UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

# Deep Learning Based Automatic Segmentation of Gas Flares in Single Beam Echo Sounder Data

Teodor Lynghaug Skotnes

STA-3941 Master's thesis in applied physics and mathematics - December 2023

# Abstract

This thesis introduces the first study of instance segmentation applied to gas flares in single beam echo sounder data. We develop a comprehensive dataset consisting of 1,414 images, featuring 5,142 segmented objects identified as gas flare. A key contribution is the adaptation of the Brier score specifically for instance segmentation. Further, we show how to the Weighted Box Fusion (WBF) algorithm for instance segmentation. Using the newly developed Brier metric for instance segmentation, as well as the mAP metric, we show that our ensemble models fused with WBF are quantitatively as good as the average human expert. However, our qualitative analysis identifies critical areas where these models fall short, indicating the need for further refinement to reach human-level performance. The thesis concludes by proposing potential improvements and future research directions. We remark that if implemented, these could bridge the gap between human and machine-level performance.

# Acknowledgements

First of all, I want to thank my main advisors, Eirik and Knut Ola, for their helpful discussions and dedication in reviewing my thesis. I extend my thanks to Fred, Miguel, and Phuong for advising me on my project paper on reinforcement learning a year ago, as well as for agreeing once again to advise me on this thesis project. Additionally, I thank the experts Marie, Stetzler, Knut Ola, Kaya, Pavel, Manuel, and Benedicte for providing data.

Finally, I want to thank my close family for their never ending love and support.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| AP | Average Precision |
| AT | Anchor Template |
| BCE | Binary Cross-Entropy |
| CIoU | Complete Intersection over Union |
| CNN | Convolutional Neural Network |
| COCO | Common Objects in Context |
| DETR | DEtection TRansformer |
| EMA | Exponential Moving Average |
| FCN | Fully Convolutional Network |
| FCNN | fully Connected Neural Network |
| FN | False Negatives |
| FP | False Positive |
| FPN | Feature Pyramid Network |
| GAN | Generative Adversarial Network |
| GPU | Graphics Processing Unit |
| HSV | Hue, Saturation, and Value |

| | |
|---|---|
| IoU | Intersection over Union |
| KL | Kullback–Leibler |
| mAP | Mean Average Precision |
| ML | Machine Learning |
| MSE | Maximum Likelihood Estimator |
| MSE | Mean Square Error |
| NaN | Not a Number |
| NLL | Negative Log-Likelihood |
| NMS | Non-Maximum Suppression |
| P | Precision |
| PANet | Path Aggregation Network |
| R | Recall |
| RoI | Regions of Interest |
| RPN | Region Proposal Network |
| SGD | Stochastic Gradient Descent |
| SPP | Spatial Pyramid Pooling |
| SSD | Single Shot MultiBox Detector |
| SVM | Support Vector Machine |
| TP | True Positive |
| WBF | Weighted Box Fusion |
| YOLACT | You Only Look at Coefficients |
| YOLO | You Only Look Once |

# 1 Introduction

## 1.1 Context

Accurately monitoring greenhouse gas emissions is essential for evaluating both current and future climate scenarios, as well as understanding their potential impact on the local environment. Measuring emissions originating from the seabed, being natural or leakage from human installations is, however, a considerable challenge. These emissions can have a significant impact not only the atmospheric carbon budget but also on local biology. Challenges in constraining these emissions is largely due to the scarcity of comprehensive and geographically diverse data, making the ocean notoriously under-sampled compared to the atmosphere. These limitations emphasize the need for innovations that improve and expedite the collection of ocean data that can constrain these emissions [14, 18].

The main greenhouse gases include carbon dioxide ($CO_2$), methane ($CH_4$) and nitrous oxide ($N_2O$). The significance of these gases lies in their ability to absorb and emit radiation within the thermal infrared range. Different greenhouse gases absorb different wavelengths of infrared radiation. After absorbing the radiation, they emit it in various directions, with a significant proportion being redirected back towards the Earth, causing warming of the planet [54].

The greenhouse gas methane is considered to be 32 times more potent than carbon dioxide, and through natural as well as anthropogenic origins, it is thought to contribute to the global greenhouse effect by 16%. Methane lasts,

on average, a decade in the atmosphere, and although it disperses quicker than carbon dioxide, its potency makes it a key factor in global warming [18]. Current estimates of methane emissions and the global methane budget resulting from marine gas seepages are highly uncertain due to the scarcity of high quality ocean data [14].

## 1.2 Problem Statement and Objective

Seabed gas seepage can be observed through hydroacoustic surveys, either using singlebeam or multibeam echo sounders [78]. Echo sounders operate by emitting an acoustic signal at a specific frequency directly downward into the water. When this signal encounters the seafloor or any object in the water column, it reflects back to the surface where the echo sounder is located, giving us an echogram. Multibeam echo sounders can capture extensive 3-dimensional hydroacoustic datasets, however, their economic burden and increased payload requirements make singlebeam data more commonplace and easy to obtain. Figure 1.1 illustrates a singlebeam echogram with observed gas seepage, fish, reverberations from small organisms, and other noise disturbances. Gas seepages from the seafloor as observed in hydroacoustic data are often called gas flares[1] or simply flares. This terminology will be adopted from here on out.

An important part of the pipeline in studying gas seepage in hydroacoustic data is the actual detection and segmentation of gas seepages. The key objective here is to accurately identify flares with high probability and discern which flares are useful for further gas flow analysis. In single beam echo sounder data this is of particular importance, as it is possible to quantitatively estimate the seabed gas flow by relating acoustic backscatter target strength to gas volume in the water column [78]. In this context, a useful flare is defined as one that is connected to the seafloor and exhibits strong target strength in the echogram. Figure 1.2 shows an example of a gas flare with strong target strength, indicated by the more yellowish-red color on the left, and a flare that is disconnected from the seafloor, thus deemed not useful, on the right. Note that it is important to map out both useful and non-useful flares, but only the useful flares are of interest for segmentation as these are processed for further analysis by researchers. The segmentation process of gas flares consists of *identifying* gas flares in the echograms and then *segmenting* the flare by drawing a precise polygon over the gas flare. This process can present a significant workload, and is typically done by highly skilled researchers, whose time could be better spent elsewhere.

1. The shape of gas seepages on the echograms often distinctly resembles the flames produced by gas flaring, hence the name.

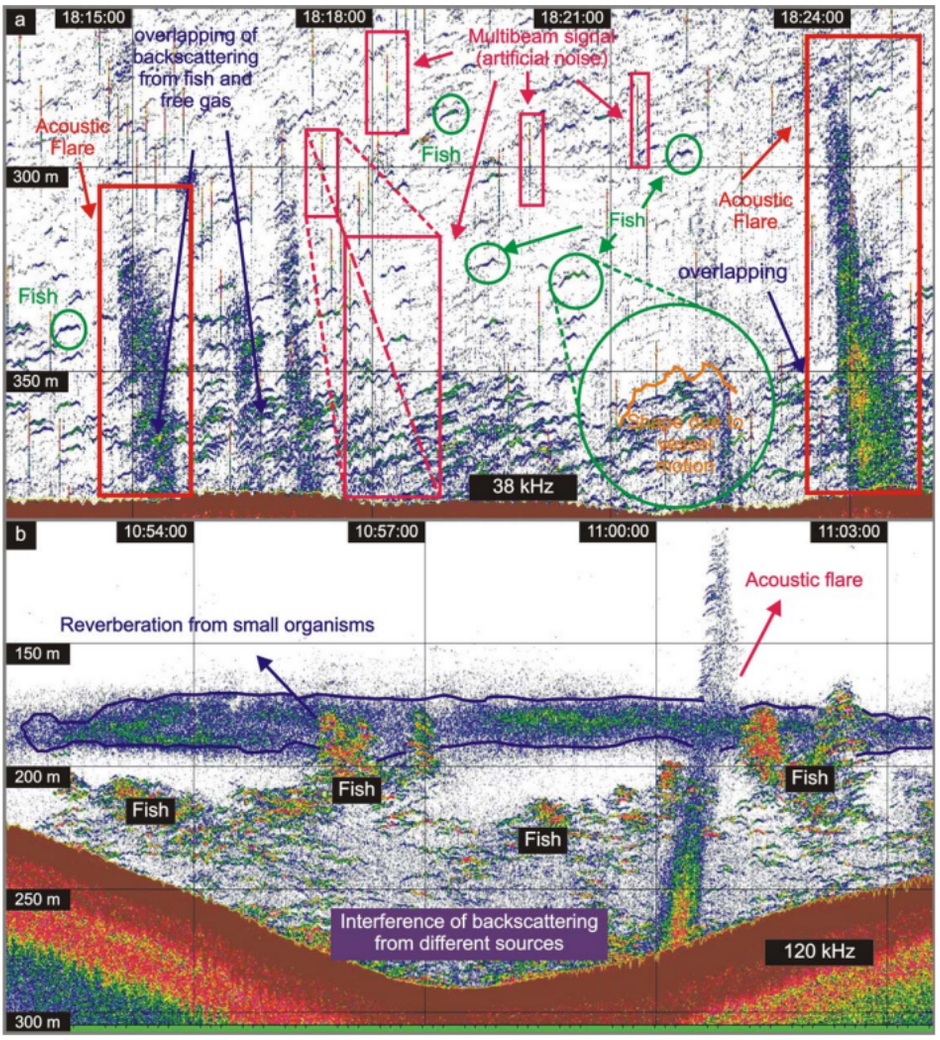



**Figure 1.1:** Image and caption reproduced from [78]: (a) Echograms (38 kHz) showing flares as manifestation of rising bubbles and sources of noise (multibeam) and reverberation (fish). This complicates the identification of free gas fluxes as the interference of the different signal sources results in wrong backscattering values for bubbles, which again may result in flux overestimations. The image shows the effect of the vessel motion on the acoustic data, that is, the shape of backscatter signals of fish (wobbly shape). (b) Echogram (120 kHz) showing the interference of hydroacoustic signals from bubbles (flares) and sources of reverberation (fish and plankton). Here, the plankton layer shows more distinctly because of the higher frequency used.

In addition to being a significant time sink, the detection and segmentation of flares can be subject to personal interpretation. Often, it is not clear where one gas flare ends and another begins. Furthermore, there can be ambiguity regard-

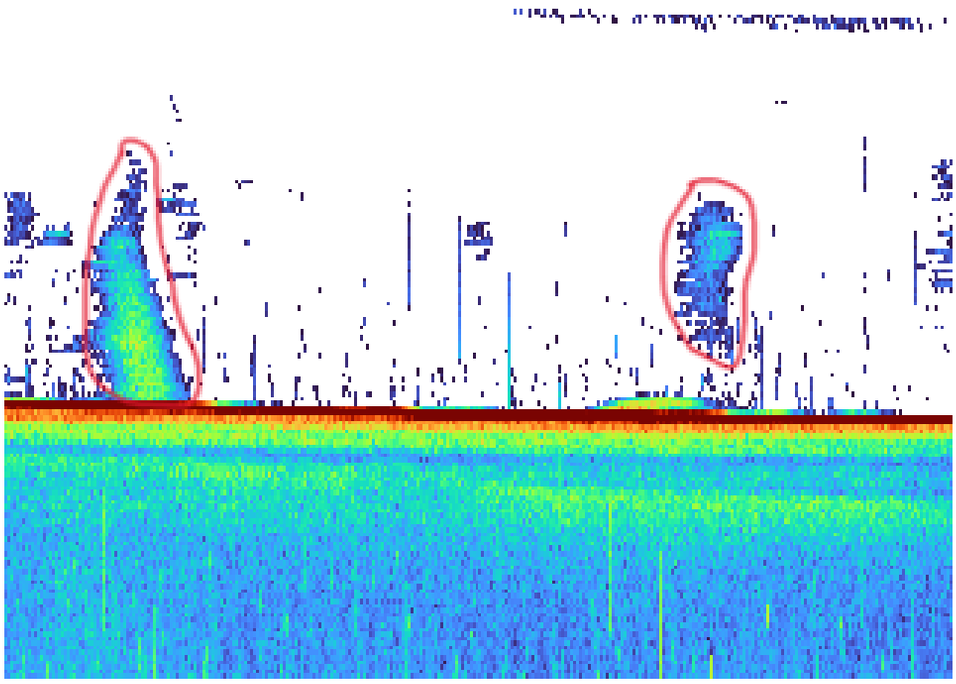


**Figure 1.2:** Flares encircled in red. On the left is a flare deemed useful because of its root to the seafloor and strong signal. On the right is an object unlikely to be a flare, but is included to illustrate how a flare deemed not useful might look.

ing whether what is observed is actually a gas flare and not an amalgamation of different objects like marine life and noise, which together might mimic a flare's appearance. Disagreement between experts can generate considerable noise in flare datasets, as shown in [18] and herein.

It is of interest to fully automate flare identification and segmentation at or beyond human-level accuracy, so as to relieve humans of this task. Additionally, full automation would simultaneously mitigate noise caused by disagreement among experts by having a common machine standard to refer to. The approach to automation of segmentation and detection of gas flares in this thesis will be through instance segmentation. We implement instance segmentation through the popular one-stage model architecture of YOLOv5 (You Only Look Once) [36]. We outline the reasons why we chose to use YOLOv5 in Chapter 3. Furthermore, we will ensemble together multiple different YOLOv5 model architectures to enhance performance. To this end, we adapt Weighted Box Fusion (WBF) [70] for instance segmentation.

Due to the often subjective nature of segmenting flares, discerning the true

underlying segmentations (ground truths) needed for accurate benchmarking performance is challenging. To address this, we have recruited six experts who have graciously agreed to assist with this project. These experts have done segmentation on a test data set, enabling us to benchmark the experts against each other's work as well as evaluate how the model(s) compare. The objective of this thesis, then, is to develop a deep learning based automated system for detecting and segmenting flares, establish methodology for model evaluation, and use this to compare the performance of the model with that of human experts.

As mentioned earlier, it is of interest to discern between useful and not useful flares. However, only the first problem of segmenting flares will be tackled. Approaches on how one can extend the work in this thesis to allow for discerning between types of flares will be discussed in Chapter 5. Although this thesis primarily focuses on the automatic detection and segmentation of methane seepage, the methodology is more general and can be applied to other types of gas seepages, such as carbon dioxide, or gas leaks from subsea oil and gas facilities [33].

