the $(\hat{w}_i, \hat{h}_i)$ are the predicted ones.

- The objectness confidence loss for object:

$$L_{\text{conf\_obj}} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1^{\text{obj}_{ij}} \left(C_i - \hat{C}_i \right)^2 \tag{2.49}$$

  where C_i is the true objectness confidence score, and $\hat{C}_i$ is the predicted one.

- The objectness confidence loss for no object:

$$L_{\text{conf\_noobj}} = \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1^{\text{noobj}_{ij}} \left(C_i - \hat{C}_i \right)^2 \tag{2.50}$$

where C_i is the true objectness confidence score, and $\hat{C}_i$ is the predicted one.

- The class probability loss:

$$L_{\text{class}} = \sum_{i=0}^{S^2} 1^{\text{obj}_i} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2 \tag{2.51}$$

where $p_i(c)$ is the true class probability for class c and $\hat{p}_i(c)$ is the predicted one.

The total YOLOv1 loss is just the sum of these losses, and it is presented in equation 2.52 [21].

$$L_{\text{YOLO}} = L_{\text{center}} + L_{\text{dimension}} + L_{\text{conf\_obj}} + L_{\text{conf\_noobj}} + L_{\text{class}} \tag{2.52}$$

Loss function YOLOv8

Ultralitycs YOLO model training is based on three losses, similar to the ones in YOLOv1, and their curves are presented in the training process. The losses are:

- **box_loss**: Regression loss for the bounding boxes measures the error between the predicted coordinates and the ground truth ones. It is a mean squared error [36].
- **cls_loss**: Classification loss incurred from incorrect object class predictions within the bounding boxes. Measures the error between the predicted class probabilities and the ground truth ones. The binary cross-entropy loss is used [36].
- **dfl_loss**: Distribution Focal Loss adjusts the standard focal loss to address the class imbalance in object detection tasks. It dynamically adjusts the weights of each category during training based on their distribution. It uses the predicted and true box geometrical features and compares the distribution of them [32].

Performance Metrics in Object Detection

Different metrics are commonly used in object detection models to evaluate their performance; their definition lies in the need for a unified framework to assess the object detection models, whose performance must be computed in both the classification and localization tasks. Here, an overview of some of the metrics is presented. It is essential to notice that a classification of true positive/false positive of object detection is related to an IoU threshold between the ground truth box and the predicted one [37].

- **Precision:** Ratio of correctly predicted positive observations to the total predicted positives. It is computed as equation 2.53. Precision shows how much the model misclassifies, giving false detections [37].

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{2.53}$$

- **Recall**: Ratio of correctly predicted positive observations to all the actual positives. It's given by equation 2.54. This measures the model's capacity to detect the objects in the image [37].

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2.54}$$

- **Average Precision (AP)**: For each class, an AP score is computed as the area under the precision-recall curve. This provides insights into the trade-off between precision and recall for different thresholds [37].

- **mean Average Precision (mAP)**: This is the average of AP over all classes, providing a single score that summarizes the object detection model's performance. For a dataset with C classes, it's computed as equation [37].

$$\text{mAP} = \frac{1}{C} \sum_{i=1}^{C} \text{AP}_i \tag{2.55}$$

  where AP_i is the average precision for class i.

- **mAP50**: mAP with an IoU threshold of 0.5, denoted as mAP@0.5 alternatively, is a widely used metric in object detection. A predicted bounding box is considered correct if it has an IoU of 0.5 or higher than any ground-truth bounding box [37].

3
Methods

This chapter describes the analysis performed in the synthetic images of two datasets based on STRIKE, DS-I and DS-II. Researchers at RFX provided both datasets, so their generation process is briefly described. Artificial data generation is elaborated in the "Datasets" section. Labels for the supervised learning model are the location and the sizes of bounding boxes around each Gaussian. This labeling process is also described.

The models applied are briefly cited, and any meaningful information regarding the parameters used is described. In this thesis, the final objective was to identify the centers and standard deviations (sigmas) of these Gaussians, and the methods of the Gaussian Mixture Model (GMM) and the YOLO model were tested to achieve this objective. Additionally, the PX modifier and PX modifier 2 were introduced to refine the estimations of the YOLO method, aiming to enhance the center and covariance detections. Finally, an ensemble strategy used to combine the results of two methods is described.

At the end of this chapter, the evaluation errors used to compare the estimation models are discussed, and their equations are provided. All analysis, model training, and prediction were carried out using Python on the Google Colab environment, and the notebooks can be consulted on the GitHub of the author [38].

3.1 Datasets

This section describes the labeling and the image dataset generation for this thesis. Two datasets were generated, differing in the Gaussian parameters. The generated datasets consisted of a 2D map of intensity values, mapped to grayscale images using Matplotlib's default colormap before applying the detection methods. Eighty beamlets approximated as Gaussians were generated separately for both datasets, considering random values for its center and diagonal covariance matrices. The Gaussians didn't have rotation.

The Gaussians were expected to be distributed in 5 columns and 16 rows. Hence, the centers of the generated Gaussians were set based on a random offset from this expected grid. Gaussian standard deviations in the x and y direction were expected to be bounded, so the sampled values from a random distribution reflecting this prior were used.

Ideally, all beams would have identical shapes and currents, so the Gaussians should have the same amplitude and minimal x and y deviations, appearing as concentrated peaks in the heat map. However, beam variations were introduced, causing the Gaussians to differ in amplitude and spread with some overlapping and becoming entangled, making them hard to distinguish.

The first dataset (DS-I) was created to challenge the identification methods, so large deflections, amplitude variances, and divergences were present in the images. The second dataset (DS-II) considered the expected beamlets' parameters from SPIDER in its generation, presenting low deflection and divergence and a more uniform amplitude in its images.

The DS-I presented 800 images with substantial differences between sets of 200 images. Thus, this dataset was split into four groups with 200 images in the experiments. Each group was identified as letters in the later section of the results. Accordingly, all methods were evaluated for five subsets, considering the four with 200 images and an extended one combining the first two subsets into a 400-image group.

The images in this first dataset had dimensions of 377 x 143 pixels. Each pixel represented roughly 2 mm in the STRIKE tiles [14]. As it had high deflection values, some Gaussian had its centers lying outside the image or were entirely outside. Only Gaussians with its center in the image were considered in the analysis.

The second dataset (DS-II) presented uniform characteristics between the whole 400 images, so no grouping of subsets was constructed. The methods were evaluated once for this dataset, which was used entirely. It had a lower resolution than the DS-I images, presenting only 180 x 72 pixels per image.

3.1.1 Labeling

The labeling for the Gaussians was constructed from the parameters used in their generation. The bounding box centers were the Gaussian centers, and the width and height were three times the x and y standard deviations (sigmas or σ), which covered 99% of the Gaussian area. Therefore, each Gaussian in the image received a five-number label: one for its class (0 - Gaussian), two for the center location, and two for the box size.

3.1.2 Errors evaluation

Different errors were evaluated, focusing on the Gaussians localization and amplitudes and the number of objects identified—the localization errors were computed separately in the x and y directions. The computation of errors required pairing predicted and true Gaussians in the image, so a matching algorithm was used after the models' predictions. All errors are relative, and their value is reported in percentage in the tables of the results section.

The applied matching algorithm found the optimal one-to-one assignment of Gaussians by minimizing the total squared distance of the centers between matches in the image. This optimization was accomplished using the Scipy library optimize.linear_sum_assignment. Any true Gaussians not matched to an estimated one were excluded from the error calculation, so unidentified Gaussians did not contribute to the reported errors of the methods. Any falsely identified Gaussian did not affect the errors computed as well.

The center error metric is presented in expression 3.1.

$$E_{Center_i} = \frac{1}{m} \sum_k \frac{\hat{C}_{i,k} - C_{i,k}}{C_{i,k}} \tag{3.1}$$

where m is the number of true Gaussians matched in the image, $C_{i,k}$ is the i^{th} center coordinate for the k^{th} Gaussian with $i = x, y$, and k = 1,...,m, and $\hat{C}_{i,k}$ is the predicted one.

The standard deviation error metric is presented in expression 3.2.

$$E_{\sigma_i} = \frac{1}{m} \sum \frac{\hat{\sigma}_{i,k} - \sigma_{i,k}}{\sigma_{i,k}} \tag{3.2}$$

where m is the number of true Gaussians matched in the image, $\sigma_{i,k}$ is the standard deviation in i^{th} direction for the k^{th} Gaussian with $i = x, y$, and k = 1,...,m, and $\hat{\sigma}_{i,k}$ is the predicted one.

The difference between the true number and the estimated number of Gaussians was calculated for each image. This difference provides a rough error measurement for the global identi-

fication accuracy as it doesn't consider a minimum IoU to match predicted and true Gaussians. The expression used in the computation is presented in equation 3.3.

$$E_{\#} = \frac{\text{Number of identified Gaussians}}{\text{Number of Gaussians identifiable}} \tag{3.3}$$

where the Number of Gaussians identifiable were the Gaussians with its center inside the image, and the Number of identified Gaussians were the number of predicted Gaussians in the image.

The amplitude in the image, as it was in pixel scale, needed to be converted back to the intensity values used in the datasets. The original image at the predicted Gaussian centers was used to get the actual intensity values. The amplitude error metric is presented in 3.4.

$$E_{\text{A}} = \frac{1}{m} \sum \frac{\hat{A}_k - A_k}{A_k} \tag{3.4}$$

where m is the number of true Gaussians matched in the image, A_k is the amplitude for the k^{th} Gaussian with k = 1,...,m, and $\hat{A}_k$ is the predicted one.

Another error evaluated but not as important in comparing the methods was the reconstruction error based on the mean squared error of pixels. This pixel-level error metric evaluates how well the estimated Gaussians can recreate the original image. Lower errors indicate more accurate Gaussian fitting. To compute this, the image is reconstructed using the estimated Gaussian parameters, and the difference between the reconstructed and original image is calculated for each pixel in each Gaussian region. The reconstruction error metric is presented in expression 3.5.

$$E_{\text{Reconstruction}} = \frac{1}{m} \sum_{k} \sum_{P \, \varepsilon \, bb_k} \frac{(\hat{P} - P)^2}{n(bb_k)} \tag{3.5}$$

where m is the number of true Gaussians matched in the image, bb_k is the predicted bounding box for the k^{th} Gaussian with k = 1,...,m, P are the pixels' value inside bb_k, and $\hat{P}$ is the predicted value.

3.1.3 Dataset splitting

Each evaluation comprised three subsets of the dataset used in the study of each method. The splitting was related to dividing the data into training, validation, and test datasets. The proportion used was 70%, 20%, and 10%, respectively.

The validation dataset is used during training to measure the model's performance in generalizing. As it is used in the training, it can't be used later to estimate the true risk of the model. The test dataset is held out from the training to assess the true risk of the model.

It is important to note that the GMM is an unsupervised learning method. Therefore, it was directly applied to the test dataset, not using the training and validation datasets in the methods.

The test dataset is the same for all methods to guarantee an equivalent estimation of the true risk between them, and all errors reported are based on the set of images belonging to it. The numbering of images for each dataset is registered in the appropriate result section.

3.2 GMM method

In the Gaussian Mixture Models (GMMs), an image is modeled as being generated from a mixture of Gaussian distributions, as described in section 2.4.3. GMM fitting allows passing initial estimates for the Gaussian parameters like centers and standard deviations.

Without initial estimates, the model randomly initializes the parameters. However, reasonable initial estimates are essential for converging to an accurate solution, so the estimated centers of the Gaussians were passed to the method. The standard deviations were randomly initialized, and the Gaussians' covariance type was chosen as *diag*; for more details, check 2.4.3.

The initial center estimates were obtained by applying the peak detection algorithm from the scikit-image library, peak_local_max, to a Laplacian-filtered version of the image. The peak_local_max function finds the highest peak within a local region of the data, ignoring other lower values. This helps identify the center of a Gaussian curve in a sub-region of the image, even if parts of different Gaussian curves are also present in that region.

The Laplacian from scipy library, ndimage.laplace, was applied to the images to highlight areas of rapid intensity change. The filtered images comprised pixels with a Laplacian value greater than or equal to 0, indicating a Gaussian peak or edge.

As the filtered image highlighted the centers (peaks) of the Gaussians, it enabled better detection than on the raw image by the peak_local_max function. Using these estimated centers to initialize the model helped guide the fitting process compared to random initialization.

The x and y coordinates of pixels exceeding a Laplacian threshold of 0 were used as the input of the model fitting, as the GMM model in the sklearn only considers the x and y coordinates. The number of Gaussians that the model used in the fittings was the number of identified peaks by the peak_local_max function.