## 1.3 Contributions

The work presented is, to our knowledge, the first application of instance segmentation on marine gas seepage in single beam echo sounder data. We show how to adapt the Brier score [8] for instance segmentation and demonstrate that for the problem at hand, ensemble models are better than single models, and that ensembling models together with WBF is better than ensembling models together with Non-Maximum Suppression (NMS) [74]. Further, we show that our ensembled models are largely indistinguishable from the average human expert in terms of the Brier and Mean Average Precision (mAP) [74] metrics, and provide evidence of potentially more accurate segmentations. Despite the strong quantitative performance we still consider our models as inferior due to a qualitative analysis done. Areas for improvement are identified and direction for future work is given. We note that with these improvements it might be possible to bridge the gap between human and machine-level gas flare segmentation and detection.

As part of the work, a dataset had to be created. Starting with 4,057 images, 1,414 were determined to have flares in them. In these 1,414 images, 5,142 objects identified as flares were segmented. This dataset was created by the author of this thesis, who can be considered to be a trained non-expert. The created dataset can serve as a starting point for others who wish to improve upon it or create a new dataset. Making changes to the dataset should be a

less daunting task than starting from scratch.

## 1.4 Outline

In Chapter 1, we introduce the context and problem statement, and highlight the contributions of this thesis. In Chapter 2, we review the fundamentals of deep learning and provide a brief history of deep learning in object detection, along with the concepts we deem necessary to understand YOLOv5. Chapter 3 outlines how we came to use YOLOv5. Additionally, the architecture of YOLOv5, including how we train it, is described in detail. Further, we show how we adapt WBF for instance segmentation and by extension how we ensemble models together. This chapter also describes the dataset and the base evaluation metrics that will be used for evaluating results in Chapter 4. In Chapter 4, we present our results and explain how to interpret them. We follow this up with quantitative and qualitative analysis of the results for both the ensembles and experts. Finally, Chapter 5 concludes the thesis by providing directions for future work summarizing the results and summarizing the results.

# 2

# Theoretical Background

This chapter assumes the reader has some familiarity with deep learning and statistics. The chapter will serve as a review of concepts important for all of deep learning, as well as concepts more specific for YOLOv5. Detailed mathematical exposition and precise algorithmic breakdown is largely omitted in favor of illustrations and descriptive text.

## 2.1 Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that focuses on the study and development of algorithms enabling computers to perform tasks without explicit instructions.

All ML algorithms process some form of data. To allow for an explicit study of the data, we convert each example in the dataset into vectors $\boldsymbol{x} \in \mathbb{R}^n$. These vectors could represent images, videos, words, paragraphs, a combination of the preceding, and so on. Some tasks of interest in ML include:

- **Regression:** In this setting, the objective is to predict the correct numerical value given the input vector $\boldsymbol{x}$. More formally, we aim to learn a function $f : \mathbb{R}^n \to \mathbb{R}$. The simplest problem in this setting is linear regression, where one assumes a linear relationship between the independent and dependent variables. Examples of regression problems include

predicting the adult height of a child, house prices, life expectancy, and stock prices.

- **Classification:** The classification scenario is much like the regression setting, except instead of predicting any numerical value, the output is constrained to a finite number of discrete outputs, called classes, i.e., the interest lies in learning a function of the form $f : \mathbb{R}^n \to \{1, \ldots, k\}$. Here, $k$ represents the total number of classes. The output of the function will typically be one or multiple values between 0 and 1, possibly representing the output as a discrete probability distribution before applying some rule to select the class. This rule usually consists of selecting the class with the highest numerical value.

- **Density Estimation:** In this scenario, the goal is to learn a good approximation of the underlying distribution that produced the data vectors $\boldsymbol{x}$. More formally, we aim to learn a function $p : \mathbb{R}^n \to \mathbb{R}$, where $p$ is either explicitly a probability distribution or can be interpreted as such. An example of density estimation using deep learning is the method of Generative Adversarial Networks (GAN) [24], where one neural network is trained to discriminate between real and fake examples from the dataset, and another is trained to generate fake examples. In the ideal scenario, the generator learns to output examples indistinguishable from the true underlying distribution, leaving the discriminator with only the option to guess with a 0.5 probability if the example is fake or not.

- **Other Tasks:**

  - Imputation of missing values: given a $\boldsymbol{x}$ with entries missing, learn to predict those missing entries.

  - Denoising: Given some corrupted version of $\boldsymbol{x}$, predict the original $\boldsymbol{x}$ before the corruption process.

  - Machine translation: Translate from one language to another.

  - Large language models: Develop models capable of understanding and generating human-like text.

The list above represents just a small portion of the tasks studied in ML. Common to all these tasks is either the implicit or explicit modeling of the underlying probability distribution. If the algorithm aims to perform well on new examples, i.e., to generalize, it needs to capture the underlying structure of the data and infer from it [23].

Typically, regression, classification, and any problem where there is a distinct label $\boldsymbol{y} \in \mathbb{R}^n$ associated with $\boldsymbol{x}$ are dubbed supervised learning problems. In these tasks, the objective is to infer $\boldsymbol{y}$ based on $\boldsymbol{x}$. Tasks where there is no clear $\boldsymbol{y}$ associated with each $\boldsymbol{x}$ are often categorized as unsupervised learning tasks. Unsupervised learning tasks include, among others, dimensionality reduction through autoencoders [3], clustering through deep clustering algorithms [63], and generative modeling through GANs [24].

### Dataset Split and Hyperparameters

In ML, it is commmon to divide the dataset into three parts: a training dataset, a validation dataset, and a testing dataset. The training dataset is the largest, followed by the validation, and then the testing dataset. For example, a possible split could be 80% for training examples, 15% for validation examples, and 5% for testing examples. Ideally, each set should be large enough to be a good approximation of the underlying probability distribution that generated the data.

The training dataset is the primary dataset that the model uses to understand and learn the underlying patterns. The validation dataset is used to monitor the model's performance on independent data during training, as well as to tune hyperparameters. Hyperparameters are parameters that are not learned during explicit training but are generally set by a human or found through methods such as a grid search[1]. The testing dataset is the final set of examples used solely to benchmark the model's performance.

## 2.2 Neural Networks

Neural networks are a set of algorithms modeled loosely after the human brain, designed to recognize patterns [1]. The study of all things surrounding (deep)[2] neural networks is called deep learning.

The basic building block, which all neural networks use, is the perceptron or neuron. The perceptron takes as its input other perceptrons, or if it's the first layer it takes as its input the examples in the dataset. We associate with each

1. A grid search involves defining a set of possible discrete values for each hyperparameter individually and then finding the optimal ones by exhaustively trying every combination, using the validation set for performance evaluation.
2. Neural networks are structured in layers. Neural networks with two or more hidden layers are conventionally called deep neural networks, but since virtually all neural networks of interest are more than one hidden layers deep this is sometimes omitted.

input to the perceptron, $x_j \in \mathbb{R}$, $j = 1, \ldots, d$, a weight $w_j \in \mathbb{R}$ and a bias term $b \in \mathbb{R}$:

$$y = \sum_{j=1}^{d} w_j x_j + b. \tag{2.1}$$

Here, $y$ is sometimes called the preactivation value, as this value is necessarily input into a non-linear function $f$ called the activation function, outputting the activation value, $a$, giving us the complete perceptron:

$$a = f(\sum_{j=1}^{d} w_j x_j + b). \tag{2.2}$$

We can build neural networks by stringing together these neurons in various ways. The simplest form a neural network could take is the form of a fully Connected Neural Network (FCNN). In a FCNN, all neurons in a layer are connected to all the neurons in the previous layer. A simple illustration of a FCNN with one input layer, $L - 2$ hidden layers and one output layer is shown in figure 2.1. As an example, assuming two hidden layers, one input layer consisting of $\boldsymbol{x}$ and an output layer outputting $\boldsymbol{a}$, we can write out a four layer FCNN as:

$$\boldsymbol{a} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{W}_3 \sigma(\boldsymbol{W}_2 \sigma(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3, \tag{2.3}$$

where we have gone over to using matrix notation, assume that all the dimensions of the matrices and vectors align, and the activation function $\sigma$ is applied in an elementwise fashion. $f$ is the full function defined by the neural network with parameters $\boldsymbol{\theta} = (\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3)$

In practice, creating a shallow but wide neural network is not often seen. Rather, the depth of the neural network seems to be of crucial importance [28]. In 2014, a state-of-the-art neural network in image classification had a depth of 19 layers [69]. In 2015, on the same problem, a state-of-the-art neural network had 110 layers, but fewer parameters [28].

Neural networks are function approximators. Indeed, the universal function approximation theorem states that a neural network with one hidden layer can approximate any continuous function to arbitrary precision provided the width of the neural net goes to infinity [32]. Similar results can be shown for neural networks with infinite depth [45].

## 2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) represent another way to connect perceptrons together. Unlike in FCNNs, where every perceptron in every layer

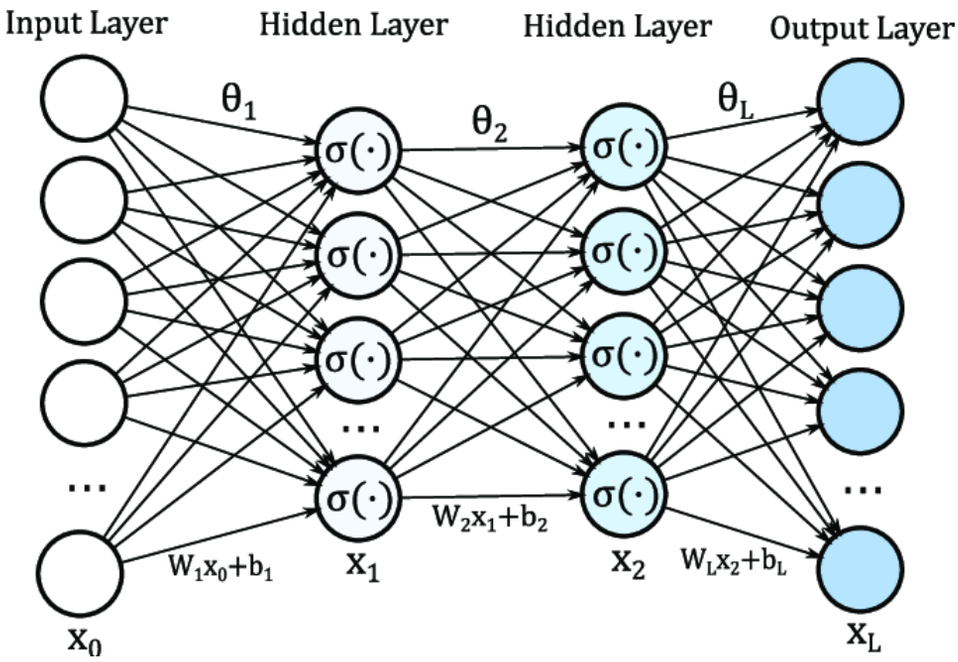



**Figure 2.1:** Illustration of a FCNN. Every neuron in a layer is connected to all the preceding neurons. If we set $L = 4$ in the illustration we get a FCNN with two hidden layers.

is connected to all preceding perceptrons with distinct weights, CNNs utilize parameter sharing and sparse interactions.

Figure 2.2 illustrates the nature of interaction between perceptrons in terms of the convolution operation. The convolutional operation consists of performing an elementwise multiplication between the convolutional filter and the area to which the filter is applied, followed by summing up the results. The figure shows a $3 \times 3 \times 1$ convolutional filter applied to a $5 \times 5 \times 1$ input tensor, padded with zeros to preserve the width and height. The convolutional operation applied in this manner induces equivariance to translation; that is, upon moving the input, the output will shift in the same manner. This property is useful as it means that if the CNN learns to detect an object, it will detect it no matter where it is in the input. For instance, it is known that CNNs applied to images usually learn edge detectors in the early layers [86]. The same edges appear practically everywhere in the image, so learning a single[3] filter for detecting them is practical. Additionally, the filters learned, combined with the convolution operation, can be thought of as doing template matching [27]. To see this, imagine vectorizing the areas the filters are applied to, as well as the

3. Edges vary in their orientation, so possibly many orientations of edge detection filters are learned.

filter itself, then assume that the areas and the filter are constrained to have the same Euclidean norm. Now, when the vector defining the area and the filter are equivalent, the dot product between them is at its maximum, hence giving the largest possible preactivation value precisely when the area it is applied to is the same as the filter. This is why edge detection filters visually look like edges, and why one can visually study the filters applied to the input image to see what is learned. Beyond the first layer, it is practically impossible to tell what the filter represents by visually inspecting them, and one has to resort to other techniques to tell what is learnt [2, 86, 71].

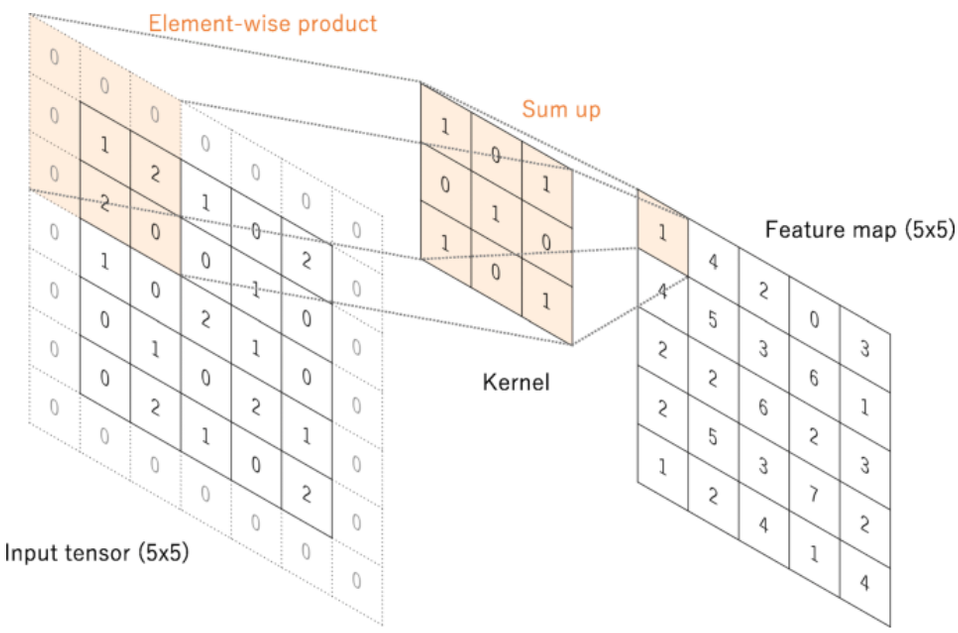



**Figure 2.2:** Illustration of the convolution operation used in a CNN.

Smaller filters such as $3 \times 3$ and $1 \times 1$ are the standard in CNNs [69], although occasionally larger filters like $7 \times 7$ are used in certain parts of the network. Applying a $3 \times 3 \times c$ filter twice uses only $18c$ parameters and provides a $5 \times 5$ receptive field[4], whereas applying a $5 \times 5 \times c$ filter once requires $25c$ parameters for the same receptive field. Here, $c$ refers to the number of channels in the filter. For instance, the input image is usually an RGB image, meaning the filters applied to this input image have 3 channels. The receptive field of neurons grows in a hierarchical manner as one progressively moves down the network, due to the convolution operation. Because of this, we say that CNNs learn through local spatial hierarchies, which we tend to believe is helpful for solving

4. The receptive field refers to the input neurons that the neuron is a function of. In very deep CNNs, neurons in later layers can have the entire input image as their receptive field. Large receptive fields are important for neurons in as they allow the neurons to "reason" with more information.

certain problems.

The power of CNNs comes from their ability to effectively utilize (through the convolution operation) the inherent spatial information embedded in data such as 2-D images or 3-D videos while being incredibly efficient. As an example of the efficiency, assume we have an image of width 320 pixels and a height of 240, and that the CNN applied to it is defined by a single convolution. A convolution operation with a $3 \times 3$ filter would reduce the image size to $318 \times 238$ pixels and would require $318 \cdot 238 \cdot 10 = 756,840$ floating point operations (9 multiplications and 1 addition per pixel). Implementing the same transformation (or any transformation for that matter) with an FCNN would require $318 \cdot 238 \cdot (320 \cdot 240 + 1) = 581,260,6884$ floating-point operations, making the CNN 7,680 times more computationally efficient. Additionally, the CNN requires only storing 9 parameters compared to the 581 million for the FCNN, making it far more memory efficient.

Another point on the efficiency of CNNs is that the layers using the convolution operation can take input of any size, whereas something like an FCNN must have a fixed-size input. The ability to take input of any size can have implications for the total amount inference time. Larger inputs can be processed faster than smaller ones, relatively speaking, thanks to the ability to reuse computations done by the convolutional layers. For instance, AlexNet [42] takes 1.2 ms, on a 2014 Graphics Processing Unit (GPU), to output a $1 \times 1$ grid cell of class probabilities for a $227 \times 227$ image. However, for a $500 \times 500$ image, AlexNet takes only 22 ms to produce a $10 \times 10$ grid of class probabilities [51]. This means it takes just 0.22 ms per grid cell of class probabilities, making the process more than five times faster per grid cell compared to processing the smaller image.

## 2.4 Loss Function

The loss function of a deep learning problem defines the objective and measures the error or discrepancy between what the model predicts and what the true value should be for those predictions. Generally, the aim is to find the point that minimizes the loss function[5].

One of the simplest loss functions is is the Mean Square Error (MSE): $J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} ||\boldsymbol{y}_i - f(\boldsymbol{x}_i; \boldsymbol{\theta})||^2$, where $i$ denotes an example in the dataset consisting of $n$ pairs $(\boldsymbol{y}_i, \boldsymbol{x}_i)$ and $f(\boldsymbol{x}_i; \boldsymbol{\theta})$is the output of the model with parameter $\boldsymbol{\theta}$

5. Pure minimization is not always the goal as we also have to consider the possibility of overfitting (to the training dataset).

and input $\boldsymbol{x}_i$. In a classification scenario with 3 classes, $\boldsymbol{y}_i$ could, for instance, be one hot encoded, i.e. take on a value of $[0, 0, 1]$, $[0, 1, 0]$ or $[1, 0, 0]$. If the value of $f(\boldsymbol{x}_i; \boldsymbol{\theta})$ were the same as $\boldsymbol{y}_i$ for all examples, we would achieve a loss of 0.

MSE is intuitive and easy to define, but that does not mean it should the loss function of choice in most applications. Borrowing from statistics, if we make some assumptions about the data, we can get a loss function with more rigorous justification.

Suppose we have a dataset of $n$ examples $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ sampled independently from some unknown ground truth probability distribution $p_{true}(\boldsymbol{x})$. Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distributions over the same support as $p_{true}(\boldsymbol{x})$, defined by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, where $\boldsymbol{\Theta}$ is a subset of finite Euclidean space. We assume that for some $\boldsymbol{\theta}$, $p_{model}(\boldsymbol{x}; \boldsymbol{\theta}) = p_{true}(\boldsymbol{x})$ for all $\boldsymbol{x}$. Then, the Maximum Likelihood Estimator (MLE) [9] for $\boldsymbol{\theta}$ is:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{model}(\boldsymbol{x}_i; \boldsymbol{\theta}). \tag{2.4}$$

If we divide by $n$, we can express the estimator in terms of the empirical distribution of the data, $\hat{p}_{true}(\boldsymbol{x})$:

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{true}} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta}). \tag{2.5}$$

Multiplying by -1, we equivalently get:

$$\boldsymbol{\theta}_{ML} = \arg\min_{\boldsymbol{\theta}} H(\hat{p}_{true}, p_{model}) = \arg\min_{\boldsymbol{\theta}} - \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{true}} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta}). \tag{2.6}$$

$H(\hat{p}_{true}, p_{model})$ is the cross-entropy between $\hat{p}_{true}$ and $p_{model}$. Because $\hat{p}_{true}$ is not a function of $\boldsymbol{\theta}$, we can also equivalently minimize the Kullback–Leibler (KL) divergence, $D_{KL}$, between the distributions:

$$\boldsymbol{\theta}_{ML} = \arg\min_{\boldsymbol{\theta}} D_{KL}(\hat{p}_{true} || p_{model}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{true}} \Big( \log \hat{p}_{true}(\boldsymbol{x}) - \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta}) \Big). \tag{2.7}$$

Therefore, upon optimizing for the MLE for $\boldsymbol{\theta}$, we see that we also minimize both the KL divergence and cross-entropy, which are both measures of dissimilarity between distributions. Thus, we can view MLE as trying to make the model distribution $p_{model}$ more similar to the empirical distribution $\hat{p}_{true}$. This provides a view on how overfitting occurs in deep learning - the neural network defining $p_{model}$ tries to replicate the infinite density found at points defined by $\hat{p}_{true}$, neglecting to put density on the points around it, which is required for generalization.

To adapt the preceding into a loss function for the supervised learning scenario, we assume that some joint distribution $p_{true}(\boldsymbol{x}, \boldsymbol{y})$, which defines the conditional distribution $p_{true}(\boldsymbol{y}|\boldsymbol{x})$, generated the data $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ in an independent manner. We could do the same derivation as above, but this time for $p_{model}(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$, and we would similarly as before get the MLE as:

$$\boldsymbol{\theta}_{ML} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} -\log p_{model}(\boldsymbol{y}_i|\boldsymbol{x}_i; \boldsymbol{\theta}), \tag{2.8}$$

where the loss function, dubbed cross-entropy or Negative Log-Likelihood (NLL), is:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} -\log p_{model}(\boldsymbol{y}_i|\boldsymbol{x}_i; \boldsymbol{\theta}). \tag{2.9}$$

MLEs have strong theoretical support; under the independent and identically distributed assumptions, and some regularity conditions, they are consistent and asymptotically efficient estimators [9]. Meaning that as the sample size increases, MLEs converge to the true parameter values, and among all consistent estimators, they achieve the smallest possible variance in the limit.

The NLL loss function requires the assumption of a model distribution. For classification tasks, one would typically assume a categorical distribution. In regression problems, a common choice is the multivariate normal distribution with some mean $f(\boldsymbol{x}; \boldsymbol{\theta})$, possibly defined by a neural network, and the identity matrix $\boldsymbol{I}$ as the covariance matrix:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{I}). \tag{2.10}$$

Interestingly, using this model distribution recovers the MSE loss function, providing justification for its use beyond its simplicity.

More complex loss functions that combine NLL with other types of loss functions, as shown in YOLOv5 (to be discussed later), are not at all uncommon in deep learning.

### 2.4.1 Empirical Risk

More generally, the loss function we are concerned with minimizing is the expected generalization error [38], also known as risk, for some loss per example function $L$ :

$$R(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_{\text{true}}} L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}). \tag{2.11}$$

This expectation is taken over the true underlying distribution generating the data, which we usually have no access to. Therefore, we aim to approximate this

loss function by replacing the true underlying distribution with the empirical distribution, giving us the empirical risk:

$$\hat{R}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{true}}} L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}). \tag{2.12}$$

Pure minimization of this loss is prone to overfitting. Indeed, a sufficiently large neural network can simply memorize the training dataset [87]. Strategies to help improve generalization and, by the same token, reduce overfitting will be discussed in the regularization section.

## 2.5 Optimization

Optimization in deep learning refers to how one finds the parameters which optimize the performance of the neural network on some metric. In image classification, the metric might be accuracy, in instance segmentation, it could be mAP50(defined in Chapter 3), and if no obvious metric is available, the loss function itself can be considered as the metric. In any case, finding the parameters which significantly reduce the loss function is involved.

Optimizing the performance of the neural network directly on certain metrics is not always possible. For instance, in classification, using the 0-1 loss directly (accruing 0 loss for a correct prediction, 1 for incorrect) is not possible since the algorithms used in deep learning require useful derivatives (the 0-1 loss is discontinuous). Moreover, when optimizing for a deep learning problem, finding a global minimum or, more generally, just pushing the loss as low as possible is not the deciding factor in determining the desired parameters. Instead, a common practice is to monitor the performance of the desired metric on the validation set and halt training when the criteria set by the early stopping algorithm (discussed later) are met.

Premature stopping due to getting stuck in local minima is believed to be a non-issue for large neural networks. As the size of the network increases, it is conjectured that the likelihood of encountering a high-loss local minimum becomes vanishingly small. Instead, it is believed that most local minima are close to the global minimum in terms of loss, making the distinction between the two largely irrelevant. The optimization problems previously believed to be caused by getting stuck in high-loss local minima are instead thought to be caused by the abundance of saddle points in combination with high-loss plateaus [10, 12]

In practice, the first-order method gradient descent and variations on it are used to incrementally traverse the loss landscape. Second-order methods are computationally expensive, and directly using a method like Newton's method

for optimization is only feasible for small networks [38]. Instead, high-level approximations are needed to make second-order methods work [75]. One iteration of gradient descent can be succinctly written as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{2.13}$$

where $J$ is a loss function that decomposes as a sum over the $n$ examples in the training dataset:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{true}} L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}), \tag{2.14}$$

and $L$ is some per example loss, such as the per example loss defined by NLL: $-\log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$. Additionally, $\epsilon$ is some small learning rate and $-\epsilon\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ indicates the step size and direction taken as determined by the gradient at the point of $\boldsymbol{\theta}$.

In practice, computing the gradient using the entire dataset is not ubiquitous. For datasets with examples in the millions and beyond, the computational cost of computing the gradient becomes too great to handle. This is especially true for large language models, which are often said to be "trained on all the text on the internet." " ' ' " ' '" ' ' ' Instead, smaller approximations of the gradient are computed using a minibatch of $n' \ll n$ examples $\mathbb{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_{n'}, \boldsymbol{y}_{n'})\}$:

$$\nabla_{\boldsymbol{\theta}} J'(\boldsymbol{\theta}) = \frac{1}{n'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n'} L(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}). \tag{2.15}$$

This batch of examples is usually sampled uniformly without replacement from the full set of examples. Replacing the gradient with the minibatch approximation in gradient descent is known as Stochastic Gradient Descent (SGD). Updating once with all the examples in the training dataset using minibatches is known as one epoch[6].

The algorithm used to compute the gradients for the parameters in the neural network is called the backpropagation algorithm [64]. This algorithm stores the output of the neurons in the forward pass and then computes the gradients in each neuron, starting with the final loss neuron, in a backward pass. The significance of this algorithm is in the fact that it stores the preactivation values in the forward pass and the gradients in the backward pass. This allows one to effectively use the chain rule of calculus to recursively compute the gradients

6. Some datasets are so large that a full pass over the entire dataset, concluding an epoch, is not possible before training is stopped [23]. In these scenarios we instead describe training in terms of update iterations, where one iteration is updating the weights once with SGD

of all the preceding neurons of a neuron using only the preactivation value of that neuron along with the gradients of the neurons in its subsequent layer. The activation values are used to compute the gradients of weights and biases as gradients are backpropagated.

The backpropagation algorithm trades memory for compute, as one could imagine recomputing the gradients in the backward pass each time the respective gradient is needed. This would, however, incur exponential computational cost with the layer depth as the same values would have to be recomputed over and over again [23].

### 2.5.1 Optimization Techniques

There are many techniques used to ease the difficulties associated with optimization in deep learning. Here, we will discuss some of the most common and widely used optimization techniques in deep learning, which are also used in YOLOv5.

#### Momentum and Adaptive Learning Rates

Modifying the gradient descent algorithm is one of the most straightforward optimization changes one can make.

Momentum-based changes involve incorporating previous update steps to simulate the real physical phenomenon of momentum, i.e., where a ball can accumulate speed downhill via gravity but eventually comes to a halt when faced with sufficient opposing forces. The explicit change we make to equation (2.13) is to add a term $\boldsymbol{m}$, which we can interpret as the momentum of a ball in a sloped environment with gravity and some viscous fluid providing resistance to it [23]:

$$\hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{m}, \tag{2.16}$$

$$\boldsymbol{m} \leftarrow \alpha \boldsymbol{m} - \epsilon \nabla_{\boldsymbol{\theta}} J'(\hat{\boldsymbol{\theta}}), \tag{2.17}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m}, \tag{2.18}$$

where $\alpha \in [0, 1)$ is used to decay previous additions. The inclusion of equation (2.16) transforms basic SGD with momentum into SGD with Nesterov momentum [72]. Nesterov momentum is a slight improvement on the basic version; it evaluates the gradient after applying the current momentum to add a correction factor. Momentum can help in traversing large flat regions, characterized by small but consistent gradients, more quickly than usual. The inertia gained can also help push through optimization killing saddle points.