As discussed in section 2.4.3, GMM is an unsupervised learning model, and no labeling about the dataset was passed to the model. The final output of the GMM model is the centers of the Gaussians and the covariances. The amplitude value for each Gaussian was considered as the intensity value in the object's predicted center.

3.3 YOLO method

A pre-trained YOLO model on the COCO-120 dataset, "yolov8s.pt," available on the library of Ultralytics [34], was retrained on the labeled images of Gaussians in the datasets. The model was trained in six different datasets and used to predict the sampled images. Therefore, the evaluation metrics in the Results chapter were reported identifying the dataset the model was trained by coupling "YOLO" with the identification of the dataset. This suffix is suppressed when evident from the context or when the model is being discussed generally, so the model is only identified as "YOLO."

After 100 training epochs, the training was stopped, and the training metrics were evaluated to check the final model performance. In one of the datasets, the model was trained for 200 training epochs to assess the improvement of the performance metrics. The batch size was passed as "-1" to the model, meaning the algorithm maximized the number of images in each batch to the capacity in memory of the GPU.

The confidence and intersection over union (IoU) thresholds are key parameters when using YOLO. The confidence threshold filters out detections with a low likelihood of accuracy. The IoU threshold is used to merge overlapping bounding boxes that result from multiple detections of the same object. The predictions in this thesis used the default values provided by the library for these parameters, 0.25 and 0.7, for confidence and IoU, respectively.

The input for the YOLO model was the grayscale images, and the output was the bounding box coordinates. Each Gaussians' parameter set could be determined using the bounding box location and size as the labeling process allowed this by construction. The amplitude value for each Gaussian was the intensity value in its predicted center in the image.

To train the YOLO, each image needed labels in this format:

- Column 1: The class ID - use as 0 for all Gaussians;
- Columns 2 and 3: The x, y coordinates of the bounding box center;
- Columns 4 and 5: The width and height of the bounding box.

This labeling structure allows YOLO to learn to detect the objects and localize them with bounding boxes during training. The bounding box coordinates predicted by the YOLO algorithm should be normalized to the range [0, 1] considering the image sizes, so this allows the predictions to be standardized across varying dimensions of images.

The YOLO folder structure requires separate train, validation, and test subsets. Each subset contains an images folder and a labels folder. This latter one presents the information structures described above for each Gaussian in the image so that the file would have more or less eighty lines with five columns.

3.4 PX modifier and PX modifier 2

Considering the characteristics of the Gaussian distribution, a refining method for the YOLO predictions was developed to adjust its estimated centers and standard deviations. The technique was decomposed into a first stage of center adjustment and a second stage of standard deviation estimation. Each step was applied in the x and y direction for each identified Gaussian by YOLO. It is important to note that after using the method in the x-direction, the center's coordinates passed to the y-direction step is the initial one, even though it might have occurred an update in the x-coordinate.

The first stage was accomplished by analyzing a square trial region around the first estimated centers. This trial region worked with a step in the grid, set by a *step_size*, and by comparing the amplitudes in the points in this trial zone. If the amplitude at a trial point was higher than the original estimated center, the center was moved to this new coordinate. The value of *step_size* equal 1 was used in the experimentations, as higher values were demonstrated to be highly unstable. If, with the incremental *step_size* from the center, the point was outside the edge, the edge was considered the center to be evaluated.

Using the density formula of Gaussian distribution, 2.1, it was possible to estimate the standard deviation in x and y in the second phase. The equation used in the estimation is presented in expression 3.6.

$$\sigma = \sqrt{-\frac{(x_{i\pm3} - x_i)^2}{2log\left(\frac{f(x_{i\pm3})}{f(x_i)}\right)}} \tag{3.6}$$

The formula of estimations depended on the amplitude ratio between the peak of the Gaussian and another point. The estimate used a *step_size* of 3 pixels, chosen through a trial and error procedure. Therefore, the amplitude ratio compared in the second phase was between the center and pixels 3 steps above/below.

The variances were calculated from points above and below the center. These were averaged to estimate the final variance. If the computed variance was negative, the point 3 steps away had a higher amplitude likely caused by overlapping Gaussians. Only positive variance values were averaged in this case, so negative variances were excluded.

The method was later modified to a PX modifier 2 version, in which the lowest variance was used if the up and down variances differed too much. The distribution of the up and down variance differences for each Gaussian in the image was analyzed, and the outliers were selected

using the IQR method. This new version was based on the observation that a significant difference flags a high Gaussian overlap probably present in the outlier direction. The PX modifier 2 encompasses the expression 3.7 as a post-processing of the estimations from 3.6.

$$\text{if } \hat{\sigma}(x_{i+3}) - \hat{\sigma}(x_{i-3}) \text{ outlier:} \quad \hat{\sigma} = min\left(\hat{\sigma}(x_{i+3}), \hat{\sigma}(x_{i-3})\right) \tag{3.7}$$

3.5 Ensembled

Expecting that the PX modifier 2 would give different errors for different Gaussians compared to YOLO, an ensembled method was proposed. The ensemble was tested only with PX modifier 2 and YOLO, as the PX modifier presented high errors, as reported in the results.

The ensembling was based on the $E_{Reconstruction,k}$, to select the best estimation method for each Gaussian in the image. The $E_{Reconstruction,k}$, was calculated on the cropped region of the *kth* Gaussian for each method, considering the bounding box predicted by each. The errors were then compared for each Gaussian and each method, caring that the same Gaussian was being compared. The model with the lower $E_{Reconstruction,k}$, for the region of the *kth* Gaussian, was used to predict its parameters. Therefore, the prediction in an ensembled image combined Gaussians' predicted parameters of the two methods.

4

Results

This section presents the results of evaluating the provided datasets with the proposed methods. It starts with a brief analysis of the images dataset, an important exercise to understand the variances between the images. Qualitative images of the detection were vastly presented to deepen the discussion about the advantages and weaknesses of each method.

The first dataset (DS-I) was split into four groups of images; each was analyzed using the four proposed methods, and the errors were reported. A study combined two groups (A and B) from DS-I to understand the model's robustness. The YOLO's test error and training metrics for this extended group (DS-IE) were also reported.

The second dataset (DS-II) was analyzed entirely, without following a group splitting as the first one, using the best model selected from the tests in the first dataset (DS-I). YOLO trained in the DS-IE, the extended group of DS-I, was tested, and a new training was performed using images from DS-II. All errors for both YOLO models and the GMM method in the DS-II were reported in this chapter.

To conclude, critical remarks and recommendations for using the models in the future operation of SPIDER are provided, as well as the opportunities for enhancements for the YOLO model, considered the best method developed.

Images / ID	Amplitude	C_x	C_y	σ_x	σ_y
Base / Base	5.57E7	-	-	9.9	5.19
0 - 200 / A	+1.5E6	Same	$\pm$ 10	+ 3	+ 2
200 - 400 / B	+2E6	$\pm$ 4	Same	+ 6	+ 3
400 - 600 / C	+2.1E6	Same	Same	+ 17	+ 10
600 - 800 / D	+2.1E6	$\pm$ 5	Same	+ 17	+ 10

Table 4.1: Groups of images in DS-I and characteristic parameters compared to the base image.

4.1 Dataset analysis

The two datasets provided were initially analyzed in terms of general characteristics of deflection and divergence as an exercise to understand the complexities involved in detecting the Gaussians.

The first dataset (DS-I) was compared with the provided base image. As described in the previous chapter 3, the Gaussians were generated using random offset parameters from this expected base image. The first image in the figure 4.1 presents it.

Table 4.1 presents the mean offsets of the parameters for the four groupings created from the 800 images from the DS-I. The values in the table are presented as the mean increment/decrement of the Gaussians in the images from the ones in the base image. The increment/decrement values for the center and standard deviation (σ) are based on the number of pixels. The center of the base is not provided in the table, as it requires 80 pairs of coordinates, but it can be seen in the figure 4.1. The groups are identified later using the dataset identification concatenated with its ID, e.g., DS-IA.

The figure 4.1 presents a sample of the images in DS-I in grayscale. The images shown are 109, 203, 401 and 601. The base image is presented first; it is possible to observe a very organized 5 x 16 array of Gaussian peaks that are well separated from each other and with a consistently high amplitude. The positioning in the x-axis of the Gaussians centers are all aligned in the columns, and a different positioning in the y-axis is used, maintaining a reasonable distance between Gaussians. The space between columns was also varied to simulate an x and y deflection expected from the SPIDER beamlets.

Analyzing image 109 in the second place, a sample from images in DS-IA, there's a noticeable distortion in the placement of some Gaussians, mainly in the y direction, with intense entanglement of some Gaussians. Not all Gaussians are identified, and some are out of the image due to the high difference in C_y described in 4.1. The amplitudes are consistently high, and in

some places, the amplitude appears much higher due to the overlapping of Gaussians in the y direction.

Analyzing image 203 in the third place, a sample from images in DS-IB, there's a clear overlap of many Gaussians, especially in the x direction, related to the higher σ_x and C_x values. The overlapping causes a less distinct appearance of individual Gaussians and a more continuous luminance profile, but more Gaussians are identifiable than the images from group A.

Analyzing the image in the fourth and fifth places, a sample from images in DS-IC and DS-ID, respectively, almost all Gaussians overlap, causing a uniform luminance profile across the entire image. The overlap is so significant that identifying any Gaussians becomes challenging, likely due to high standard deviation in both the x and y directions.

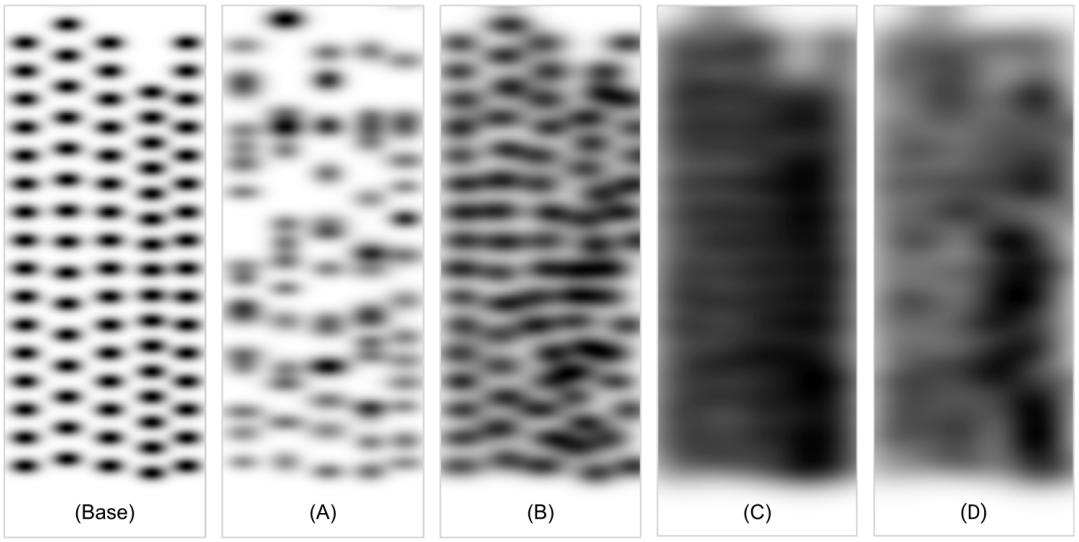


Figure 4.1: Base image and an image example for each group in DS-I, image identification based on Image ID on 4.1.

The challenge of identifying the Gaussians using a Computer Vision technique is entirely different between the groups. The first image on 4.1, the base one, provides a minimal challenge, as the Gaussians are well-separated and distinct, so simple peak detection methods should be enough. The images in groups A and B are trickier to work on, as some Gaussians are closer to each other, making them harder to distinguish.

When Gaussians overlap or entangle, a simple peak detection might mistake closely spaced Gaussians as one. Furthermore, distortions in the placement mean that assumptions about regular spacing can't be applied, requiring more adaptive methods. The summation caused by overlapping generates brighter peaks that can overshadow adjacent lighter ones, causing missed detections.

The images in groups DS-IC and DS-ID lack clear peaks, and it is hard to differentiate between overlapping Gaussians and the background. The detection task is challenging as the image resembles a blurred gradient, hardly resembling any Gaussian structure.

The amplitudes of the Gaussian will be an essential characteristic when applying the methods in the experimental data, as this regulates the contrast between the objects and the background. Higher contrast facilitates detection, and it is better if noise is present, as the noise and Gaussian amplitude difference would be more significant.

Lastly, it is essential to notice that the images used in this thesis have variations in Gaussian width, height, amplitudes, and positioning, as recommended for a training dataset for a model that would need to deal with this generalization in the experimental images.