Finally, the preferred path of SGD might be plagued by some type of oscillating or zigzagging behavior. Adding a momentum term helps smooth out these oscillations by accumulating momentum in the relevant direction that decreases the loss.

The method of adaptive learning rates dynamically adjusts each parameter's learning rate during training, based on individual gradient values. For instance, the AdaGrad algorithm [16] accumulates the square of the gradient in a variable $\boldsymbol{r}$, using this accumulation to adapt each parameter's learning rate:

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \nabla_{\boldsymbol{\theta}} J'(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}} J'(\boldsymbol{\theta}), \tag{2.19}$$

$$\Delta\boldsymbol{\theta} \leftarrow \frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \nabla_{\boldsymbol{\theta}} J'(\boldsymbol{\theta}), \tag{2.20}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{2.21}$$

Here, $\delta$ represents a small constant added for numerical stability. The operations of division, square root, and multiplication, indicated by $\odot$, are performed elementwise. AdaGrad accumulates gradients from the beginning of training, which often ends up halting training completely, especially in sufficiently non-convex settings [23].

Another method is ADAM [40], which integrates adaptive learning rates along with momentum. ADAM deals with AdaGrad's problem of continual accumulation of past gradients by decaying past accumulation using a parameter $\alpha \in [0, 1)$, like in equation (2.17). ADAM is known for its robust performance across various hyperparameter settings [23].

While individually fine-tuned optimizers may perform better in specific scenarios, the two best optimizers, as measured by their popularity, are ADAM and SGD with some form of momentum [66].

### Activation Functions

The choice of activation function matters significantly for the optimization of the neural network. At least one of the neurons needs to have a non-linear activation function, or the output of the neural network itself will be a linear function of the input[7]. This becomes clear for FCNNs from equation (2.3) if one removes the $\sigma$ activation function. The same, of course, holds true for CNNs, as convolutional operations can be easily cast as matrix multiplications. And in fact, at lower levels of programming, convolutions are indeed cast as matrix

7. This is a problem as linear models are unable, by definition, to capture the highly non-linear relationships needed to solve the problems found in deep learning.

multiplications to harness the power of highly optimized matrix multiplication routines [88].

Early activation functions commonly used in deep learning include the sigmoid $f(x) = 1/(1+e^{-x})$ and the tanh activation function $f(x) = (e^{2x}-1)/(e^{2x}+1)$. These functions saturate, meaning their derivatives approach 0 quickly outside a small range around 0. This saturation can cause issues with vanishing gradients, i.e., the gradients become so small that the finite precision of the computer causes them to be set to 0. If the gradients of enough neurons are set to 0, then training will be significantly slowed or even halt completely, due to the lack of backpropagated information [42]. Additionally, the maximum value of the derivative of the sigmoid is 0.25, and for the tanh we have that $f'(x) < 1$ for $x \neq 1$. Thus, when applying the chain rule in the backpropagation algorithm, even if one achieves maximum or close to maximal derivatives, one will still get vanishing gradient problems due to the repeated multiplication of factors less than one.

More commonly used is the ReLU (rectified linear unit) function $f(x) = \max(0, x)$ or ReLU-like activation functions such as leaky ReLU [82] and SiLU [58]. These activation functions solve the vanishing gradient problems caused by functions such as the sigmoid and tanh, while holding many other desirable properties. In particular, we highlight the computational efficiency of ReLU. This function has only two possible values for its derivative, which we can determine by the sign of the input. Thus, the only computation we need do is to recognize the sign of the input, which we do once in the forward pass. Also, the issue of not being continuous at 0 for the ReLU is not problematic, as we can assume that the finite precision of the computer was not able to capture the infinite precision needed for the reals. Hence, when the preactivation fed into ReLU is 0, we assume it took on some small value $\epsilon$ instead, where the sign is determined randomly. The practical implementation of this is, again, exceedingly cheap.

### Batch Normalization

Batch normalization [35] refers to the process of normalizing the output of a neuron based on the other outputs in the same batch. To implement batch normalization, assume a batch size of $n$, with outputs $x_1, x_2, \ldots, x_n$ from a neuron, and compute the mean $\mu$ and standard deviation $\sigma$ of these outputs:

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j, \tag{2.22}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_j - \mu)^2 + \epsilon}, \tag{2.23}$$

where $\epsilon$ is a small value added for numerical stability. Next, normalize each activation as follows:

$$y_i = \frac{x_i - \mu}{\sigma}. \tag{2.24}$$

Here, $y_i$ is the new output of the neuron for the $i$th example in the batch. Finally, multiply and add learnable parameters $\alpha$ and $\beta$ to obtain the input for the next layer:

$$y_i' = \alpha y_i + \beta. \tag{2.25}$$

The extra parameters grant the model greater flexibility. They allow the model to adapt the normalization process for each neuron, compensating for the possible loss of representational ability when implementing equation (2.24) [81]. During training, the algorithm maintains running averages of the mean and standard deviation for each neuron. Once training is complete, these running averages are used for normalization instead of the batch specific estimates. This ensures consistent normalization during the inference phase, regardless of batch size. Batch normalization can be applied either before or after the activation function, although before is most commonly used. It is also possible to normalize neuron outputs based on the $L^p$ norm, with $L^1$ and $L^\infty$ normalization possibly outperforming regular batch normalization [65].

Batch normalization works by smoothing the highly non-convex loss landscape of neural networks, thereby mitigating vanishing gradient problems, reducing sensitivity to hyperparameter settings, and improving the directionality of the gradient [65]. Finally, normalizing across the batch, as just described, is not the only method of normalization [81, 42]

### Skip connections

Skip connections are a general technique where the input to a layer includes not only the neurons from the preceding layer but also those from many previous layers. This technique has been used in many state-of-the-art image classification CNNs over the years [26]. In [28], the first instance of skip connections was presented, implemented through the addition of an identity mapping. They conjecture that if an identity mapping is indeed optimal for a specific layer, then this reformulation helps with learning it[8]. Additionally, the identity mapping helps in backpropagating the more easily gradient as it passes unchanged through the identity mapping. They find that the addition of the

8. The reasoning being that learning the weights in a layer needed for an identity mapping is harder than learning to push the weights to 0.

skip connection allows for the training of exceedingly deep neural networks, more than 1000 layers, which had previously been impossible due to vanishing gradient problems.

Rather than allowing only one skip connection per input, [34] includes all previous feature maps as inputs, where the skip connection is now implemented through the concatenation of the feature maps. The inductive bias[9] is now that preserving and reusing features throughout the network can be beneficial for the learning process. As in [28], they also achieve state-of-the-art performance in image classification on several datasets.

Skip connections aid optimization, in part, by improving gradient flow through more direct or shorter pathways and, by the same token, mitigate vanishing gradients. [44] finds empirically that, much like batch normalization, skip connections can help in optimization by smoothing the loss landscape. An example of such smoothing is illustrated in Figure 2.3.

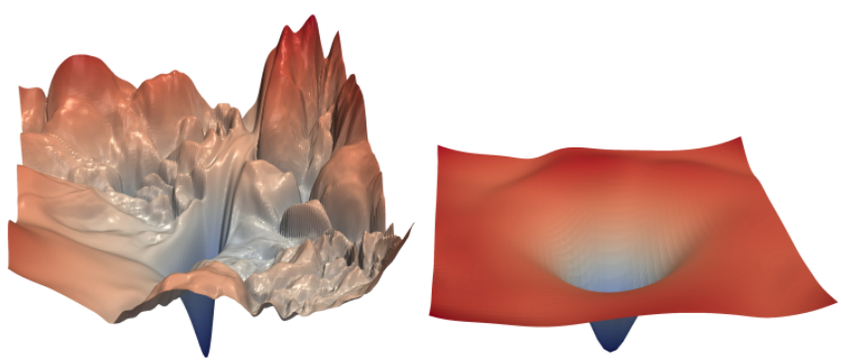



**Figure 2.3:** Image from [44], visualizing the smoothing effect of skip connections on the loss surfaces of ResNet-56 [45] with and without skip connections.

## Gradient Clipping

Sometimes, instead of vanishing, gradients blow up, caused by the accumulation of large gradients during backpropagation due to repeated additions of gradients greater than 1. In the loss landscape, we can interpret the areas where this occurs as being cliff-like structures. Once these cliff structures are reached, the gradient blows up, resulting in large steps being taken. These large steps can undo a lot of previous gradient updates, causing significant

9. Inductive biases refer to assumptions baked into a model, which aim to help it generalize to new data. The convolutional operation used in CNNs introduces inductive biases that translation equivariance and local spatial hierarchies are helpful for image problems.

slowdown in training or even making convergence impossible. To mitigate this, we use the gradient clipping heuristic, with the idea being that the gradient only tells us the optimal direction, not the step size.

Gradient clipping can be done using element-wise clipping, where we constrain each value in the gradient to be within some interval $[-t, t]$. This method of clipping can change the direction of the gradient. This, however, is not a problem, as it is possible to converge even when taking a step in a random direction at gradient explosions [23]. Another method of gradient clipping is based on the norm $||\boldsymbol{g}||$ of the gradient $\boldsymbol{g}$, i.e., if $||\boldsymbol{g}|| > t$ then:

$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}t}{||\boldsymbol{g}||}, \tag{2.26}$$

where $t$ is some threshold value [57]. In practice, both methods of gradient clipping perform similarly well [23].

### Learning Rate Scheduler and Warm-Up

In practice, using a constant learning rate as in equation (2.13) for SGD is not common. Instead, we decay the learning rate through a specific scheme. One learning rate scheme involves decaying the learning rate, now denoted as $\epsilon_k$, linearly:

$$\epsilon_k = (1 - \frac{k}{\tau})\epsilon_0 + \frac{k}{\tau}\epsilon_\tau, \tag{2.27}$$

and another learning rate decay scheme is cosine scheduling:

$$\epsilon_k = \epsilon_\tau + \frac{1}{2}(\epsilon_0 - \epsilon_\tau)\left(1 + \cos\left(\frac{k\pi}{\tau}\right)\right), \tag{2.28}$$

where $\tau$, $\epsilon_0$, and $\epsilon_\tau$ are hyperparameters. Once $k = \tau$ is reached, where $k$ can represent iterations or epochs, it is common to set $\epsilon_k$ to $\epsilon_\tau$.

There are also approaches where the learning rate is reset after meeting certain criteria. For instance, in [52], cosine scheduling is used, but the learning rate defined by it is periodically reset according to a function based on the epoch number. They find that this scheduling improves performance on the CIFAR-10/100 datasets [41].

Learning rate decay allows for faster convergence and can help generalization. A conventional explanation is that large steps in the beginning help escape local minima, and small steps help avoid oscillation in latter stages of training. [85] finds experimentally that an initial large learning rate helps mitigate memorization of noisy data, and that decay over time assists in learning complex patterns.

Warm-up is another learning rate scheme, which is used at the start of training. For example, in [28], a constant learning rate of 0.01 is used for the first 500 iterations, before being set to the initial value of 0.1 for the scheduler used for the rest of the run. It is also possible to gradually ramp up the warm-up learning rate before employing the original scheduler [25]. In short, warm-up can reduce convergence times (measured in iterations or epochs) and enhance performance by stabilizing early training through smaller learning rates.

### Pretraining/Weight Initialization

Correct weight initialization is important for training deep neural networks. [22] shows that the heuristically defined weight initialization of $U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$ for the weights and a zero initialization for the biases, where $U$ is the uniform distribution and $n$ is the number of outputs in the previous layer, causes convergence problems and poor generalization. Instead, they derive a weight initialization designed to maintain approximate variance of the activation values and the gradients for all the layers. The result, called Xavier initialization, is to initialize weights such that their variance follows the rule:

$$\text{Var}(W^i) = \frac{2}{n_i + n_{i+1}}, \tag{2.29}$$

where $W^i$ is any of the weights in layer $i$ and $n_i$ is the number of inputs to layer $i$. For instance, if one wants to use uniform distributions for the weights, then, in order to maintain constant variances, the distribution should be:

$$W^i \sim U[-\sqrt{\frac{6}{n_i + n_{i+1}}}, \sqrt{\frac{6}{n_i + n_{i+1}}}]. \tag{2.30}$$

The weight initialization scheme they derived assumed a symmetric activation function with a unit derivative, i.e., $f'(0) = 1$. This weight initialization scheme is appropriate for neural networks where the layers use linear, sigmoid or tanh activation functions.

[29] derives a weight initialization scheme for the ReLU activation function, using the ideas of [22], that is, the activation values and gradients should have constant variances across layers. Their weight initialization rule, called He initialization, is for the ReLU and ReLU-like activation functions and states that the variances of the weights $W^i$ in layer $i$ should follow:

$$\text{Var}(W^i) = \frac{2}{n_i}. \tag{2.31}$$

In [29], it is demonstrated that a 30-layer deep network using ReLU activation functions fails to converge when initialized with the Xavier initialization. However, convergence is achievable and consistent for the network when using He

initialization. For a uniform distribution, the weight initialization scheme using He initialization is given by:

$$W^i \sim U\left[-\sqrt{\frac{6}{n_i}}, \sqrt{\frac{6}{n_i}}\right]. \tag{2.32}$$

Weight initialization schemes with constant variances help optimization by mitigating vanishing gradient problems [22, 29].

Another method of weight initialization is through pretrained weights. Using pretrained weights is an instance of transfer learning, where model weights trained on a general problem are repurposed for another [84]. In section 2.3, it was mentioned that edge detection filters tend to be learned in the early layers of CNNs. To leverage this, assume we have a CNN trained on one problem, and now we want to solve another. Using the same architecture as before, we could initialize the model with the already learned edge detection filters, instead of starting with random weights. This approach requires only fine-tuning the weights to the specific problem, rather than learning them from scratch. Starting with these already learned features can prevent overfitting, especially when dealing with smaller datasets, as the model is already initialized towards more general features. One can use pretrained weights for some or all the layers in the model.

Instead of repurposing the weights used for an entirely different problem, one could pretrain weights by building shallow neural networks for the proposed problem. This method, known as greedy supervised pretraining [4], involves training a shallow neural network, $A$, and then using the outputs of $A$ as inputs to another shallow neural network, $B$, during its training. This process can be repeated multiple times, and at the end, all the disjoint shallow neural networks can be combined and fine-tuned as the final step in the training process.

## 2.6 Regularization

[23] defines regularization as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." Here, generalization error is defined as the expected error for a new test example, which is estimated by the empirical performance on the test set.

Regularization can be designed to encode prior knowledge, such as convolutional filters in a CNN (utilizing local spatial information), and constrain the

neural network to define simpler models, which reduces its capacity[10]. Ensemble methods regularize by combining multiple models, which individually may overfit, but when joined together, smooth out each other's mistakes.

Some of the most common and widely used regularization techniques, which are also used in YOLOv5, will be described next.

### Choice of Batch Size

Smaller batch sizes can improve generalization error, with batch sizes as small as $n' = 1, 2$ possibly giving the best performance. Training with such small batch sizes typically increases the amount of time[11] needed for convergence due to the smaller learning rate required to maintain the stability of the high variance gradient approximation and the need for more steps [23, 55, 44].

A possible explanation for better generalization with smaller batches is that they tend to converge to flatter minima, while larger batches tend towards sharp minima [39]. The intuition here is that the loss in a flat region is insensitive to small perturbations of its parameters, which is what happens when general features are learned, whereas a small perturbation on a sharp minimum can greatly increase the loss, potentially indicating overfitting.

### Explicitly Modifying the Loss Function

Perhaps the easiest way to regularize a model is to express towards a simpler model, which can be done by the loss function explicitly:

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \beta\Omega(\boldsymbol{\theta}^*). \tag{2.33}$$

Here, we have modified the loss function by adding onto it a parameter norm penalty, $\Omega(\boldsymbol{\theta}^*)$, where $\boldsymbol{\theta}^*$ is a subset of $\boldsymbol{\theta}$, and $\beta \in [0, \infty)$ is a hyperparameter that weights the contribution of $\Omega$ to the loss. Large values of $\beta$ place more emphasis on fulfilling the objective defined by $\Omega$.

A common form of $\Omega$ is the Euclidean or $L^2$ norm $\Omega(\boldsymbol{\theta}^*) = \frac{1}{2}||\boldsymbol{\theta}^*||_2^2$. The $L^2$ norm parameter penalty is also known as weight decay and the gradient descent update for $\boldsymbol{\theta}^*$ with it can be shown to be:

$$\boldsymbol{\theta}^* \leftarrow (1 - \epsilon\beta)\boldsymbol{\theta}^* - \epsilon\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}^*). \tag{2.34}$$

10. Capacity here refers informally to the breadth of functions the neural network can learn, as well as how easily it will learn them. High capacity neural networks overfit easily.
11. Low batch sizes typically underutilize the parallel capacity of modern GPUs, leaving compute resources unused which could have been used to speed up training.

For a single update, the $L^2$ regularizer moves $\boldsymbol{\theta}^*$ towards 0 by multiplying it with the shrinking factor $(1 - \epsilon\beta)$ before applying the regular update. Over the course of an entire training run, this slight modification serves to penalize large weights, encouraging the model to maintain smaller weights, thus leading to simpler models.

Also of note is $L^1$ norm regularization, where $\Omega(\boldsymbol{\theta}^*) = ||\boldsymbol{\theta}^*||_1$ applies a sparse preference to the weights, i.e., it tends to make some weights exactly 0. This regularizer can prune away connections between neurons, leading to simpler models.

### Exponential Moving Average Parameters

Exponential Moving Averaging (EMA) of parameters is a general regularization technique that can be used for inference on the validation set or testing set. The technique involves keeping a moving average of the parameters during training. More specifically, after each training step $t$ (or at specific intervals), the EMA parameters $\overline{\boldsymbol{\theta}}_t$ are updated using the current parameters $\boldsymbol{\theta}_t$ of the training model:

$$\overline{\boldsymbol{\theta}}_t = \epsilon\boldsymbol{\theta}_t + (1 - \epsilon)\overline{\boldsymbol{\theta}}_{t-1}, \tag{2.35}$$

where $\epsilon$ is a hyperparameter determining the weighing of current parameters. Values for $\epsilon$ are usually set close to 0, typically in the multiple-zeros range: 0.001, 0.0001, etc. Models using EMA parameters for inference often generalize better to new data. The EMA model represent a more stable version of the trained model, capturing its long term behavior rather than placing possibly too much influence on the most recent data presented to the model (overfitting) [36].

### Dataset Augmentation

The most effective way to improve model performance is by using more data. When additional data is not available, applying transformations to the current dataset to artificially increase its size is a viable option. In fact, data augmentation is an integral part of problems involving image data, as it consistently improves performance [83]. The data augmentations should generally be sensible for the problem at hand. For instance, in a problem involving the detection of boats, flipping the images horizontally is an obvious choice, but vertical flipping might not be beneficial.

There are various strategies to implementing dataset augmentation to. One approach is to apply different augmentations independently to the original

dataset. For instance, applying one augmentation, like horizontal flipping, would double the dataset size. Adding a second, independent augmentation, such as random cropping, to the original dataset and the flipped dataset would quadruple the dataset size (original, flipped, cropped,cropped+flipped). Similarly, a third independent augmentation would octuple the dataset.

An alternative approach involves randomly applying augmentations to the selected images, where each augmentation has been assigned some particular probability of being selected. This method can introduce a diverse range of variations without the need for a combinatorial increase in the dataset size. Additionally, this way of applying data augmentation can be applied to the images comprising a batch during training itself. This removes the need to store extra images, and instead only requires some more computation during training.

Some typical augmentation techniques that might be considered when dealing with image data are shown in Table 2.1.

| Methods | Description |
|---|---|
| Flipping | Flip the image horizontally, vertically, or both. |
| Rotation | Rotate the image at an angle. |
| Scaling Ratio | Increase or reduce the image size. |
| Noise injection | Add noise into the image. |
| Color space | Change the image color channels. |
| Contrast | Change the image contrast. |
| Sharpening | Modify the image sharpness. |
| Translation | Move the image horizontally, vertically, or both. |
| Cropping | Crop a sub-region of the image. |

**Table 2.1:** Table reproduced from [83]. Common data augmentation methods in image processing.

### Ensemble Methods

Ensemble methods involve training several different models separately for a given problem and then having all the models cast a vote for the correct prediction [19]. This kind of model averaging can be looked at as a "wisdom of the crowd" approach, where often a few models will make occasional mistakes, but rarely will every model make the same mistake. In cases where some models make mistakes, the rest of the models will compensate for it. Contrasting this with a regular method, we see that if a mistake is made, it cannot be averaged away by other models. The cases where every model makes the same mistake are rare, making the ensemble approach much less varied than a single

model.

For a concrete example, consider $n$ regression models, each making an error $\epsilon_i$ on examples in the testing set, where $\epsilon_i$ is drawn from some multivariate distribution with mean zero, variances of $\mathbb{E}(\epsilon_i^2) = b$, and covariances of $\mathbb{E}(\epsilon_i \epsilon_j) = v$. If we average the predictions, the output of the ensemble of models will be $A = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i$. Next, the variance of the random variable $A$ is:

$$\mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right)^2\right] = \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n}\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right], \tag{2.36}$$

$$= \frac{1}{n}b + \frac{n-1}{n}v. \tag{2.37}$$

We see that if the errors have a correlation of 1, then $b = v$ and the variance of the averaged random variable is $b$. In the case where the errors are independent of each other, the correlation is 0, i.e., $v = 0$, and the variance reduces to $\frac{1}{n}b$. Therefore, the variance of the ensemble model is at least as low as any of the individual models, and in the case where they are making independent errors, the ensemble model has reduced variance.

In practice, by training neural networks with different architectures, random initializations, random hyperparameters, random selection of minibatches, and so on, we can induce enough variability so that the neural networks make quasi-independent errors. The ensemble of many such models should, on average, be better than any of its parts.

Ensemble methods serve as a way to trade extra computation and memory for improved prediction accuracy and generalization capabilities.

### Early Stopping

Large neural networks are capable of memorizing small datasets [87]. This occurs when training for too long, resulting in the neural network losing its ability to generalize. One can effectively monitor the generalization abilities of the network by logging the performance of the neural network on a validation set during the training process. Once the error of the model on the validation set has increased for a certain number of epochs, we can say with high certainty that the model is overfitting to the training dataset.

The early stopping algorithm halts training when the performance on the validation set has not improved beyond its best point for $p$ number of epochs (or iterations), where $p$ is a hyperparameter called the patience parameter. Figure 2.4 shows an idealized plot of the training process and where early

stopping determines the best stopping point to be. The weights and biases of the network at the "Stop training" point are saved and selected as the best parameters.

Early stopping requires no modification of the underlying training process and only adds a small penalty to the amount of computation and memory needed — the error on the validation set needs to be periodically computed, and the best parameters need to be periodically saved as well.

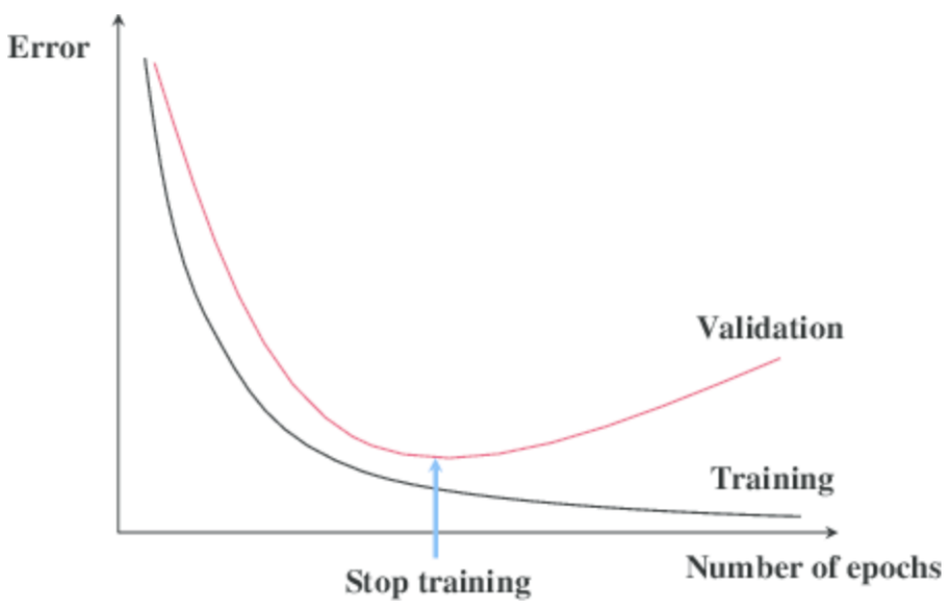



**Figure 2.4:** Idealized plot of how the error on the training and validation set might evolve with the number of epochs.

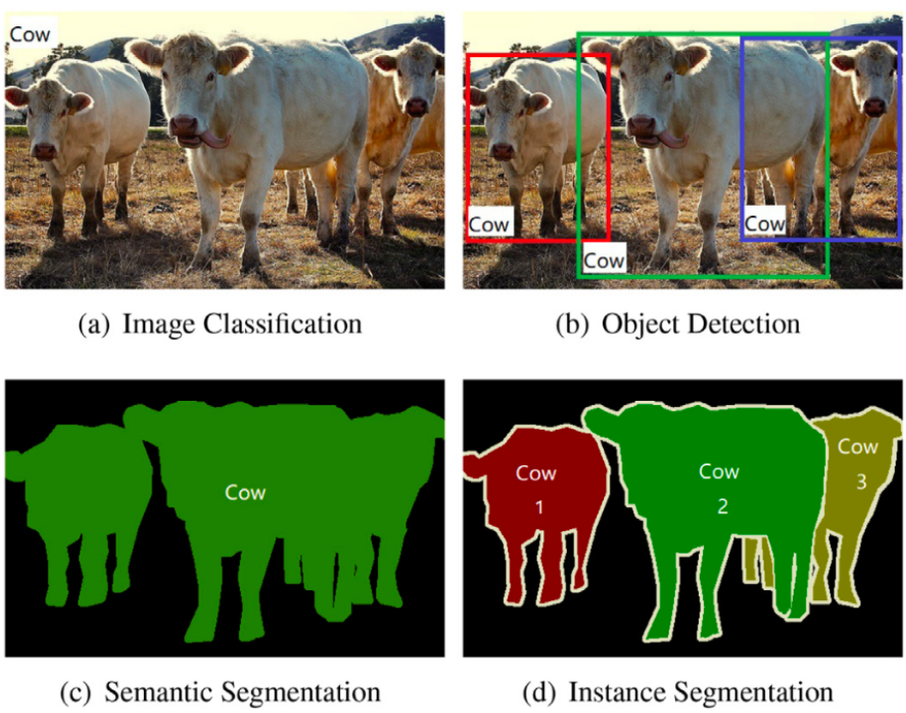
### Optimization Techniques as Regularization

Many of the techniques used to aid optimization can also be considered to have a secondary function as regularization techniques. For instance, batch normalization involves multiplying and subtracting quantities (the standard deviation and mean) whose estimates fluctuate during training. These fluctuating estimates introduce noise into the problem, forcing every layer to learn to be more robust to significant variation in its input and, by extension, making it more difficult to overfit.

In addition to batch normalization, the specific choice of skip connections, activation functions, and pretrained weights can also provide regularizing effects.

## 2.7 Object Detection

Object detection deals with how to detect instances of specific visual objects in digital images. It has numerous applications such as detecting anomalies in medical images, vision systems for autonomous vehicles, facial recognition systems, and various fields of automated inspection, among others. In deep learning, there are three ways of performing object detection: the first is bounding box object detection; the second is semantic segmentation, where each pixel in the image is classified into a class; and the third one is instance segmentation, which is a combination of the preceding two. An example illustrating the different cases is shown in Figure 2.5.