Figure 4.2 presents the images from 4.1 labeled using the bounding boxes. Those labels were used to train the YOLO model.

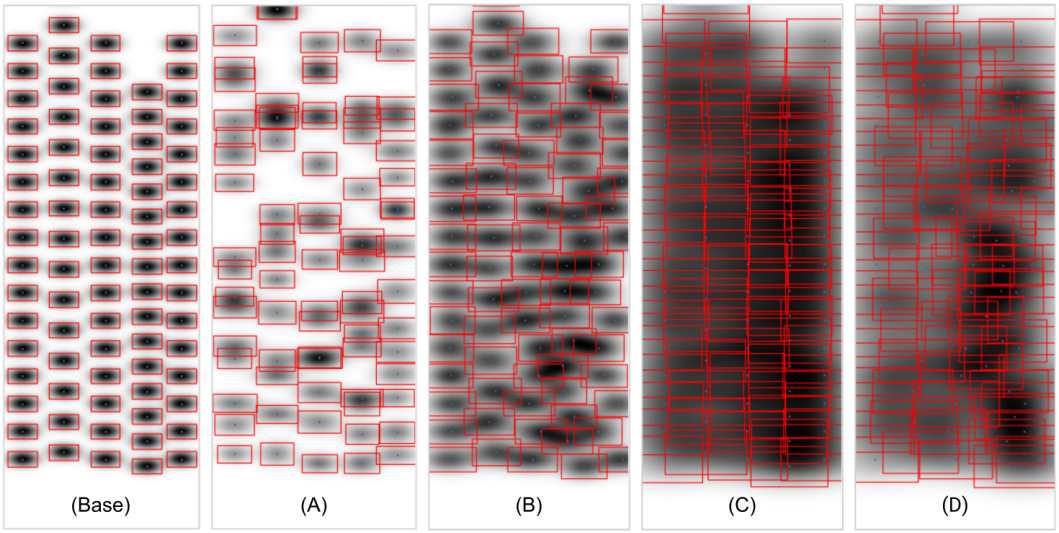


Figure 4.2: Labeled base image and samples for each group in DS-I, image identification based on Image ID on 4.1.

In the second dataset (DS-II), the 400 images provided have a restricted parameter variance, making them appear more rounded than those on the first dataset. A notable limitation is that these images have only a quarter of the resolution of the prior models, with half the pixels in both the x and y directions. All methods tested with the previous Gaussians were also applied to this new dataset.

The figure 4.3 presents two samples from DS-II and the bounding boxes constructed from the ground truth parameters for each Gaussian. It is possible to check that the lower resolu-

tion results in a loss of fine detail compared to the DS-I. The individual Gaussians and their structures are not as sharply defined, and their differences have been somewhat blurred. The amplitudes between Gaussians are also variable, visible by the image's brightness. The Gaussians overlap and merge frequently, making individual object detection more challenging.

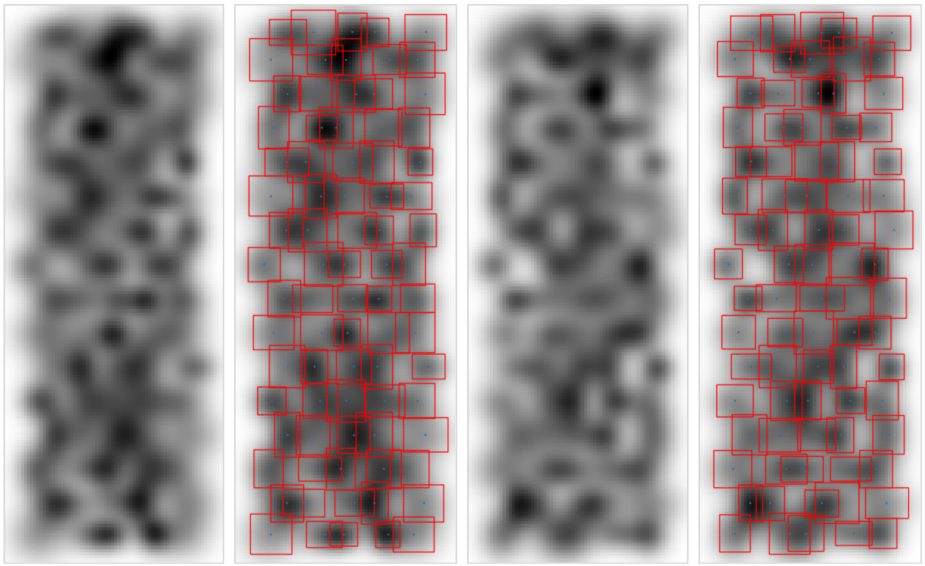


Figure 4.3: Two image examples with labeling in DS-II (a) image 0, (b) image 0 with bounding boxes, (c) image 313, (d) image 313 with bounding boxes.

It is possible to observe from the labeled data with bounding boxes on 4.3 that Gaussians vary in shape and size. Some Gaussians are more circle-like, unlike the Gaussians from DS-I, which all presented an ellipse-like format with a vertical flattening, which was expected as the objects showed this format in the first campaign of SPIDER [2].

4.2 Results first dataset

The performance of the methods in the first dataset (DS-I) is described in this section, and the structuring of the datasets follows the proposed grouping explained in the previous section 4.1.

The figures analyzed in this section are the predictions of the GMM and YOLO methods; each figure presents one image for each dataset: figure 4.4 presents group DS-IA (image 109); figure 4.5 presents group DS-IB (image 203); figure 4.6 presents group DS-IC (image 401); figure 4.7 presents group DS-ID (image 601). Each figure shows: (a) a raw grayscale image, (b) the image with the ground truth bounding boxes and centers, (c) predicted bounding boxes and centers by GMM, and (d) predicted bounding boxes and centers by YOLO.

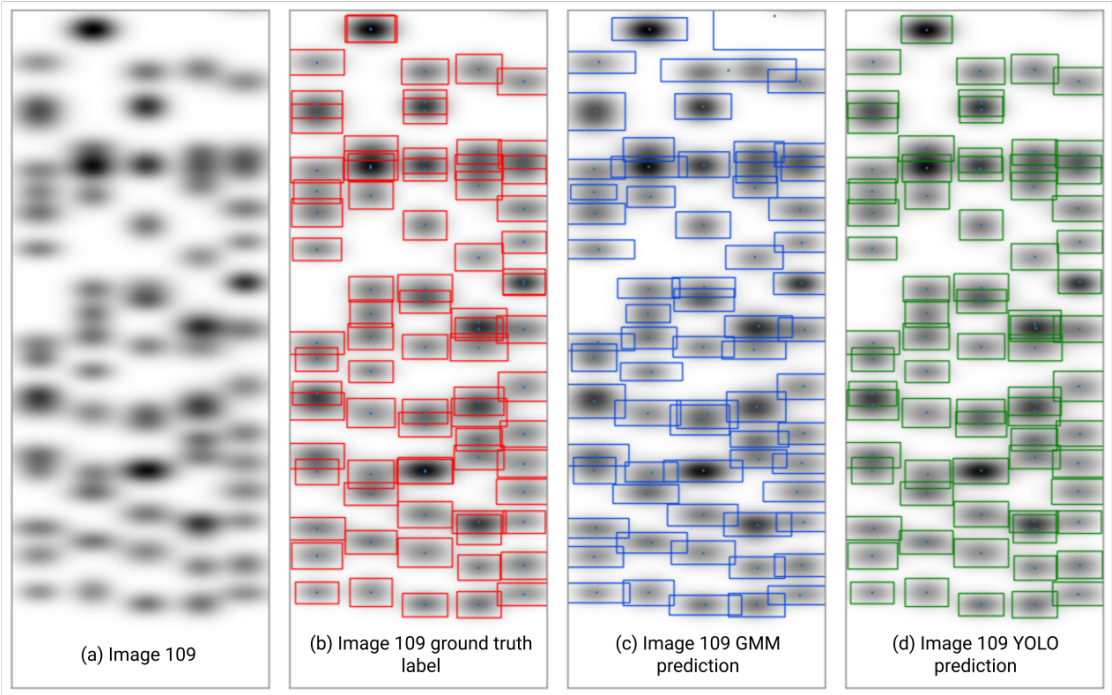


Figure 4.4: Image 109 (DS-IA) detection using GMM and YOLO methods.

By examining figure 4.4, GMM seems to reasonably predict the positions and standard deviations for distinct Gaussians, as seen by the close alignment of several green boxes to their corresponding blue ones. Challenges arise for the method in areas where multiple Gaussians are closely spaced or overlap. In these cases, GMM tends to predict a single, larger bounding box encompassing multiple Gaussians instead of individual ones, interpreting the overlapping Gaussians as one entity. Therefore, it misses some Gaussians.

The bounding boxes for the GMM method tend to overestimate the standard deviation of the Gaussians in the x direction, probably related to the initial shape of the data points. As this seems a generalized problem for all Gaussians in figure 4.4, this can be dealt with a proportional constant to be applied after the prediction to correct the overestimation. Since the chosen points are more dispersed along the x-axis when no Gaussians are overlapping, the estimates for the standard deviation in the x-direction tend to be overestimated and, in the y-direction, underestimated.

There is an apparent absurd false positive Gaussian identification in the top right of figure 4.4. This occurred because a tiny part of an out-of-the-image Gaussian was passed to the GMM algorithm. Recalling that Gaussians that the center is out of the image was not labeled nor considered in the error computations, but parts of it were present in the images and were passed to the methods.

A closer examination of the fitted Gaussian centers for image 109 in figure 4.4 revealed significant shifts in the initial center estimates during the Gaussian Mixture Model's optimization process. As a result, two close yet distinct Gaussians were erroneously combined into one. The combination occurred with the center of the Gaussian positioned between the two original centers. This led to a notable increase in amplitude error because the model didn't accurately represent the true distribution with two distinct peaks. The underlying reason for this problem is that when the GMM model optimizes both centers and covariances, it only treats the provided centers as starting points and can modify them as needed.

The problem in figure 4.4 highlights how the initial conditions or starting points for the GMM might influence its predictions. Different initializations might lead to different model fits, especially in complex scenarios with overlaps. In this thesis, the centers of Gaussians were estimated by the peak local method and given to the GMM algorithm, but the standard deviations were randomly initialized.

One solution for the problem in 4.4 is creating custom code that optimizes the covariances, keeping the initial centers static. This approach would prevent unwanted center shifts, ensuring that closely spaced Gaussians remain distinct and don't combine. Another further implementation that might improve the GMM algorithm is passing estimates for the standard deviation to initialize it. Although promising, the solution was not followed in this thesis, as the GMM method took a long time to converge, and the latter development of the YOLO model was more promising in the detection precision and prediction time.

By examining figure 4.4, the YOLO model effectively identifies and places bounding boxes around most of the individual Gaussians, matching closely with the ground truth. Unlike the

GMM, YOLO handles overlapping or closely spaced Gaussians more efficiently. It can differentiate between closely spaced Gaussians without merging them into a single entity or missing them entirely, drastically reducing the rate of false merges compared to GMM. Furthermore, the bounding boxes predicted by YOLO in 4.4 seem to be tighter and more precise, closely encapsulating the Gaussian. This is a better improvement related to the overestimations in the x direction in GMM.

It is essential to notice that Gaussians with an intense overlapping, e.g., more than fifty percent, are not correctly identified by YOLO. However, the author hardly identifies these overlapping Gaussians in a close visual inspection of (a) in figure 4.4. The existence of some Gaussians is only known after looking at the labeled image (b).

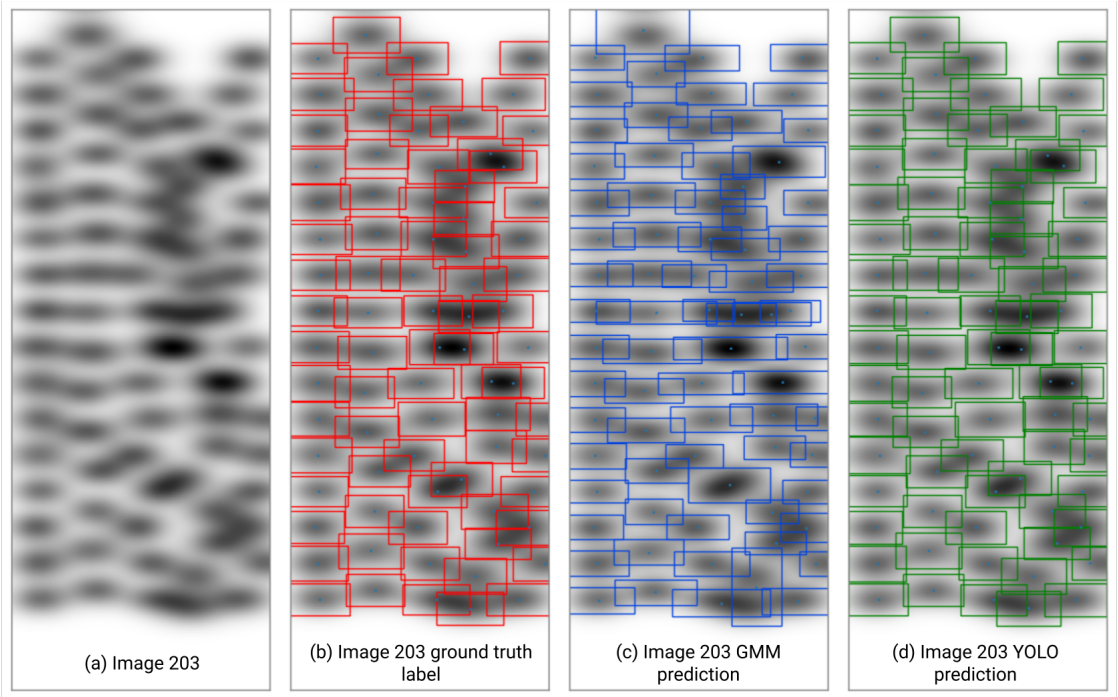


Figure 4.5: Image 203 (DS-IB detection using GMM and YOLO methods.

Analyzing the figure 4.5 shows that the models are challenged with the overlapping Gaussians as well, YOLO being able to detect better in conditions of overlapping and closely spaced Gaussians. On a closer look at the figure, one significant difference from the previous image 109 on DS-IA is that the GMM method produces multiple bounding boxes for a single Gaussian, indicating that the model incorrectly identified more local peaks than the ground truth ones, an opposite problem of the one of merging Gaussians in the prediction. When producing

many predictions for the same ground truth Gaussian, the model increases the false positives, decaying in precision.

The error of false positives highlights one of the disadvantages of GMM as it attempts to fit a specified number of Gaussian distributions to the data that needs to be passed a priori. This choice of the number of Gaussians to fit can impact performance. Too few and it might merge multiple entities; too many might over-segment, as occurred in the prediction in figure 4.5.

It is possible to check as well that from figure 4.5, as the GMM model considers all Gaussian data points to fit the data, one Gaussian at the top is overestimated in the y-direction due to the influence of another Gaussian that its center is out of the image, as the same problem presented and explained from figure 4.4. As these out-of-image Gaussians are not labeled and, therefore, not used in the training process, YOLO usually doesn't predict them.

In image 203, YOLO did a better job than the previous 109, as it dealt better with overlapping Gaussians in the x-direction. No extreme overlappings also occurred; looking at the image (b) in figure 4.5, it is possible to notice that almost all overlaps are less than 50% IoU.

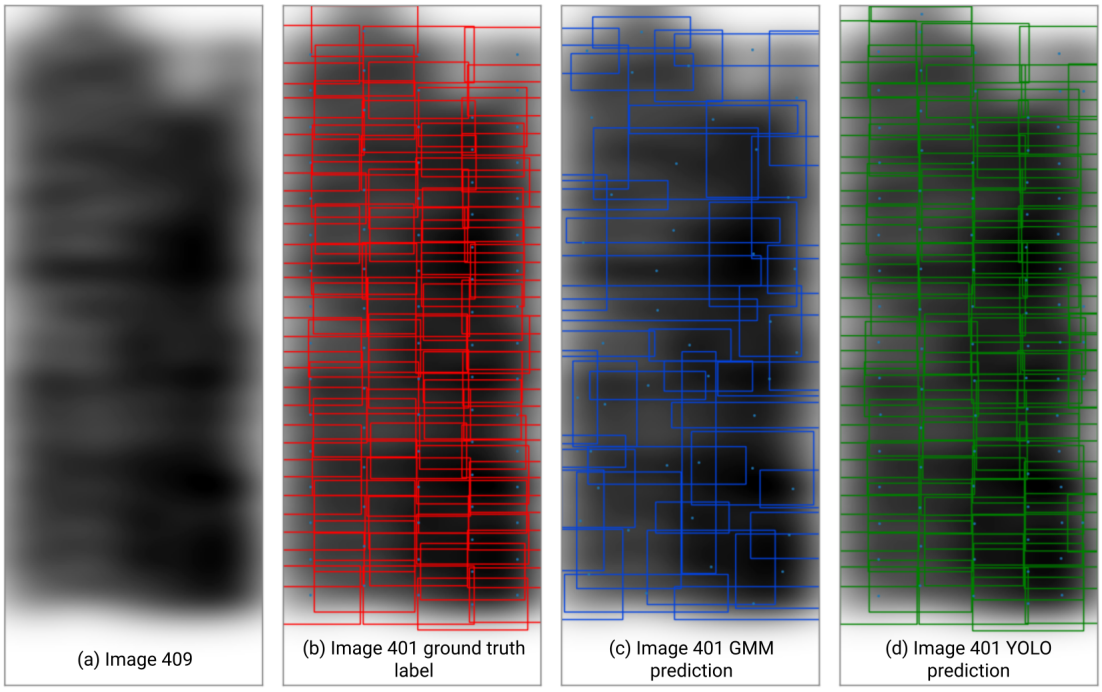


Figure 4.6: Image 401 (DS-IC) detection using GMM and YOLO methods.

In figures 4.6 and 4.7, it is hard to understand the matching Gaussians from the picture. Later, the methods are evaluated regarding error metrics, which is more suitable in this case.

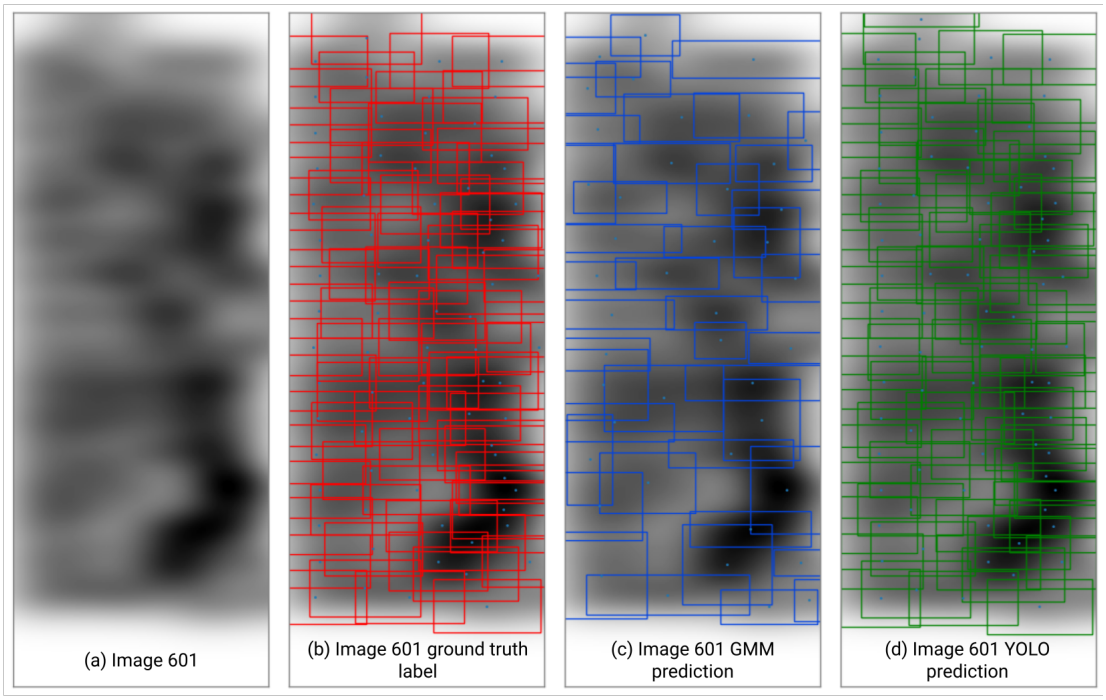


Figure 4.7: Image 601 detection using GMM and YOLO methods.

Nevertheless, it is possible to check that GMM seems to miss entirely or only partially capture some Gaussians, especially in areas with more faded blobs. The method merges many entities and has bounding boxes with a substantial standard deviation in the x-direction.

YOLO's boxes for images in figures 4.6 and 4.7 appear more uniform across the image, and some can be correctly matched back to the original ground truth. YOLO generates many overlapping bounding boxes, suggesting the method doesn't frequently merge the entities as the GMM. YOLO might miss a few Gaussians, but its miss rate is way lower than GMM's. In figure 4.7, there is a clear false positive at the top for a Gaussian in the edge identified by YOLO, a rare circumstance for the method.

In conclusion, YOLO appears to have a more consistent and slightly superior performance than GMM, looking at the figures 4.6 and 4.7. Thankfully, the DS-IC and DS-ID image patterns with spread Gaussians are improbable in SPIDER's operation. It is expected that a human can't detect the Gaussians for such patterns as well, so if this type of image appears, it will be immediately understood that a problem is present in the generation of the beamlets, as the strict control of deflection and divergence is a must requirement for operation on ITER. Nevertheless, these images were tested to stress the object detection models and understand the limiting

	$E_{\#}(\%)$		$E_{C_x}/E_{C_y}(\%)$		$E_{\sigma_x}/E_{\sigma_y}(\%)$		$E_A(\%)$	
Image	GMM	YOLO	GMM	YOLO	GMM	YOLO	GMM	YOLO
109	85.9	94.9	2.2/0.4	0.9 / 0.1	101.1/80.7	2.4/2.0	21.9	22.4
203	94.9	100	2.7/0.6	2.0/0.1	24.9/46.7	5.9/1.7	10.49	12.62
401	54.4	107.6	13.7/2.5	10.6/0.2	38.9/46.7	15.1/3.8	86.2	97.5
601	56.6	101.3	15.9/1.8	10.7/0.3	39.5/41.1	10.4/4.0	99.54	108.4

Table 4.2: GMM and YOLO prediction errors in DS-I for images 109, 203, 401, and 601.

constraints.

Consequently, the later description of the developments of the PX modifier and ensemble in section 3.4 doesn't include the analysis of the sets DS-IC and DS-ID, as the two other ones, DS-IA and DS-IB, are the ones that are closer to the expected in the SPIDER operation.

It is worth mentioning that the trained YOLO model exceeds the speed requirement of prediction, requiring less than one second, a margin significantly lower than the few minutes needed for the experiment [14].

Finally, the error metrics of the prediction for each image in the figures 4.4 to 4.7 for the YOLO and GMM methods are presented in table 4.2.

Comparing the errors in all images, the influence of the Gaussians overlapping in increasing the mean error for standard deviations and amplitude is evident. YOLO is less affected than the GMM method in the standard deviation estimation. Still, as both methods estimate the amplitude similarly, YOLO suffers from high errors in amplitude estimation of the Gaussians as the GMM. It is also clear that the GMM number of entities identified is significantly reduced when the objects overlap, losing nine percentual points from image 209 to 109, while YOLO only loses 5. As GMM merges many Gaussians in the predictions, the reported errors in standard deviation estimations are prohibitive, as presented in 4.2.

The number of Gaussians identified by YOLO and GMM in the groups DS-IC and DS-ID images differs substantially. GMM presents a low detection rate, as many Gaussians are merged in a unique one. YOLO gives many false positives, which results in a greater number of Gaussians identified than the ground truth, i.e., a percentage greater than 100. YOLO is a reasonable model to estimate the dispersions even for images in groups DS-IC and DS-ID, but the amplitude is overestimated, doubling from the ground truth value. This demonstrates the robustness and capacity of the model. Although good when trained with subsets of images, a more general dataset was tested to understand the generalization capabilities of YOLO.