(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

**Figure 2.5:** (a) The input image to some neural network object detection method. (b) Bounding box object detection methods produce bounding boxes, an objectness score, and a class for each individual object. (c) Semantic segmentation involves classifying each pixel in the image, losing the distinction between different objects. (d) Instance segmentation involves correct classification of the pixels as well as maintaining the distinctions between objects, combining the methods in (b) and (c).

### 2.7.1 Approaches to Object Detection

There are three broad methodologies when it comes to object detection: one-stage methods, two-stage methods, and transformer-based methods. Briefly, some early important work on one-stage and two-stage methods will be presented before touching on transformer-based object detection.

#### Two-Stage Methods

The first significant breakthrough featuring the use of a neural network for object detection was a region proposal method combined with a CNN for feature extraction. The method, called R-CNN, performed significantly better on the ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC2013) [13] than its contemporaries. R-CNN first proposes Regions of Interest (RoI) in the input image using selective search [76], then these RoIs are rescaled and fed into a CNN for feature extraction, and finally, these features are input into linear Support Vector Machines (SVM) trained on individual classes. R-CNN is a slow method as it can produce several thousand RoIs, each of which has to be processed by the CNN; furthermore, each output from the CNN has to be processed by a number of SVMs equal to the number of classes.

In 2015, Fast R-CNN [21] was released, boasting over 200 times faster inference than R-CNN while also delivering significantly better performance. Here, R-CNN is improved by allowing input images of any size, made possible by the fact that the RoIs (still computed by selective search) are now not on the input image but are projected onto the last feature map of the CNN used for feature extraction. This reduces the number of forward passes needed for the CNN from several thousand to just one. This projection of features is possible because of the translational equivariance property of CNNs discussed in Section 2.3. The RoIs are then resized using a special case of the Spatial Pyramid Pooling layer (SPP) [31], called RoIPool, before being input into a FCNN that outputs class probabilities and also outputs fine-tuned coordinates of the RoIs. Although the primary purpose of the fine-tuning of the coordinates is more accurate localization, it also opens up a secondary purpose: the recentered RoI can be re-fed into the FCNN, which allows it to classify it again with potentially higher accuracy.

Later that same year, Faster R-CNN [62] was introduced. The main contribution was the creation of a neural network-based region proposal method called Region Proposal Networks (RPN) to replace the selective search used in Fast R-CNN. The RPN takes as its input the last feature map of the feature-extracting CNN and learns to predict RoIs on it. The RPN uses a set of rectangles, called anchors, with several predefined scales and aspect ratios as a means of injecting

prior information. For instance, if the object detection problem involves human detection, then at least one of the anchor boxes should be taller than it is wide to easily fit around a standing human. The RPN can be trained with a CNN independently of the Fast R-CNN or it can be trained jointly [62].

In 2017, another important breakthrough in object detection happened, which was the the Feature Pyramid Network (FPN) [46]. This new feature introduces a top-down pyramid structure with lateral (skip) connections from earlier feature maps to help build high-level semantic feature maps at different scales. The feature pyramid allows the simultaneous prediction of objects at several different scales, typically three. This multi scale prediction scheme helps localization of smaller objects. The addition of the feature pyramid to a Faster R-CNN architecture allowed for state-of-the-art performance on Microsofts COCO (Common Objects in Context) dataset [48].

Associated with each RoI prediction from the RPN is a value called the 'objectness' score. This score is a value between 0 and 1, which we can interpret as the RPN's confidence that the RoI is localized over an object. The objectness score is central to many object detection methods, such as YOLOv5, and will become a common term in the coming chapters.

All the preceding methods are known as two-stage methods because there is a clear two-step process: first, the RPN proposes potential RoIs with an objectness score; second, the RoIs are classified and given refined bounding box coordinates. Although two-stage models are accurate, they can be very slow, primarily due to the need to wait for the RPN to propose RoIs and the need to evaluate the RoIs individually (although they can all be evaluated in parallel with a powerful enough GPU).

### One-stage Methods

In contrast to two-stage methods, which are inherently slow due to their sequential nature, one-stage methods process everything in a single forward pass. That is, they propose RoIs, assign an objectness score, and classify them all in one step.

The first one-stage method developed was the YOLO (You Only Look Once) method in 2015 [61]. Here, instead of proposing regions and objectness scores, and then classifying them, the regions of interest are simultaneously predicted with their objectness and class scores. The method can use any CNN as a backbone[12] for feature extraction, such as the well-known ResNets [45] or

12. For some object detection methods there is a clear three part system. The backbone is one

AlexNet [42]. To adapt these CNNs to YOLO methods, a final convolutional layer, termed the YOLO head, outputting an $S \times S \times (B * 5 + C)$ tensor, is appended. Here, $S$ is the number of grid cells one wants to discretize the input image into, $B$ is the number of bounding boxes associated with each grid cell, multiplied by 5 to account for the five coordinates associated with a bounding box: width, height, $x$ and $y$ coordinates, and an objectness score between 0 and 1 quantifying the likelihood of an object being within the bounding box. $C$ is the number of classes predicted for each grid cell. Due to the one-stage nature of the method, only a single loss function is needed, where the terms related to bounding box regression, objectness, and classification are combined through a sum. This is in contrast to two-stage methods, where sometimes the RPN and the CNN predicting on the RoIs from the RPN are trained disjointly, meaning two wholly separate loss functions.

Another interesting one-stage method from 2015 was the Single Shot Multi-Box Detector (SSD) [50], which employs multi-reference and multi-resolution detection techniques. These distinct techniques end up improving detection accuracy and speed over YOLO. Unlike YOLO, which predicts only on its last layer, SSD computes the class and localization of objects at different scales on various layers of the network. This technique of computing outputs on different scales is later adopted by newer versions of YOLO, such as YOLOv5.

### Transformer-based Methods

Transformer-based object detection methods are not easily classified as either one-stage or two-stage methods.

The paper "Attention Is All You Need," coined as one of the most influential papers in deep learning, was the first to introduce the transformer architecture [77]. Originally designed for machine translation, the transformer's encoder-decoder structure, featuring an attention-based mechanism, has proven effective for a wide range of problems beyond natural language processing, including image classification and object detection.

In 2020, state-of-the-art performance using transformers for image classification was first achieved [15]. They showed that when pretrained with increasingly larger datasets (in the end more than 300 million examples), the performance of transformers improves, eventually surpassing that of traditional CNNs. The performance discrepancies at lesser amounts of data can be explained by noting that transformers lack the inductive biases inherent to CNNs, such as equivariance to translation and locality. These inductive biases are helpful

---

the three parts, which serves to compute useful features for the problem at hand.

for smaller amounts of data, and their omission causes transformers to not generalize well on small datasets.

The first object DEtection TRansformer (DETR) was also introduced in 2020 and achieved competitive results on the COCO dataset. As of late 2023, DETRs obtain the best results on the COCO dataset and have demonstrated beating different YOLO versions in speed and accuracy [53].

### 2.7.2 YOLOv1 to YOLOv4

Multiple incremental improvements were made to YOLO before YOLOv5 was created. Here, we will briefly go through some important changes in the different versions, which end up being utilized in YOLOv5. Details will be expanded upon when the YOLOv5 architecture is discussed in the next chapter.

In the 2017 paper on YOLOv2 [59], perhaps the most significant modification was the introduction of anchor boxes, as previously seen in Faster R-CNN [62]. As mentioned, this allows for the injection of prior information via the shapes and sizes of the anchor boxes, enhancing performance. Additional improvements to the backbone CNN feature extractor were made. These modifications include converting the CNN to be fully convolutional, which allows for the processing of input images of any scale and consequently enables multi-scale training. Multi-scale training is an essential regularization technique that randomly scales the input images during training. In YOLOv5, we implement multi-scale training by scaling images by a factor ranging from 0.5 to 1.5 times their normal aspect ratio. Batch normalization throughout the network is another new feature in the updated CNN. Lastly, there is a change in the output. The class output previously used a categorical distribution over all classes but has been changed to a binary categorical distribution for the individual classes. This can particularly help in cases where multiple classes should have high class scores. For instance, if we have a dog class and a dwarf schnauzer class, then it is reasonable to put high class score on both of these. In the case where we use a categorical distribution over all classes, then it is usually the case that the model will learn to supress the other classes when predicting its prefered class.

In 2018, YOLOv3 [60] was released with some incremental improvements. As before, a superior backbone CNN is used, as measured by its performance on ImageNet. The performance of the backbone CNN architecture on ImageNet is important because training for classification on ImageNet is used as a pretraining step before the YOLO head is added for fine tuning to object detection. This pretraining setup is by many object detection methods, such as the next version of YOLO to be presented. To improve the detection of medium and

smaller objects, which had been a weak point for YOLO, a FPN structure is included. Also, a modified SPP block in the final layer of the CNN is used to increase the receptive field, significantly enhancing performance, while barely increasing the computational load. Finally, the loss function used has been modified. Previously, MSE, as defined in Section 2.4, was used for the class loss, but it has been replaced by the cross-entropy loss. As we noted in Section 2.4, using the MSE for no other reason than it being a simple loss function, is not a good idea. The fact that MSE is unsuited for object detection was exemplified by the performance boost in YOLOv3 when swapping to the cross-entropy loss.

YOLOv1, YOLOv2, and YOLOv3 were all created by the same authors. In 2020, YOLOv4 [5] was released. Although the authors are different, they adhered to the same naming protocol, which has since been the norm for all subsequent significant improvements of YOLO based architectrues on the COCO dataset. Notable changes in YOLOv4 include a more effective CNN backbone, the first introduction of the popular mosaic data augmentation technique, the transition from MSE to the Complete Intersection over Union (CIoU) [89] loss for bounding box prediction, and finally, the modification of the FPN structure with bottom-up connections to replicate the Path Aggregation Network (PANet) [49]feature aggregator. The new PANet structure, like the FPN, allows multi-scale predictions and is an important structure used in YOLOv5, as it helps in detecting objects of varying sizes.

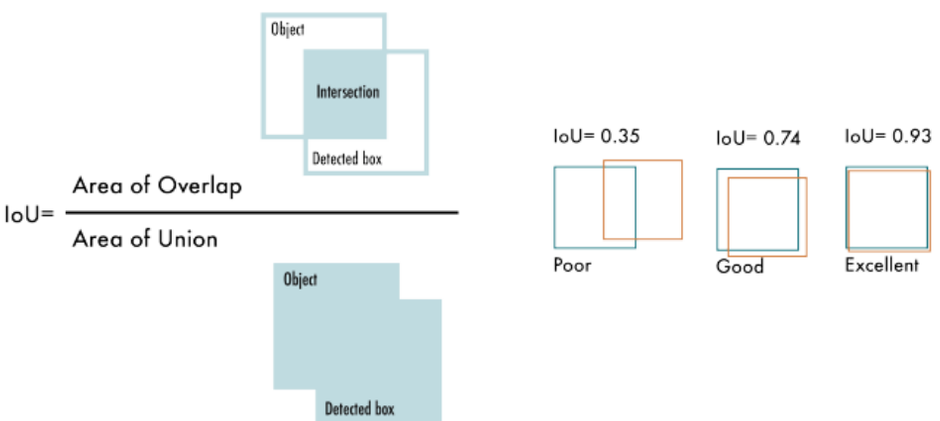
### 2.7.3 Non-maximum Suppression

Due to the often several thousand bounding box predictions that different object detection networks output, many boxes will detect the same object with varying accuracy. To clean up the predictions, the post-processing technique of Non-Maximum Suppression (NMS) [74] or some variation on it, like soft NMS [6] is commonly used. Additionally, the post-processing technique of Confluence represents another post-processing technique, which can possibly be a more robust alternative to NMS or its variations [67].

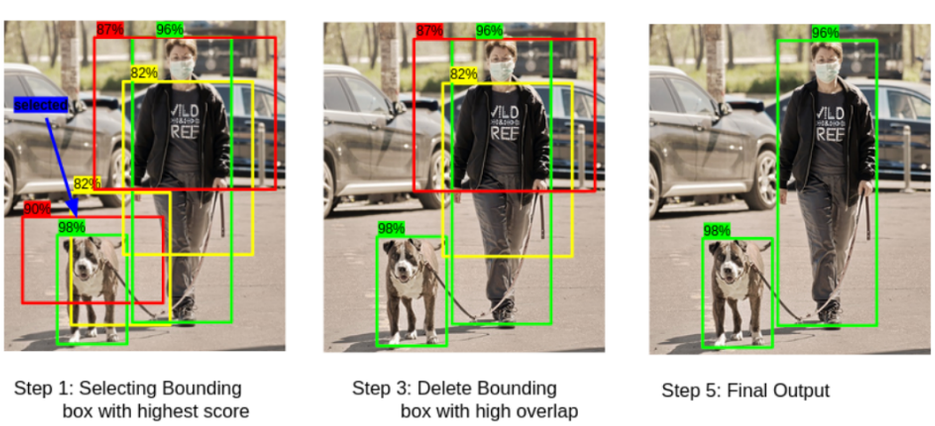
Before providing a brief description of the NMS algorithm we must touch on the concept of Intersection over Union (IoU). IoU measures the extent of overlap between two shapes and is a fundamental concept in object detection. It is calculated as the ratio of the intersection of the two shapes to the union of the two shapes. Figure 2.6a illustrates visually how one computes IoU, and provides three examples of possible IoU values between objects.

The NMS algorithm begins by filtering out all boxes where the objectness score is below a certain threshold. Next, the bounding boxes are sorted by their

associated class score in descending order, and boxes with an Intersection over Union (IoU) greater than a specified threshold with the box having the highest class score are removed. The aforementioned pruning of boxes is done in a class wise manner. Finally, this process is recursively applied to the remaining bounding boxes until no boxes remain. Figure 2.6b succinctly illustrates how NMS works in practice.




**(a)** Illustration of IoU.




**(b)** Illustration of NMS.

**Figure 2.6:** (a) Demonstration of how to calculate IoU with three example calculations. (b) Example of how NMS operates to remove redundant bounding boxes.

### 2.7.4 From Bounding Box Object Detection to Instance Segmentation

All the previously mentioned methods have been focused on bounding box object detection. To adapt them to the more complex task of instance segmentation, some modifications are necessary.