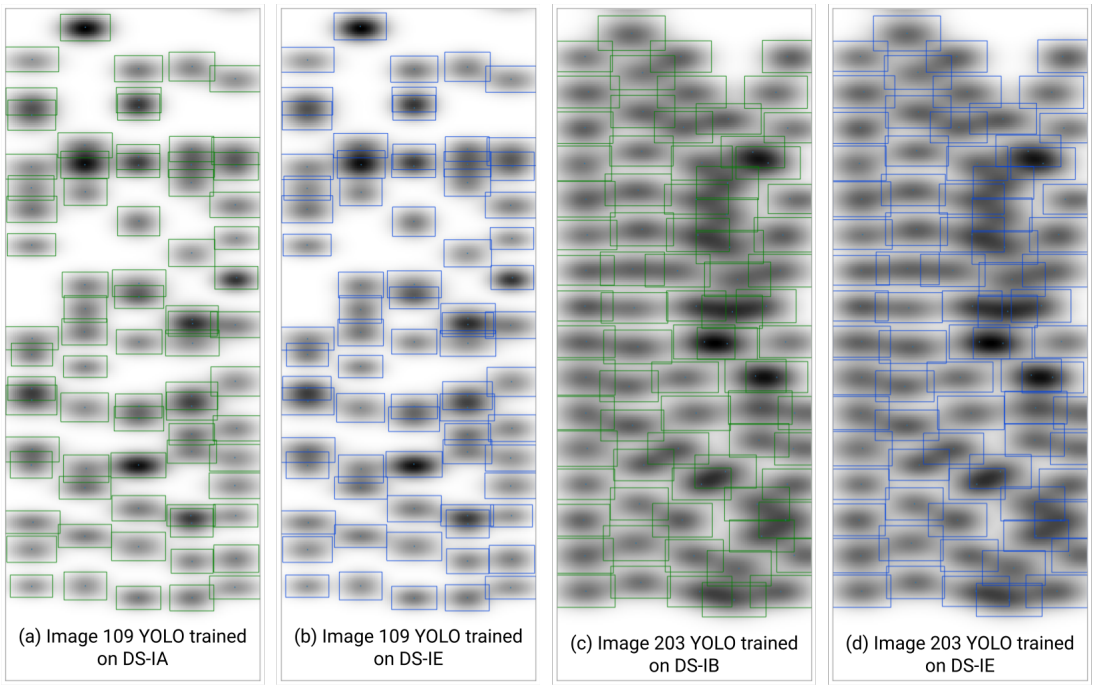


Figure 4.8: Images 109 and 203 detections using YOLO trained using DS-IA, DS-IB and DS-IE.

Extended DS merging DS-IA and DS-IB

The YOLO method was then tested in an extended DS-I group (DS-IE), training the model with images from groups DS-IA and DS-IB. The objective was to evaluate a more generalized model to be applied directly without retraining to the profiles that might arise on STRIKE in the next campaign of SPIDER. As GMM doesn't have a supervised training step, using an extended group would not affect any of the method's performance, so it was not experimented with in this dataset.

The prediction for images 109 and 203 using the model trained with the extended group compared to the models trained with its specific datasets is presented in figure 4.8.

Comparing the predictions in figure 4.8, the model trained in the extended set (DS-IE) has an almost identical prediction to the ones trained on the specific datasets (DS-IA and DS-IB). This indicates the capacity of YOLO for generalization, opening up different possibilities of training, for example, including many images with artificially generated datasets with noise and a variety of Gaussian shapes in the train set, which might not affect the excellent performance of the model in the restricted datasets and could increase the generalization capabilities.

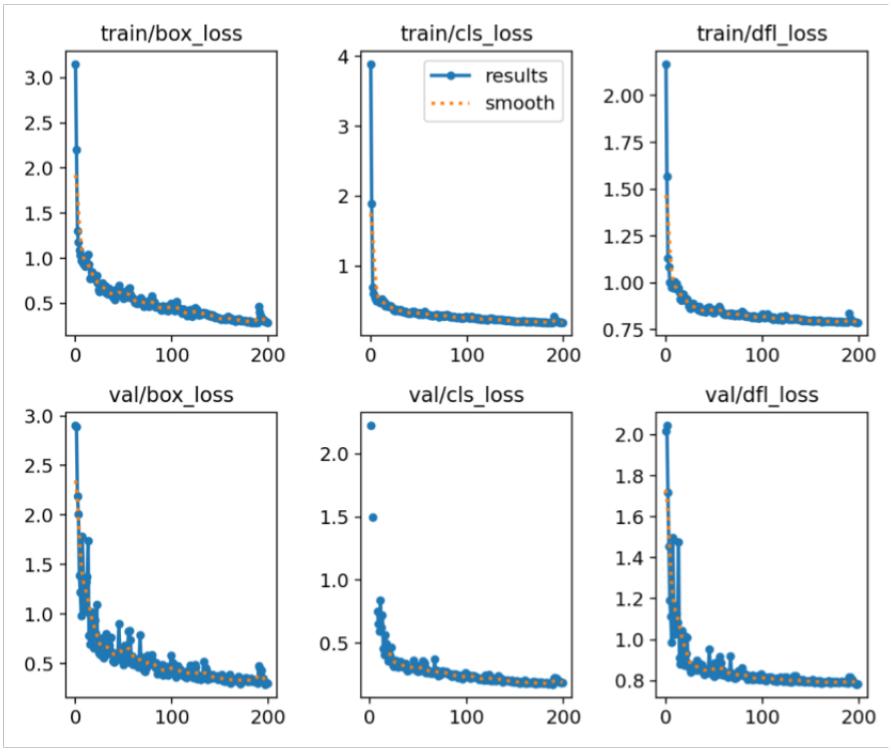


Figure 4.9: Training metrics for YOLO using dataset DS-IE. Figure generated by the library [34].

Training metrics of YOLO for DS-IE

YOLO was trained with more epochs in the extended dataset, 200, to understand the limit of fitting on the model before overfitting. Figure 4.9 presents the three losses of the model for the train and validation dataset during training as the epochs evolved. Figure 4.10 shows the detection performance of the model. The other models trained followed a similar trend in the errors, so only the training metrics of this extended dataset are analyzed in this section. The model's training with the extended dataset, i.e., 360 images in the training set, took 1 hour to complete the 200 epochs utilizing the Tesla T4 GPU with 15GB memory on Google Colab.

Analyzing the figure 4.9, the training losses decrease substantially in the initial epochs and continue afterward, suggesting that the model is learning and improving its performance on the training set, therefore without evident underfitting issues. The validation loss values follow a similar trend as the training loss, which is a positive sign, indicating the model's improved performance on unseen data. There is a clear indication of no overfitting as training and validation losses decrease with training, and the last one doesn't increase in the process. All three losses

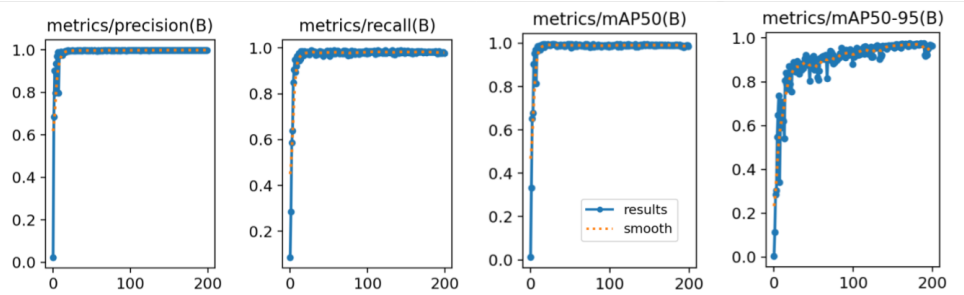


Figure 4.10: Detection metrics for YOLO using dataset DS-IE. Figure generated by the library [34].

decay, so the model improves its performance in the localization of the bounding boxes and the detection of objects.

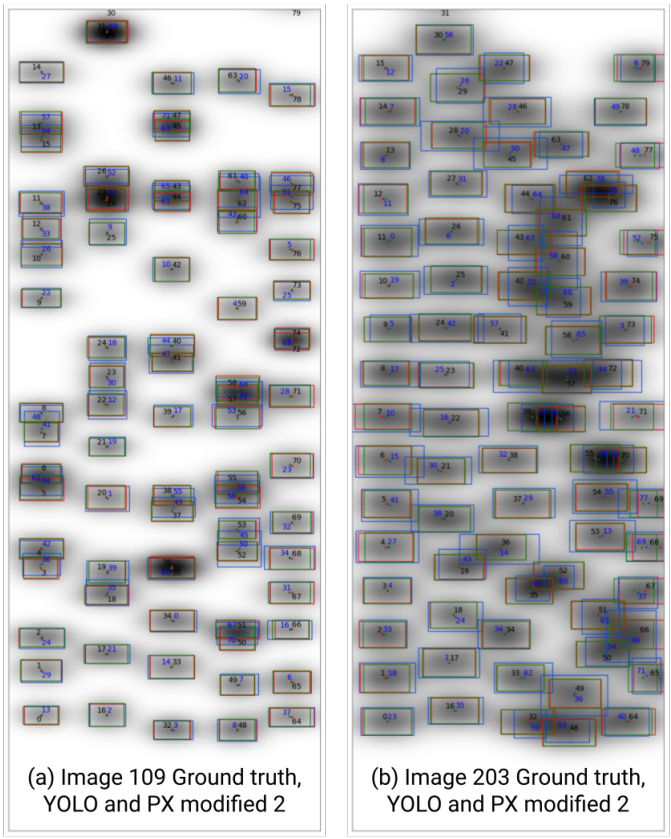
As there is a sharp loss drop in the initial epochs, it might be beneficial in the future, if larger datasets are used, to train the model with early stopping, setting a reasonable threshold for the loss-decreasing. The dataset in this thesis was small, so the early stopping was overruled by setting patience of 100 epochs in the training.

In figure 4.10, the precision, recall, and consequently mAP50 increase with the model's training, indicating that it identifies most objects without many false positives. The robust detection performance metrics are consistent with the observed decreasing losses in 4.9. The mAP50-95, which is the computation of the mean mAP over the range of IoU from 50 to 95, indicates that the model with training can localize the Gaussians in the image with the bounding boxes with high precision.

PX modifier and Ensemble

Attempting to find the local maximum using the refinement method described in 3.4 didn't yield optimal results. Some computed centers were further from the ground truth ones compared to the predictions of YOLO before the refinement. Intersections between Gaussians are likely the cause; the combined effect of one Gaussian over another can distort the place of the peak of both Gaussians in the image.

This observation is supported by analyzing the figure 4.11, which plots the true centers against the estimated and refined ones (ground truth, YOLO, and PX modifier 2). The blue boxes in figure 4.11 are the predictions of PX modifier 2, and the text color blue is the identification of the center of the method. The bounding boxes in red and the numbering in red are the ground truth labels. The bounding box and centers in green are the predictions of YOLO.



(a) Image 109 Ground truth, YOLO and PX modified 2

(b) Image 203 Ground truth, YOLO and PX modified 2

Figure 4.11: Image 109 and 203 prediction YOLO and PX modifier 2.

Analyzing the figure, it is possible to observe that some Gaussians aren't distinctly differentiated as the sum of two separate entities; for example, Gaussians 57 and 58 centers in image 109 in figure 4.11 have a distance of only 4 pixels.

Therefore, from the comparison of image 4.11, it's evident that the Gaussian separation heavily influences the standard deviation estimation using the inverse of the probability density function. The amplitudes fed into the inverse function are taken directly from the image. Consequently, they may exceed the ground truth amplitude of the Gaussians due to their entanglement. Even when Gaussians are distinctly separated, YOLO typically provides a more accurate standard deviation estimation than the method derived from the inverse of the density function.

In conclusion, the predictions from the YOLO method are better without the post-processing using the PX modifier 2. The next hope of improvement in YOLO is that the errors committed in the Gaussians are different from PX modifier 2 and YOLO, which gives the possibility of ensembling to reduce error. This didn't yield better results, as shown in table 4.4.

After careful analysis of the predictions of YOLO and PX modifier 2 combined in the Ensemble, the hypothesis for the inexistence of a significant enhancement using the method is that the error associated with one Gaussian is often intertwined with errors from other Gaussians. Consequently, even if a Gaussian has a reduced $E_{Reconstruction}$ error in one estimation context, this occurs due to the influence of close Gaussians summing up the error in one or the other method, not that the Gaussian per se was better estimated.

The figure 4.12 presents a reconstructed image using the predicted parameters from YOLO, PX modifier, and the Ensembling method. The image reconstructed is the number 140 from the DS-IA.

It is hard to notice any difference between the Ensemble and the YOLO in figure 4.12, as many Gaussians of the Ensemble come from the latter. In the image, a Gaussian that was better estimated in the PX modifier 2 method is highlighted. As it is possible to notice, the PX modifier 2 successfully identified that the Gaussian had a wider standard deviation in the y-direction.

In figure 4.12, the apparent weakness in the intensity of the Gaussians in the reconstructed images compared to ground truth is due to the overestimation of amplitude in overlapping Gaussians. The images were not normalized to the same intensity to highlight the presence of this error.

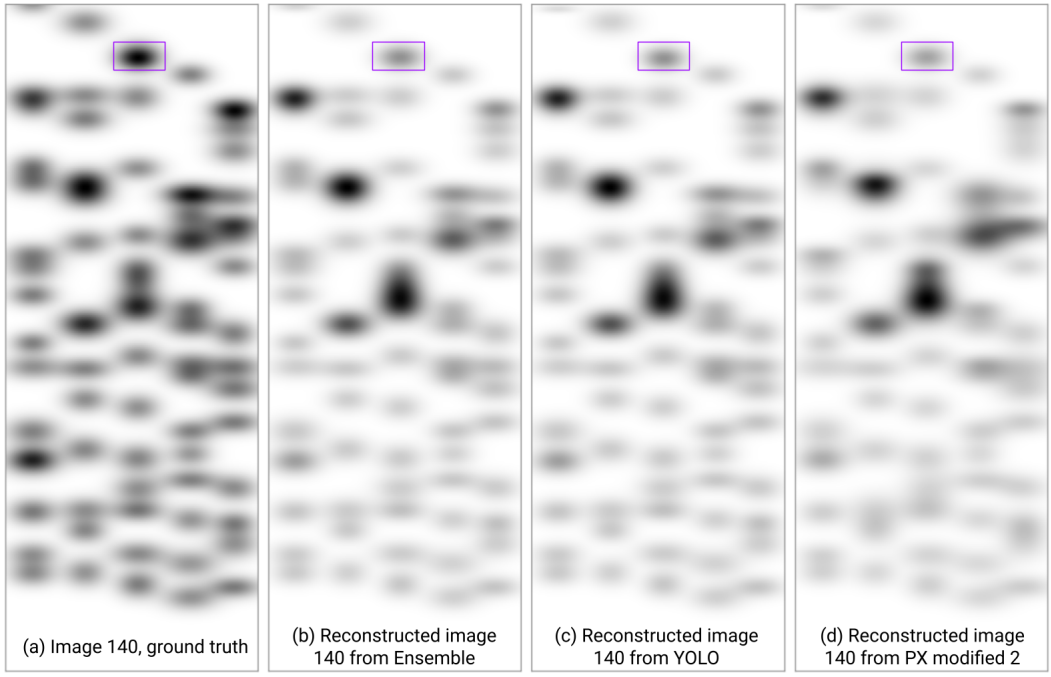


Figure 4.12: Image 140 reconstruction by prediction YOLO DS-IA, PX modifier 2, and Ensemble.

4.2.1 Test error metrics - DS-IA, DS-IB, and DS-IE

The table 4.3 presents the identification of images in each test set used to compute this section's errors. The following error tables show the mean and standard deviation of the errors between these images in the test set.

The table 4.4 and 4.6 presents the errors for the DS-I groups DS-IA and DS-IB, respectively. The table 4.4, and 4.6 presents the amplitude estimation and pixel reconstruction errors. The unit of pixel error reconstruction is the unit of intensity for this dataset.

In the DS-IA errors presented in tables 4.4 and 4.5, it is clear that GMM fails to identify a

Dataset	Test set size	Images
DS-IA	20	18, 53, 55, 70, 79, 94, 99, 109, 112, 113, 121, 129, 131, 140, 148, 176, 184, 186, 193, 194
DS-IB	20	203, 204, 217, 239, 241, 253, 257, 274, 277, 285, 292, 306, 310, 334, 348, 349, 363, 364, 385, 395
DS-IE	40	Previous datasets combined

Table 4.3: Images in the test set in each dataset.

Method	$E_{\#}(\%)$	$E_{C_x}(\%)$	$E_{C_y}(\%)$	$E_{\sigma_x}(\%)$	$E_{\sigma_y}(\%)$
GMM	83.3 ± 3.0	1.6 ± 0.4	0.4 ± 0.1	16.3 ± 2.1	39.1 ± 1.4
YOLO DS-IA	96.0 ± 2.7	1.4 ± 1.3	0.3 ± 0.3	2.8 ± 0.4	3.3 ± 0.8
PX modifier	96.0 ± 2.6	1.0 ± 1.3	0.5 ± 0.3	7.2 ± 0.7	31.7 ± 9.1
PX modifier 2	96.0 ± 2.6	1.0 ± 1.3	0.5 ± 0.3	6.6 ± 0.6	15.2 ± 1.6
Ensembled	96.0 ± 2.6	1.3 ± 1.3	0.3 ± 0.3	2.7 ± 0.4	3.4 ± 0.7

Table 4.4: Localization errors for the test set on the DS-IA for all developed methods.

Method	$E_A(\%)$	E_P
GMM	20.8 ± 3.0	$(3.4 \pm 0.2) \times 10^5$
YOLO DS-IA	26.4 ± 4.8	$(6.3 \pm 1.9) \times 10^5$
PX modifier	29.0 ± 5.0	$(12.2 \pm 2.7) \times 10^5$
PX modifier 2	29.0 ± 5.0	$(9.8 \pm 2.2) \times 10^5$
Ensembled	26.4 ± 4.7	$(6.2 \pm 1.9) \times 10^5$

Table 4.5: Intensity errors for the test set on the DS-IA for all developed methods.

high number of Gaussians, accounting for only an 83% detection rate on average, as it merges them as discussed previously. YOLO has a significantly higher identification rate at 96.0%, showcasing its robustness in object detection. The identification of PX modifier, PX modifier 2, and Ensemble has an identification rate equal to YOLO, as it only adjusts the already identified Gaussians by YOLO from the image.

As expected by the previous discussion, GMM performs poorly in estimating the standard deviations in dataset DS-IA. YOLO outperforms GMM significantly in estimating the standard deviations, with YOLO showing the lowest error values in both directions, even when considering the refinement methods. The Ensemble presented a lower standard deviation than YOLO, as probably some Gaussians could be better estimated using the PX modifier 2, discussed previously in the figure 4.12, but in general, the model didn't improve significantly the results.

It is possible to observe that PX modifier 2 in the dataset DS-IA was able to reduce the error in the x coordinate of the center but increase the error in the y coordinate. This is due to the overlappings of the Gaussians in the y-direction, which causes a false peak by the summation of two Gaussians, misleading the method. This showcases that the PX modifier 2 is not a robust method to be applied, as the overlappings significantly mislead the predictions. The method only works well in conditions of clear Gaussian separation. This can also be concluded by comparing the increased error in the amplitude estimation in table 4.5, which shows that PX

Method	$E_{\#}(\%)$	$E_{C_x}(\%)$	$E_{C_y}(\%)$	$E_{\sigma_x}(\%)$	$E_{\sigma_y}(\%)$
GMM	95.8 ± 2.0	2.6 ± 0.3	0.5 ± 0.2	25.2 ± 0.9	47.6 ± 0.6
YOLO DS-IB	100.5 ± 0.8	2.1 ± 0.2	0.1 ± 0.0	5.7 ± 0.6	1.7 ± 0.5
PX modifier	100.5 ± 0.8	2.4 ± 0.2	0.3 ± 0.0	39.3 ± 12.6	19.9 ± 3.8
PX modifier 2	100.5 ± 0.8	2.4 ± 0.2	0.3 ± 0.0	23.3 ± 1.8	14.5 ± 1.2
Ensembled	100.5 ± 0.8	2.1 ± 0.2	0.1 ± 0.0	7.9 ± 1.4	2.6 ± 0.8

Table 4.6: Localization errors for the test set on the DS-IB for all developed methods.

Method	$E_A(\%)$	E_P
GMM	9.6 ± 1.0	$(5.3 \pm 0.1) \times 10^5$
YOLO DS-IB	11.7 ± 1.3	$(3.4 \pm 0.4) \times 10^5$
PX modifier	12.8 ± 1.5	$(12.2 \pm 2.2) \times 10^5$
PX modifier 2	12.8 ± 1.5	$(8.2 \pm 0.9) \times 10^5$
Ensembled	11.8 ± 1.3	$(3.2 \pm 0.4) \times 10^5$

Table 4.7: Intensity errors for the test set on the DS-IB for all developed methods.

modifier 2 was misleading in updating some centers.

In conclusion, for dataset DS-IA, YOLO outperforms all other methods in terms of robustness and performance of detection and localization. All methods suffer from amplitude estimations, including YOLO.

In DS-IB, it is interesting to highlight that the amplitude estimation errors are lower than in the DS-IA for all methods, demonstrating that the Gaussian overlapping is the main reason for the mistakes in the DS-IA dataset. GMM improved the detection rate substantially, indicating that the method is not robust to overlappings. PX modifier 2 for DS-IB suffers more in estimating the standard deviation in the x-direction compared to the y-direction in the DS-IA. This indicates that the overlapping significantly impacts the method, as already discussed.

YOLO presents a rate of identification higher than 100% for this dataset, which indicates the generation of false positives. Although this could be a problem, this margin is so low it suggests that only in some images did YOLO predict more than 80 images.

In conclusion, for dataset DS-IB, YOLO outperforms all other methods in terms of robustness and performance of detection and localization. All methods suffer from amplitude estimations, including YOLO, but in reduced intensity, as the Gaussians are less overlapped in this dataset than in DS-IA. This problem should be dealt with in future developments, disentangling the Gaussians.

The table 4.8 presents the mean error in the test set for the localization parameters on each

Method	$E_{\#}(\%)$	$E_{C_x}(\%)$	$E_{C_y}(\%)$	$E_{\sigma_x}(\%)$	$E_{\sigma_y}(\%)$
YOLO DS-IE	98.6 ± 2.2	1.7 ± 0.7	0.1 ± 0.0	3.6 ± 1.9	1.6 ± 0.7
PX modifier	98.6 ± 2.2	1.5 ± 1.0	0.4 ± 0.1	23.9 ± 18.4	24.5 ± 9.3
PX modifier 2	98.6 ± 2.2	1.5 ± 1.0	0.4 ± 0.1	15.2 ± 8.7	14.6 ± 2.6
Ensembled	98.6 ± 2.2	1.7 ± 0.7	0.1 ± 0.0	4.7 ± 3.1	2.2 ± 0.6

Table 4.8: Localization errors for the test set on the DS-IE for all developed methods.

Method	$E_A(\%)$	E_P
YOLO DS-IE	17.3 ± 6.3	$(4.3 \pm 1.3) \times 10^5$
PX modifier	19.4 ± 7.2	$(11.5 \pm 2.0) \times 10^5$
PX modifier 2	19.4 ± 7.2	$(8.5 \pm 1.3) \times 10^5$
Ensembled	17.4 ± 6.2	$(4.2 \pm 1.4) \times 10^5$

Table 4.9: Intensity errors for the test set on the DS-IE for all developed methods.

method of YOLO trained using the extended dataset DS-IE. The table 4.8 presents the amplitude estimation and pixel reconstruction errors.

As the errors in 4.9 are computed using the extended test set that combines the test set of groups DS-IA and DS-IB, it is expected that the error would be the mean between the errors of 4.5 and 4.7, if no performance degradation with the generalized dataset on training occurred. The deviation in the errors between images would also be greater, which explains the variance in the mean error reported in the results. As in the previous tables, the ensembled error is the ensemble between the predictions of YOLO and PX modifier 2 predictions.

Indeed, evaluating the tables 4.4, 4.6, and 4.8, the YOLO DS-IE prediction errors for the model trained with the mixed images are at least equal to the average from the errors from YOLO DS-IA and YOLO DS-IA. This shows the capability of YOLO in generalizing the Gaussian forms and the increase in performance with a larger dataset with more training epochs.

In conclusion, the GMM method exhibits significant errors for all datasets when estimating the standard deviation in the y and x directions. This can be attributed to the model fitting Gaussians to a point cloud that only approximates the genuine shapes. Frequent overlaps of Gaussians along the x or y direction could account for the pronounced error in standard deviation estimations, as many Gaussians were merged in bigger blobs. The lower detection percentages for GMM are probably related to this merging, which occurs in identifying the local peaks step, as the posterior fitting is done based on the number of peak identifications passed to the algorithm.

The YOLO method was developed and tested as GMM presented high errors in the stan-

dard deviation estimation and a low Gaussian detection rate. YOLO seems to recognize the expected standard deviation of the Gaussians in the images, allowing overlapping Gaussians to be detected. However, the challenge arises when estimating amplitude in overlapping Gaussians. It is possible to check from the lower mean errors of amplitude for the DS-IB compared to DS-IA that the overlapping is the primary cause of the increase of the error, as group DS-IB has a higher error in the estimation of the center than DS-IA and instead has a lower amplitude error.

YOLO fails to distribute the amplitudes between the Gaussians, as the values were taken directly from the images as the other methods. As a result, the final reconstructed data possesses up to twice the intensity of the original image in some intensely overlapping Gaussians. This discrepancy is due to the Gaussians being reconstructed with the combined amplitude of the final sum. This is the reason all reconstruction errors for all methods in all datasets present a high value.

In conclusion, YOLO appears to outperform GMM, both PX modifier, and ensembling, regarding localization accuracy, handling overlapping Gaussians, and reducing false positives. YOLO has generalization capabilities, indicating increased performance with training using a larger dataset during more epochs. Gaussian entanglement, causing the high amplitude errors, remains the main challenge to be addressed in future developments.

	Method	
Metric	GMM	YOLO
% identified	67.8 ± 2.8	109.3 ± 8.3
$E_{C_x}(\%)$	6.3 ± 0.7	8.4 ± 1.3
$E_{C_y}(\%)$	2.3 ± 0.3	6.2 ± 2.6
$E_{\sigma_x}(\%)$	20.9 ± 1.9	52.8 ± 3.4
$E_{\sigma_y}(\%)$	25.5 ± 3.1	22.6 ± 2.1
$E_A(\%)$	45.6 ± 2.8	70.0 ± 4.1
E_P	4.6 ± 0.1	14.7 ± 3.0

Table 4.10: Table of localization and intensity errors for the test set on the DS-II for GMM and previously trained YOLO DS-IE.

4.3 Results for the second dataset

The second dataset provided, DS-II, is the one that approximates the expected future profile in the STRIKE calorimeter from the SPIDER operation. The previously trained YOLO model with the extended dataset, DS-IE, and the GMM method were evaluated.

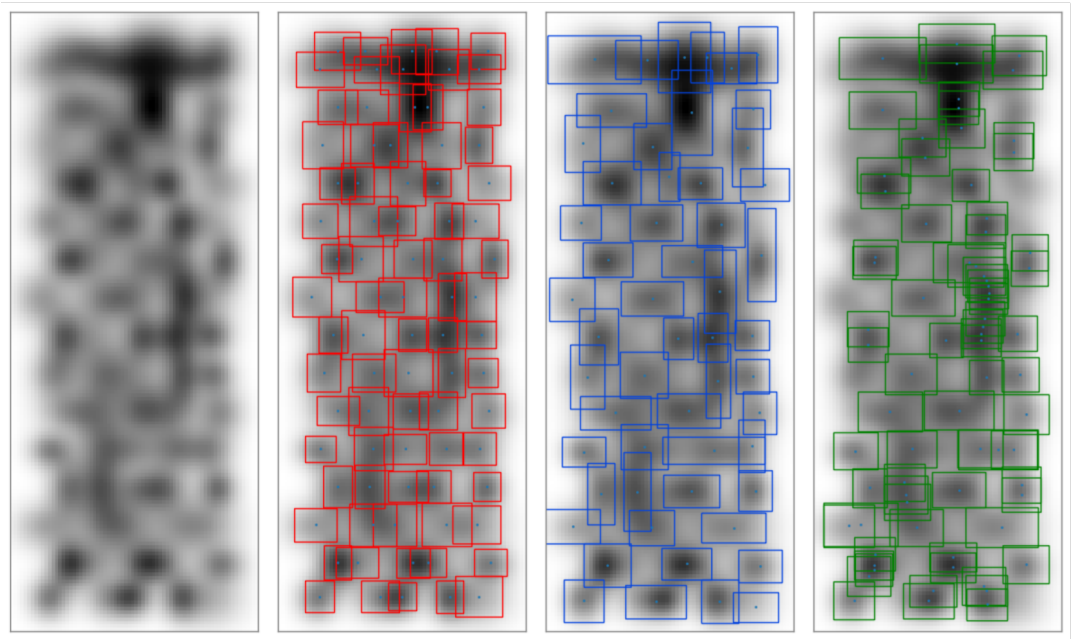
The PX modified 2 and Ensemble were discontinued and not tested in this DS-II as the PX modified method is not as robust as the YOLO, so the extra computational effort to the refinement doesn't make sense as it frequently critically worsened the predictions of standard deviation for many Gaussians.

Due to the lower resolution of the images, the background is imperfectly resolved using the Laplacian transformation. Additionally, the Gaussian regions appear fragmented. Despite these issues, most Gaussians seem to be correctly determined after inspecting the results from image 323 in figure 4.13.

The results of the second dataset (DS-II) for training the YOLO model using DS-IE and the GMM method are presented in table 4.10. Note that the intensity data in the images of DS-II were in terms of temperature, so the E_P for this dataset is not comparable to DS-I.

The errors for GMM appear lower than YOLO sometimes because the comparison of errors is based on matching one ground truth Gaussian and a predicted one. Therefore, Gaussians from ground truth that are not matched weren't in the error computations. Hence, when the GMM method has a significantly low percentage of identification of Gaussians, its error is not comparable to YOLO because of this effect.

It is possible to notice from figure 4.13 that the GMM method can identify the circular Gaussians successfully but suffer from the overlapping ones as in the DS-I, merging many. YOLO



(a) Image 323 (b) Image 323 ground truth label (c) Image 323 GMM prediction (d) Image 323 YOLO prediction

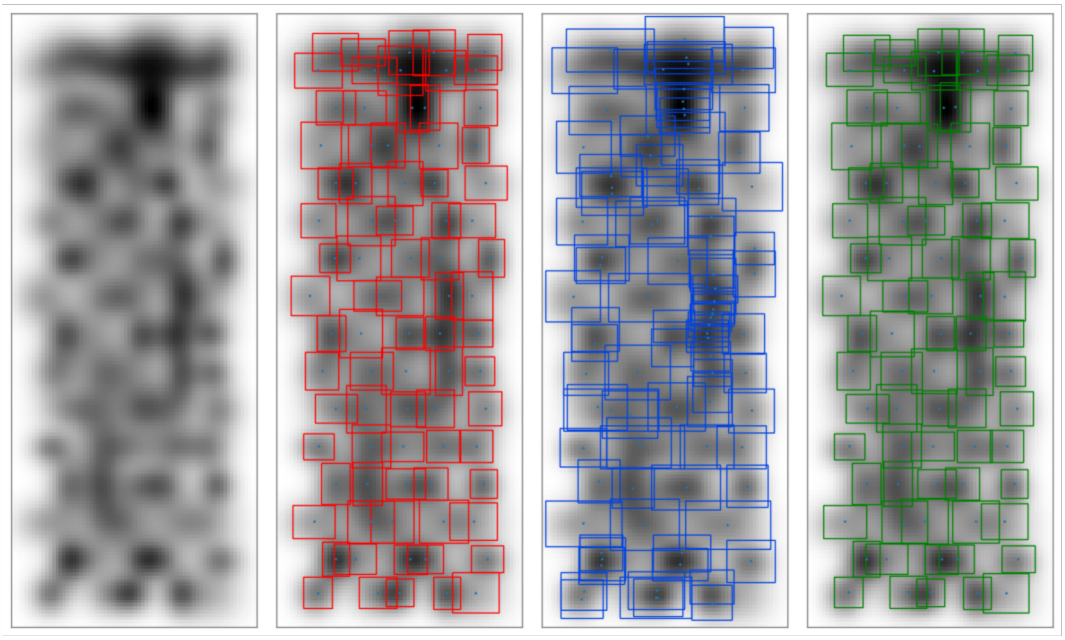
Figure 4.13: Image 323 (DS-II) detection using GMM and YOLO DS-IE methods.

trained only with Gaussians from DS-I that are flattened in the x-direction can't predict the circular formats, outputting many false positives of flattened Gaussians that cover one Gaussian that is not flattened in the image. Despite this problem, the YOLO model trained in images of DS-IE could identify some Gaussians, regardless of the lower resolution in DS-II images. This demonstrates the ability to generalize the model in different conditions.

A new training of a YOLO model was performed as the previous model's prediction performance was not good, and the clear misprediction for circular Gaussians was vastly present in this new dataset. The results of the second dataset for this further training of the YOLO model are presented in table 4.11, and the comparison of the prediction of image 323 is shown in 4.14.

Analyzing the table 4.11, YOLO trained on the DS-II didn't miss or misfire any Gaussian in the images on the test dataset, a compelling result for the identification of the 80 beamlets hitting STRIKE. The mean errors in the standard deviations were lower than 2%, a remark for the precision needed in the characterization for ITER. The maximum standard deviation error in the x-direction for one Gaussian occurred in image 188, equal to 13.9%. In the y-direction, this error is 20.6% on image 323.

These maximum errors in standard deviation should be reduced in the future to guarantee

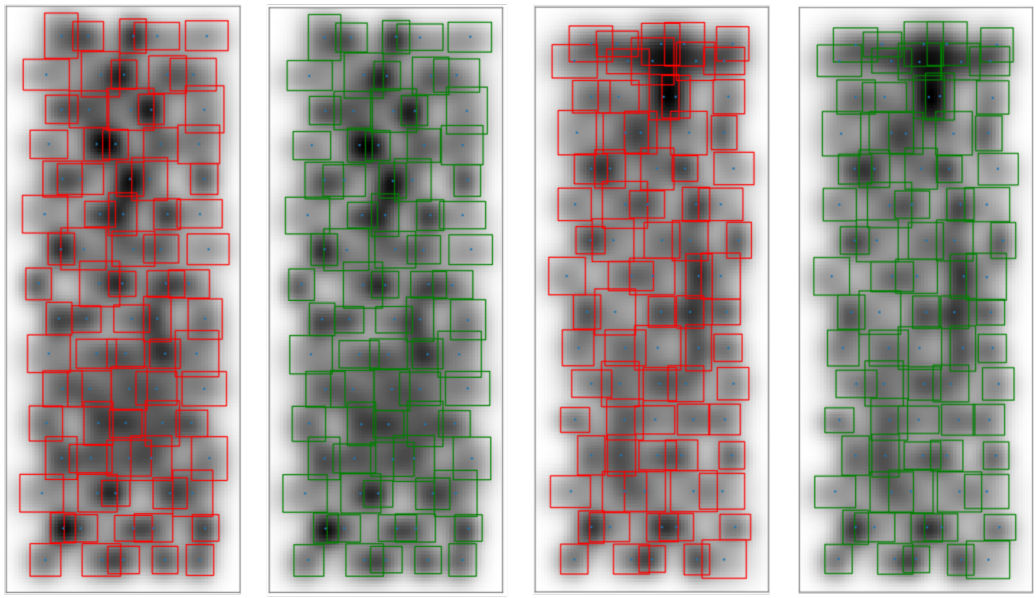


(a) Image 323 (b) Image 323 ground truth label (c) Image 323 YOLO DS-IE prediction (d) Image 323 YOLO DS-II prediction

Figure 4.14: Image 323 (DS-II) detection using YOLO DS-IE and YOLO DS-II methods.

Metric	Value
$E_{\#}(\%)$	100 ± 0.0
$E_{C_x}(\%)$	0.4 ± 0.1
$E_{C_y}(\%)$	0.1 ± 0.0
$E_{\sigma_x}(\%)$	1.7 ± 0.2
$E_{\sigma_y}(\%)$	1.7 ± 0.2
$E_A(\%)$	52.2 ± 2.8
E_P	3.1 ± 0.2

Table 4.11: Table of localization and intensity errors for the test set on the DS-II for the trained YOLO DS-II.



(a) Image 109 labelled

(b) Image 109 YOLO trained on DS-II

(c) Image 323 labelled

(d) Image 323 YOLO trained on DS-II

Figure 4.15: YOLO DS-II best and worst prediction in DS-II.

an error lower than 10% for all beamlets, a precision recommended for ITER [2]. On average, the method exceeds five times the minimum requirement in both directions. Hence, it demonstrates that YOLO can be used in the future characterization of STRIKE in the next SPIDER campaign with the remaining challenge of disentangling the Gaussian for a better amplitude estimate.

Figure 4.15 presents the worst (image 323) and best (image 109) prediction of the YOLO model considering the error metric of the standard deviation of the Gaussians. The mean standard deviation error for image 323 is 1.73% and 2.44% for x and y-directions, respectively. The maximum standard deviation error for a Gaussian in the image is 20.63% in the y-direction. The mean standard deviation error for image 109 is 1.46% and 1.3% for x and y-directions, respectively. The maximum standard deviation error for a Gaussian in the image is 6.89% in the x-direction.

It is possible to observe on 4.15 that even for the worst prediction, the Gaussians' parameters predicted by YOLO are extremely close to the ground truth ones. This indicates that the errors are bounded in the model, and the model presents a good performance even in the image with maximum errors.

4.3.1 Remarks and further developments

It's crucial to note that the dataset created for this thesis assumes a specific timeframe for extracting images from the video of the IR cameras. As the experiment advances and the beamlets hit the tile, the Gaussians expand by the heat transfer. Eventually, they may merge completely, making distinguishing between the beamlets impinging the calorimeter impossible.

Therefore, extracting the images in the true experiment will require adjustments, and the model might benefit from training using various Gaussian sizes. For instance, frames can be taken from 1s to 5s after the beamlets initiate. Another limitation is that, due to variations in SPIDER's operational conditions, the Gaussian sizes may differ significantly at the same video timestamp across different runs. Developing a model tailored for streaming video may offer a more generalized solution for varying SPIDER conditions.

When Gaussians overlap, their boundaries become indistinct, leading all fitting methods to occasionally misallocate points or combine overlapping Gaussians. This can result in several Gaussians being perceived as one peak, concealing their amplitude and the actual number of Gaussians. Such overlap complicates the distinction between individual components in a Gaussian mixture. A clear separation between Gaussians is preferable to identify their amplitude and centers accurately.

High amplitude errors present in all methods were related to these overlappings. The amplitude of each Gaussian was considered to be the direct intensity value at its center. In an ideal scenario, this value would represent the peak intensity. However, due to inaccuracies in predicting the center, it might not always coincide with the actual peak. The overlap between Gaussians exacerbates this misalignment, as one Gaussian intensity sums to another one. Thus, the amplitude estimates often deviate significantly from the true peaks, primarily due to the influence of intersecting Gaussians and off-center predictions.

Future work to disentangle Gaussians is strongly advisable if required by the SPIDER experimental images. This can be accomplished by testing the YOLO model with a high IoU in Gaussians with outlier amplitudes, which can signal an overlapping issue. Another possible path is to employ a further curve fitting varying amplitudes and maintaining the other parameters fixed from the results of YOLO.

It's vital to note that experimental images from STRIKE may exhibit noise not present in the synthetic training data. If the YOLO model underperforms on these experimental images, retraining with synthetic data that includes noise is advised if labeled experimental data is not available. The noise within the images could inviabilize the initial GMM method, which em-

ploys Laplacian filtering, which is highly sensitive to noisy data. Should this occur, testing a direct image prediction without Laplacian filtering might be a viable alternative for the GMM method. There is also the option to try noise removal techniques as a preprocessing step on the images.

The GMM method can identify Gaussians for data labeling since it employs unsupervised learning. Labeling steps can be more time-consuming than prediction, so the algorithm's computational time might be sufficient. After using GMM, a manual refinement is advisable. Labeling may pose challenges since bounding boxes are three times the standard deviation. However, utilizing dedicated labeling software can simplify the process. This manually refined labeled data is essential for training the YOLO model with experimental data. A combination with the synthetically generated dataset might also be a good exploration of the problem.

Even if the future beamlets drift from a Gaussian-like profile, YOLO learns features directly from the data through training, making it potentially more adaptable to various data distributions and complexities. In contrast, GMM relies on assumptions about the data Gaussian distribution and might struggle if the data doesn't align with those assumptions.

The prediction times of YOLO, generally less than 1 second, were remarkably faster than the GMM ones, typically almost 1 minute, both times in a CPU on Google Colab. This facilitates YOLO to predict stream video if necessary in the future, optimizing the model to run on prediction in a GPU. Another advantage of YOLO is that the model is typically more consistent, as it doesn't rely on iterative refinement based on starting point conditions as GMM does.

Finally, if rotated Gaussians appear in the experimental data and the angulation needs to be determined, the training of a YOLO segmentation model is suggested, as it would classify each pixel in the image and the restriction of bounding boxes that are parallel to the axis is surpassed. A modified model using rotated bounding boxes might also be trained; for example, YOLOv5-obb [39].

5
Conclusion

The object detection model of the 80 beamlets in the STRIKE tile was developed successfully, attending to the required detection performance and exceeding the computational time needed for prediction. The successful model was based on the YOLO architecture, and a pre-trained model was retrained using a dataset of artificially generated images. A second method that was developed didn't reach the same performance nor the computational time prediction required for the task. It was based on the GMM algorithm and an initial peak estimation using a local maximum identification function.

The YOLO model demonstrated robust performance in detecting and localizing individual 2D Gaussians, as demonstrated in the test images of DS-II, with error rates below 2% for predicting Gaussian centers and standard deviations. Specifically, the mean error for centers was 0.4% in the x-direction and 0.1% in the y-direction. The mean error reached 1.7% for standard deviations in both x and y-directions. YOLO achieved exceptional accuracy in detecting the 2D Gaussian shapes in the test images, successfully identifying all beamlets. However, amplitude estimation was challenging due to overlapping Gaussians, with a mean error of 52.2% for this estimation. Its prediction time to detect the beamlets in an image was under 1 second, efficiently meeting the speed requirements.

Despite the bad performance of the second method, as it was an unsupervised learning algorithm, it can be helpful as a labeling method in the future images of STRIKE. It exhibited significantly high errors of 6.3% and 2.3% for x and y centers, respectively, and 20.9% and 25.5% for standard deviation in the x and y direction, respectively, in the test images of DS-II. Its Gaus-

sian detection rate was also low, averaging 67.8%, and presented a high amplitude mean error of 45.6%.

The methods developed to refine the results of YOLO, PX modifier, PX modifier 2, and Ensemble didn't achieve their objective. Although PX modifier/PX modifier 2 improved the centers' localization, the standard deviation computation was tremendously affected by the overlapping Gaussians, yielding bad results for the methods. The Ensemble method did not successfully use the results of different models to improve the detections.

All methods failed to estimate the amplitude correctly due to challenges with overlapping Gaussians. Still, a further curve fitting varying amplitudes and maintaining the other parameters fixed can be pursued as future work to resolve the problem. The author expects that YOLO trained in this thesis and future refinements of the model will help understand SPIDER results and its optimization to advance the technologies required for nuclear fusion.

References

[1] I. Organization. (2023) What is iter? Accessed: 2023-09-08. [Online]. Available: https://www.iter.org/proj/inafewlines

[2] G. Canocchi, "First characterization of the spider negative ion beam by the diagnostic calorimeter strike," Bachelor's thesis, University of Padova, 9 2019. [Online]. Available: http://hdl.handle.net/20.500.12608/24289

[3] I. Organization. (2023) What is a tokamak? Accessed: 2023-09-08. [Online]. Available: https://www.iter.org/proj/inafewlines

[4] ——. (2023) Tokamak. Accessed: 2023-09-08. [Online]. Available: https://www.iter.org/mach/tokamak

[5] Creating a small sun on earth: Spider - the most powerful negative ion beam source in the world. Accessed on 2023-09-10. [Online]. Available: https://www.iter.org/doc/www/content/com/Lists/WebText_2014/Attachments/399/SPIDER_BROCHURE_WEB.pdf

[6] Neutral beam injection based on negative ions. Accessed on 2023-09-10. [Online]. Available: https://www.ipp.mpg.de/3704093/nnbi

[7] Principle of neutral beam injection. Accessed on 2023-09-10. [Online]. Available: https://www.ipp.mpg.de/3704059/nbi_principle

[8] C. RFX. (2023) Mitica: The neutral particle injector of 1 mv for iter. Accessed: 2023-09-08. [Online]. Available: https://www.igi.cnr.it/en/research/negative-ion-neutral-beam-injection/mitica/

[9] ——. (2023) The nbtf project – neutral beam test facility. Accessed: 2023-09-08. [Online]. Available: https://www.igi.cnr.it/en/research/negative-ion-neutral-beam-injection/nbtf-project/

[10] I. Organization. (2023) Neutral beam test facility (nbtf). Accessed: 2023-09-08. [Online]. Available: https://www.iter.org/construction/NBTF

[11] C. RFX. (2023) Consorzio rfx. Accessed: 2023-09-08. [Online]. Available: https://www.igi.cnr.it/en/about-us/

[12] ——. (2023) Spider: The world's most powerful negative ion source. Accessed: 2023-09-08. [Online]. Available: https://www.igi.cnr.it/en/research/negative-ion-neutral-beam-injection/spider/

[13] ——. (2023) The spider components. Accessed: 2023-09-08. [Online]. Available: https://www.igi.cnr.it/en/research/negative-ion-neutral-beam-injection/spider/components-of-spider/

[14] R. S. Delogu, A. Montisci, A. Pimazzoni, G. Serianni, and G. Sias, "Neural network based prediction of heat flux profiles on strike," *Fusion Engineering and Design*, vol. 146, pp. 2307–2313, 2019, sI:SOFT-30. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920379619305113

[15] C. RFX. (2023) Strike diagnostic system. Accessed: 2023-09-08. [Online]. Available: https://www.igi.cnr.it/en/news/strike-diagnostic-system/

[16] R. S. Delogu, A. Montisci, A. Pimazzoni, M. Savarese, G. Serianni, and G. Sias, "Strike heat flux reconstruction by using neural networks: Application to the experimental results," *IEEE Transactions on Plasma Science*, vol. 50, no. 11, pp. 3935–3940, 2022.

[17] A. Pimazzoni, "Investigation of the parameters of a particle beam by numerical models and diagnostic calorimetry," Doctoral Thesis, Università degli studi di Padova, 1 2018. [Online]. Available: http://hdl.handle.net/11577/3424561

[18] R. S. Delogu, C. Poggi, A. Pimazzoni, G. Rossi, and G. Serianni, "Analysis of diagnostic calorimeter data by the transfer function technique," *Review of Scientific Instruments*, vol. 87, no. 2, p. 02B932, 11 2015. [Online]. Available: https://doi.org/10.1063/1.4936081

[19] A. Steffinlongo, "Evaluation of negative ion beam heat flux profile on strike by neural networks," Bachelor's thesis, University of Padova, 11 2019. [Online]. Available: http://hdl.handle.net/20.500.12608/22642

[20] N. Lonigro, "Strike beamlet parameters retrieving via convolutional neural network," Bachelor's thesis, University of Padova, 6 2019. [Online]. Available: http://hdl.handle.net/20.500.12608/23935

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[22] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

[23] Scikit-learn, "Gaussian mixture models," accessed: 2023-10-17. [Online]. Available: https://scikit-learn.org/stable/modules/mixture.html#gmm

[24] A. Ng, "Cs229 lecture notes," 2022, updated by Tengyu Ma.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[26] A. Ng, K. Katanforoosh, and Y. B. Mourri, "Convolutional neural networks," https://www.coursera.org/learn/convolutional-neural-networks/home/info, Coursera, 2023, in the fourth course of the Deep Learning Specialization. Available on Coursera.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, 12 1998.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[30] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2014.

[31] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016.

[32] RangeKing *et al.*, "Brief summary of yolov8 model structure," GitHub discussion thread, 2023, opened on Jan 10; 220 comments; Accessed: 2023-10-17. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189

[33] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[34] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, K. Michael, TaoXie, J. Fang, imyhxy, Lorna, Z. Yifu, C. Wong, A. V, D. Montes, Z. Wang, C. Fati, J. Nadar, Laughing, and . M. Jain, "ultralytics/yolov5: v7.0 - yolov5 sota realtime instance segmentation (v7.0)," 2022. [Online]. Available: https://github.com/ultralytics/ultralytics

[35] "Ultralytics models documentation," https://docs.ultralytics.com/models/, accessed: 2023-10-16.

[36] LordSyd *et al.*, "Are class and box losses calculated the same in yolov8 and yolov5?" GitHub discussion thread, 2023, opened on May 23; 17 comments; Accessed: 2023-10-17. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/2789

[37] J. Solawetz, "What is mean average precision (map) in object detection?" 5 2020, accessed: 2023-10-17. [Online]. Available: https://blog.roboflow.com/mean-average-precision/

[38] K. Urazaki Junior, "STRIKE_YOLO," https://github.com/kjrurazaki/STRIKE_YOLO.git, 2023.

[39] "yolov5_obb: yolov5 + csl_label. (oriented object detection) (rotation detection) (rotated bbox)," https://github.com/hukaixuan19970627/yolov5_obb, 2022, gitHub repository.

Acknowledgments

I want to thank everyone who helped me accomplish this master's. Especially my parents, Kenji and Silvia, my sister, Mariana, and my companion, Mini, and her family, for always supporting me in pursuing my wishes and dreams. I am truly lucky to have you in my life.

I am also immensely grateful to Consorzio RFX and Dr. Gianluigi Serianni for allowing me to develop this thesis in the group. I would especially like to thank my supervisors, Dr. Rita Delogu and Dr. Nicolo' Marconato, for always being available to provide valuable guidance. I also want to thank Dr. Andrea Rigoni and Dr. Antonio Pimazzoni, with whom I briefly discussed my work but provided me with many important insights.

I have had a great time at UNIPD these years, and I thank all my colleagues, Professors, and University staff. I leave this master more hungry for knowledge than when I arrived. Part of it was Professors who were genuinely passionate about their subject, which inspired me.

Lastly, I am thankful for having the opportunity to live in Italy, especially in the Regione di Veneto, which also provided the financial means for my studies. Living and learning everything in this beautiful country was an incredible experience.