In the case of two-stage methods like Faster R-CNN, the changes are relatively straightforward. Mask R-CNN [30] is the first successful example of adapting Faster R-CNN for instance segmentation. Due to the increased need for more precise localization, a new pooling method, called RoIAlign, is used. This method preserves the spatial positions of the features much more effectively than RoIPool, which is particularly important for tasks like instance segmentation, where keeping the exact spatial layout is crucial. Additionally, a branch is added that takes the RoI aligned features as input, in parallel with the existing branches for classification and bounding box regression. This new branch, which utilize a Fully Convolutional Network (FCN) [51], upscales the RoIs and converts them into binary pixel masks. Outputting the binary masks along with the regular output of bounding boxes and classes turns the two-stage architecture of Faster R-CNN from object detection into instance segmentation.

Two years after Mask R-CNN, in 2019, YOLACT (You Only Look at Coefficients) [7] emerged as the first successful example of real-time instance segmentation, achieving state-of-the-art performance while keeping more than 30 fps. YOLACT introduces a novel modification to the conventional architectures found in YOLO models. Briefly, the modifications made include adding a "Protonet", which takes as input highest resolution feature maps from a PANet structure to capture detailed object information. This Protonet effectively acts as an FCN with deconvolutional [51] layers for upscaling and outputs $k$ different feature maps in the final layer, called "prototypes". These prototypes are key to constructing detailed instance-specific segmentation masks. In addition to the regular YOLO head, which predicts a bounding box, objectness score, and class scores for each anchor, $k$ mask scores are also predicted. To predict masks for the objects that survive NMS, a linear combination of the prototypes is taken, using the predicted mask scores as coefficients. Then, the predicted bounding boxes, which are each associated with a set of $k$ mask scores, are used to crop out a linear combination of prototypes. These cropped areas are subsequently thresholded[13] to produce a binary mask of pixels. The architecture used in YOLACT is illustrated in Figure 2.7.

13. By "thresholded", we mean setting all values below a certain threshold to 0, and all values above the threshold to 1.

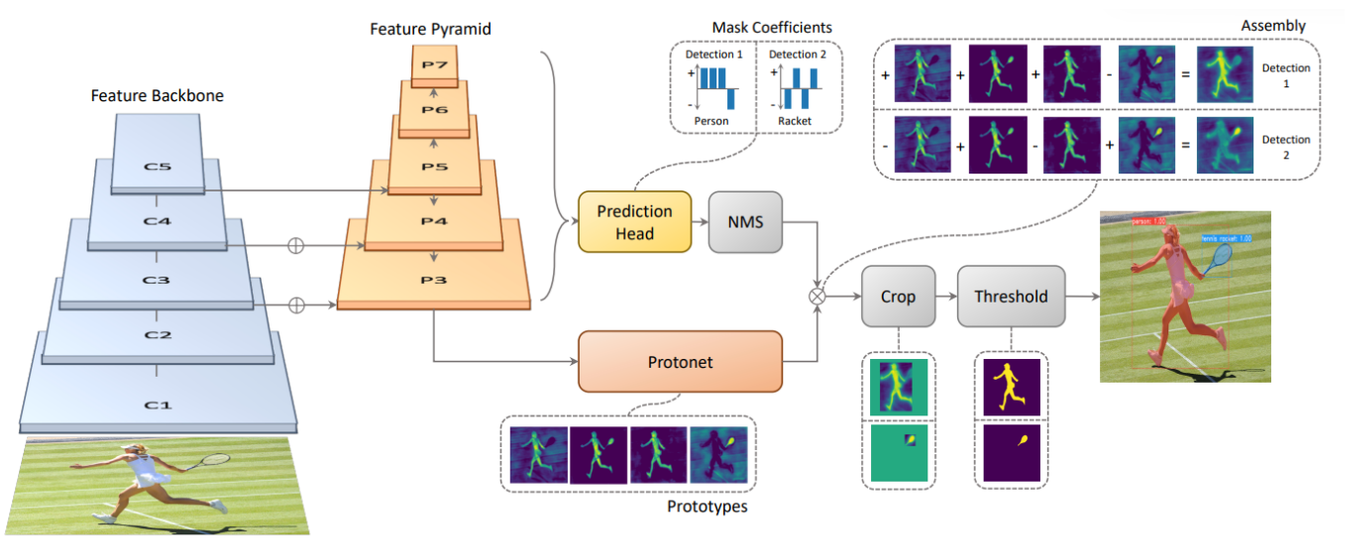



**Figure 2.7:** Caption and image reproduced from [7]. **YOLACT Architecture**: Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and $k = 4$ in this example. We base this architecture off of RetinaNet [47] using ResNet-101+FPN.

# 3

# Method

## 3.1 Why YOLOv5

Choosing a model for an instance segmentation problem is hard, with multiple factors influencing the final selection. Perhaps the two most important factors are inference speed and accuracy, the latter often gauged using general datasets such the COCO dataset. Ideally, every model architecture and every possible hyperparameter configuration would be tested to find the optimal setup for the specific problem. However, this approach is impractical, though testing a few different architectures and approaches is feasible. Thus, some criteria must be established to help the prioritization of models.

The three main criteria for model selection are speed, accuracy, and ease of implementation and documentation. Although speed is not a critical requirement for the automated flare detection program at present, it is anticipated to be important in the future. Hence, two-stage methods like Faster R-CNN are deprioritized due to their typically slower inference times. Similarly, we also reject transformer-based methods due to their data hungry nature, which we believe we can not accommodate at present time. As a result, we are left with considering one-stage methods.

Among one-stage methods, the YOLO series stands out. YOLO models are known for their balance of high accuracy and fast inference times. Additionally, their conceptual simplicity makes for easier implementation and debugging. The choice narrows down to the newer YOLO models: YOLOv5 (2020) [36],

YOLOv6 (2022) [43], YOLOv7 (2022) [79], and YOLOv8 (2023) [37]. While YOLOv6 and YOLOv7 have accompanying papers, YOLOv5 and YOLOv8 do not. However, despite lacking formal papers, they have active GitHub repositories providing plenty of information. This is particularly true for YOLOv5, which, as the oldest, has the most extensive repository, a large community, and numerous contributors. The vast amount of community questions and answers surrounding the details of YOLOv5 also helps in understanding implementation details. Furthermore, YOLOv5 offers five pretrained models, along with some code partially implementing model ensembling. This is considered a big plus as it is of interest to explore how ensemble methods perform on the problem. Therefore, with the preceding in mind, YOLOv5 is selected as the model of choice for this project.

The performance on the COCO dataset is the benchmark of choice for YOLO models. While the performance on other datasets is interesting, optimizing performance on the COCO dataset is seen as most crucial. Consequently, each new iteration of YOLO typically boasts improved performance on the COCO dataset compared to its predecessors. Although a model's performance on the COCO dataset is a strong indicator of its general capabilities, it is not always true that a model performing better on the COCO dataset will also outperform on other datasets. One possible way to gauge a model's general capabilities is to test its performance across various datasets and average the results. This is the idea behind the the Roboflow-100 [11] dataset. Curated from over 90,000 public datasets with 60 million public images on the Roboflow Universe web application [17], Roboflow-100 consists of 100 unique datasets across 7 imagery domains, totaling 224,714 images and 805 class labels. This dataset is far more diverse than the COCO dataset, which includes only 80 classes. The paper presenting the dataset also compares the average performance of YOLOv5 and YOLOv7 on it [11] . Interestingly, YOLOv5 ends up having, on average, higher accuracy than YOLOv7. This shows that performance discrepancies on the COCO dataset do not necessarily hold in general, and further solidifies the choice of YOLOv5 as the model of choice.

## 3.2 YOLOv5

### 3.2.1 YOLOv5 Specifics

#### Model Architecture

The model architecture of YOLOv5 does not depart significantly from that of YOLOv4. The architecture is built using a backbone, neck, and head, which is a common strategy in building object detectors. With this three part system,

one can independently swap out any of the parts to create new models.

The backbone portion serves as a feature extractor. The earlier layers of the backbone contain feature maps with rich spatial information and higher resolution, while the later layers have lower resolution and are high in semantic information. The neck portion uses a modified PANnet structure and takes as input feature maps from varying layers, which helps to gather both high-level semantic and high-resolution spatially rich features. The neck structure not only refines the features before they are fed into the head part, but also provides shorter gradient paths for the earlier layers, which helps optimization. Finally, the head part processes the neck output and applies a single convolution operation with no activation function to predict the final tensors containing values that represent the bounding boxes, classes, and objectness scores. Assuming the input image is of size $640 \times 640$, the output from the head(s) will be three tensors with dimensions of $80 \times 80$, $40 \times 40$, and $20 \times 20$. To modify YOLOv5 for an instance segmentation method, we incorporate the changes proposed in YOLACT, as illustrated in Figure 2.7. Specifically, we add a Protonet to the feature map in the neck with the highest resolution and allow the heads to predict the mask scores associated with the Prototype feature maps.

The three parts of YOLOv5 are composed of smaller blocks of operations. The blocks used in YOLOv5 are the Conv, C3, and SPPF blocks. The interaction between the blocks and the three part system is illustrated in Figure 3.1. Also shown in the figure is the Concat block, meaning concatenation of feature maps, and the Upsample block, which performs a 2x upsampling using nearest neighbor interpolation. Throughout the network, YOLOv5 uses the SiLU activation function and before every convolutional operation in the model, the image is padded with $(n-1)/2$ zeros, where $n$ is the filter size used.

The blocks used are detailed as follows:

- **Conv Block:** The Conv block consists of a convolutional operation followed by batch normalization and SiLU activation. Each Conv block in Figure 3.1 followed by a downward arrow signifies that the convolutional operation is done using filter sizes of $3 \times 3$ with strides of 2[1]. This holds for every Conv block except for the very first one in the model, which uses a $6 \times 6$ filter. Each Conv block followed by an upward arrow is done using $1 \times 1$ convolutions with a stride of 1.

- **C3 Block:** The C3 blocks use cross-stage partial connections [80], which

1. Stride refers to the step size used when applying the filter over an image. A stride of 2 means that the filter is applied to every other pixel of the image, effectively reducing the width and height of the output by a factor of 2.

reduce the computational load while providing a regularizing effect. The C3 block involves taking half the input feature maps and passing them through a Conv block, while the other half is processed multiple times by either of two sub-blocks: BottleNeck 1 or BottleNeck 2. The BottleNeck blocks are constructed using the Conv blocks, with the only difference between the two sub-blocks being the inclusion of a skip connection branch. Figure 3.2 illustrates the sub-blocks and their interaction within the C3 block.

- **SPPF Block:** The SPPF block uses a modified version of the SPP layer [31]. Briefly, the SPPF block applies multiple max pooling operations [42] to increase the receptive field of the final neurons in the backbone. This block adds very little extra computation needed while increasing accuracy. Figure 3.2 illustrates the composition of the SPPF block.

The YOLOv5 architecture can be scaled up or down using Figure 3.1 as a reference. For instance, scaling up the backbone is possible by using more BottleNeck blocks in the C3 blocks. Additionally, one can add more C3 and Conv blocks throughout the network, increase the number of connections from the backbone to the neck, add more output heads, and so on. Finally, the network can be scaled by varying the number of convolutional filters in a given layer.

### Model Output

The YOLOv5 heads output tensors of shape $n_i \times n_i \times a_i$, where each element contains a list of $5 + n_{cls} + k$ values. Here, $i$ is the tensor index ($i = 1, 2, 3$), $a_i$ is the number of predefined anchor boxes for tensor $i$, $n_{cls}$ is the total number of classes, $k$ is the number of prototypes, and $n_i \in [n/32, n/16, n/8]$, where $n$ is the width and height of the input image. For the models used in this problem, we have $n_{cls} = 1$, $k = 32$, $a_i = 3$ and $n = 320$. Note that even though the input images in this problem are of width 320 and height 240, they are zero-padded to be of shape $320 \times 320$ during training. With these settings, the YOLOv5 model will outputs 25,200 bounding boxes and segmentation masks, with most of these predictions being pruned away by NMS.

Each value in the predicted lists must go through a final set of functions before they are meaningful. Focusing on a single list in the tensor, four of the values, $t_x, t_y, t_w$, and $t_h$, are converted to $b_x, b_y, b_w$, and $b_h$, where the values now represent the center $(x, y)$ coordinates, width, and height of a bounding box via the equations:

$$b_x = (2\sigma(t_x) - 0.5) + c_x \tag{3.1}$$

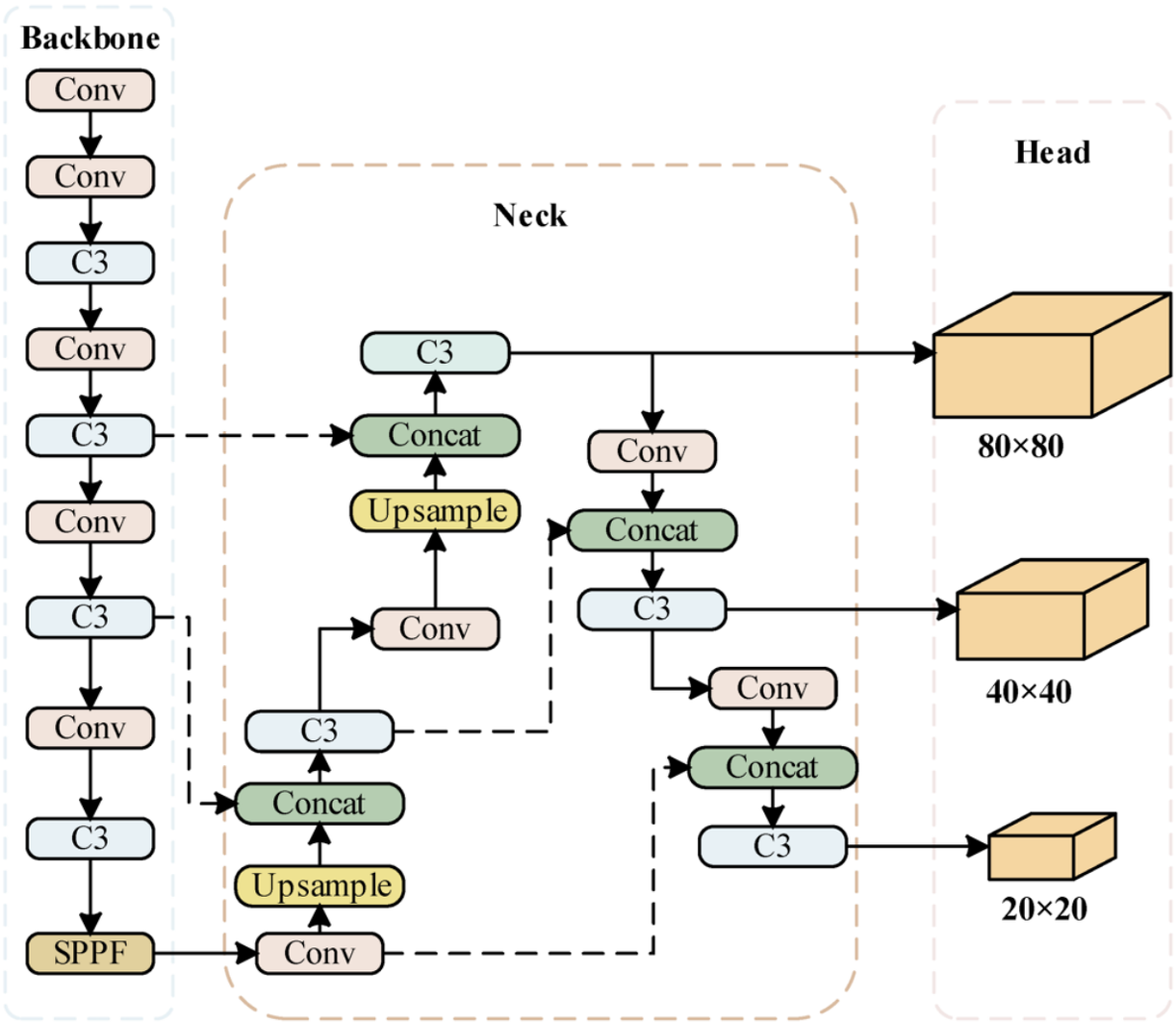


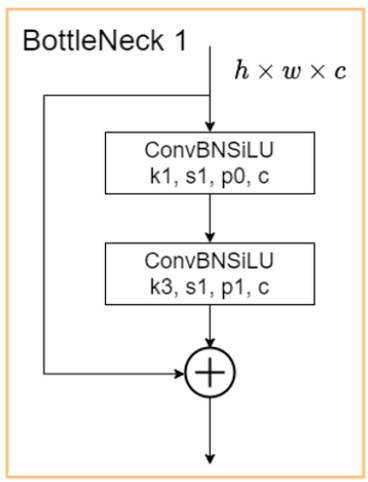

**Figure 3.1:** Illustration of the YOLOv5 architecture.

$$b_y = (2\sigma(t_y) - 0.5) + c_y \tag{3.2}$$

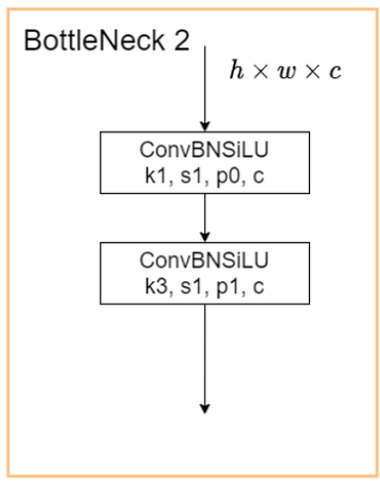
$$b_w = 4p_w\sigma(t_w)^2 \tag{3.3}$$

$$b_h = 4p_h\sigma(t_h)^2. \tag{3.4}$$

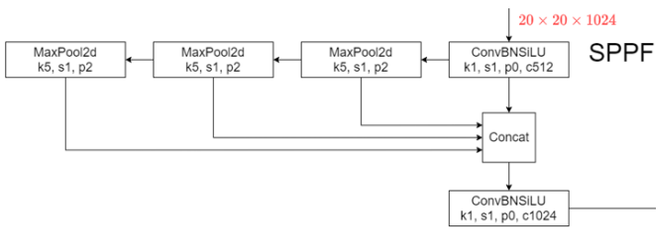
Here, $\sigma$ is the sigmoid function, $(c_x, c_y)$ represents a pixel position on the original image mapped to the list, and $(p_w, p_h)$ are the width and height of the predefined anchor box mapped to the list. From this point forward, we occasionally refer to the predefined anchor boxes as Anchor Templates (AT). Table 3.1 shows the width and height of all the anchor boxes for the 3 output tensors. The bounding box output is illustrated in Figure 3.3. In the illustration,



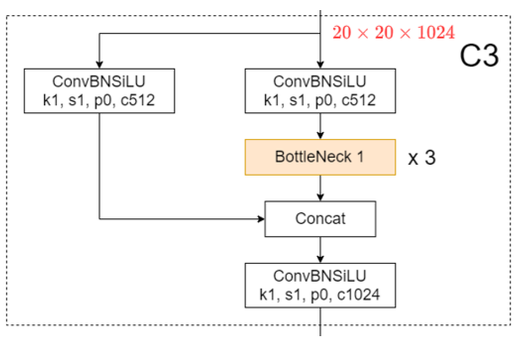

**(a)** BottleNeck 1 block, used in the C3 blocks in the backbone portion of YOLOv5.




**(b)** BottleNeck 2 block, used in the C3 blocks in the neck portion of YOLOv5.




**(c)** SPPF block, used in the C3 blocks in the backbone portion of YOLOv5.




**(d)** C3 block with BottleNeck 1, used in the backbone portion.

**Figure 3.2:** Images reproduced from [36]. The blocks take tensors of size $h \times w \times c$ as input. In the illustrations, $k$, $s$, and $p$ stand for kernel (convolutional filter), stride, and (zero) padding, respectively. For example, the tuple $[k3, s1, p1, c512]$ indicates that the convolution+batch normalization+SiLU operation uses filter sizes of $3 \times 3$ with a stride of 1 and zero padding size of 1, and the number of output channels is 512.

we assume that each side of the grid cell is of length 1 and that the point $(0, 0)$ is in the top-left corner. Note that the equations used in Figure 3.3 are different from those defined here. The equations used here allow the center of the predicted bounding box to hit the center of all the adjacent grid cells. Additionally, the width and height of the predicted bounding boxes are bounded from above to be 4 times the size of the predefined anchor box. In previous versions of YOLO, the predicted size of the bounding box was not bounded, which occasionally led to runaway gradients, instabilities, NaN (Not a Number)

valued loss functions, and even failure to converge.

| | Anchor box 1 | Anchor box 2 | Anchor box 3 |
|---|---|---|---|
| **Tensor/head 1** | (116, 90) | (156, 198) | (373, 326) |
| **Tensor/head 2** | (30, 61) | (62, 45) | (59, 119) |
| **Tensor/head 3** | (10, 13) | (16, 30) | (33, 23) |

**Table 3.1:** The width and height in terms of pixels of the anchor box templates used. Each tensor/head has 3 anchors associated with it.

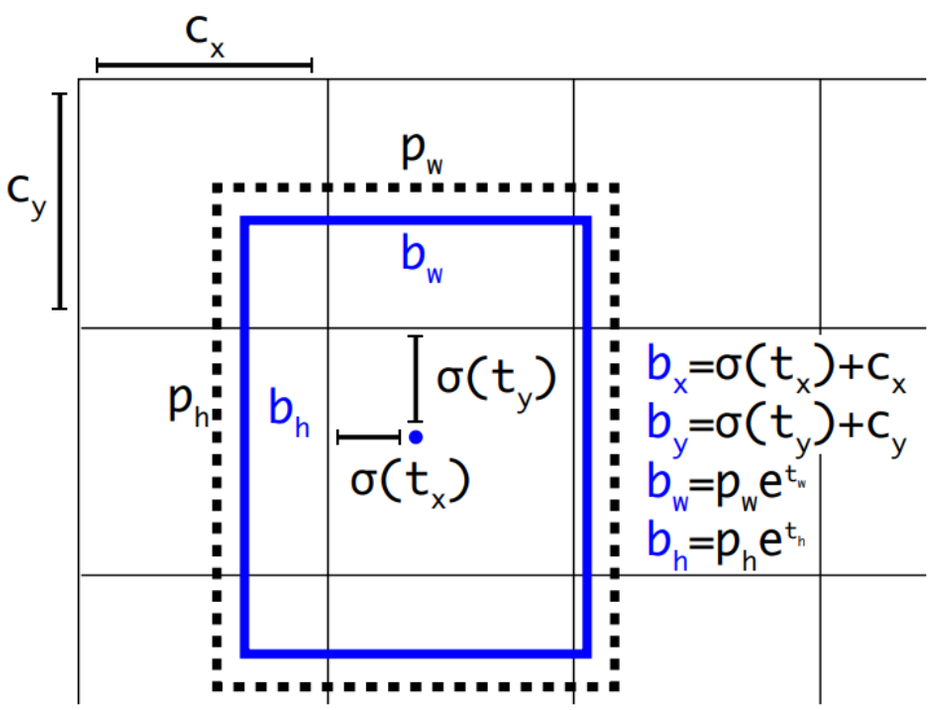



**Figure 3.3:** Image and caption reproduced from [59]. Assume that each grid cell has an area of 1 and note how the equations for $b_x$, $b_y$, $b_w$, and $b_h$ have changed from those in equations (3.1) to (3.4). **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

The values in the list corresponding to the objectness and class scores are fed through the sigmoid function to be bounded between 0 and 1. Finally, the 32 mask scores associated with the prototypes are fed through the tanh function to be bounded between -1 and 1. This approach allows the model to learn to subtract or add specific ratios of the prototype feature maps more easily, a feature essential for learning to produce the binary masks associated with a bounding box, which is harder than predicting the bounding box itself.

### Assigning Correct Ground Truths

How ground truth objects are assigned to the outputs of the model is critical for training efficiency and model accuracy. YOLOv5 differs from previous iterations of YOLO in that it can assign 3 times as many anchor boxes to the same ground truth. This change is facilitated by the fact that the center coordinates of the predicted bounding box in equations (3.1) and (3.2) range from $(c_x, c_y) \pm (1.5, 1.5)$, whereas in previous YOLO models, the range was $(c_x, c_y) \pm (1, 1)$. The extra ±0.5 means that the center of predicted bounding boxes can reach the center of all adjacent grid cells, instead of just their respective grid cell boundaries.

Ground truth objects are assigned to anchor templates according to the following equations:

$$r_w = w_{gt}/w_{at} \tag{3.5}$$

$$r_h = h_{gt}/h_{at} \tag{3.6}$$

$$r_w^{max} = \max(r_w, 1/r_w) \tag{3.7}$$

$$r_h^{max} = \max(r_h, 1/r_h) \tag{3.8}$$

$$r^{max} = \max(r_w^{max}, r_h^{max}) \tag{3.9}$$

$$r^{max} < \text{anchor}_t = 4. \tag{3.10}$$

Here, $w_{gt}, h_{gt}, w_{at}$, and $h_{at}$ represent the width and height of the ground truth boxes and anchor templates, respectively. Thus, an anchor template is considered a match for a ground truth box if its width and height are each no more than 4 times and no less than 0.25 of the width and height of the ground truth box. This is illustrated in Figure 3.4. YOLOv5 allows the assignment of three different grid cells, each with multiple anchor templates, to a single ground truth object. The assignment is based on which quadrant of the grid cell the center of the ground truth bounding box lands in, as illustrated in Figure 3.5.

### Loss Function

The total loss function is the sum of the individual losses associated with the class, bounding box, objectness, and segmentation mask predictions. If a specific anchor template is assigned to a ground truth object, then the class, bounding box, objectness, and segmentation mask predictions associated with that anchor will contribute to the loss. In cases where a specific anchor template is not assigned to a ground truth object, only the objectness score will contribute to the total loss.

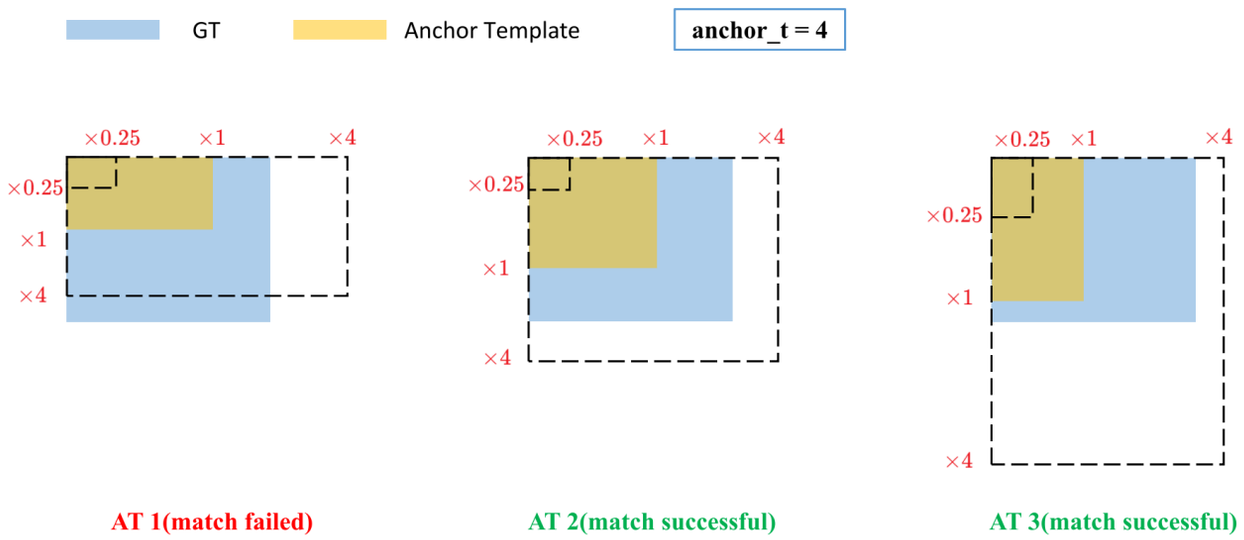



**Figure 3.4:** Image reproduced from [36]. Illustration of how ground truth objects are assigned to anchor templates based on the ratios of height and width.

More specifically, the losses associated with predicted class scores, objectness scores, and segmentation masks use the Binary Cross-Entropy (BCE) loss:

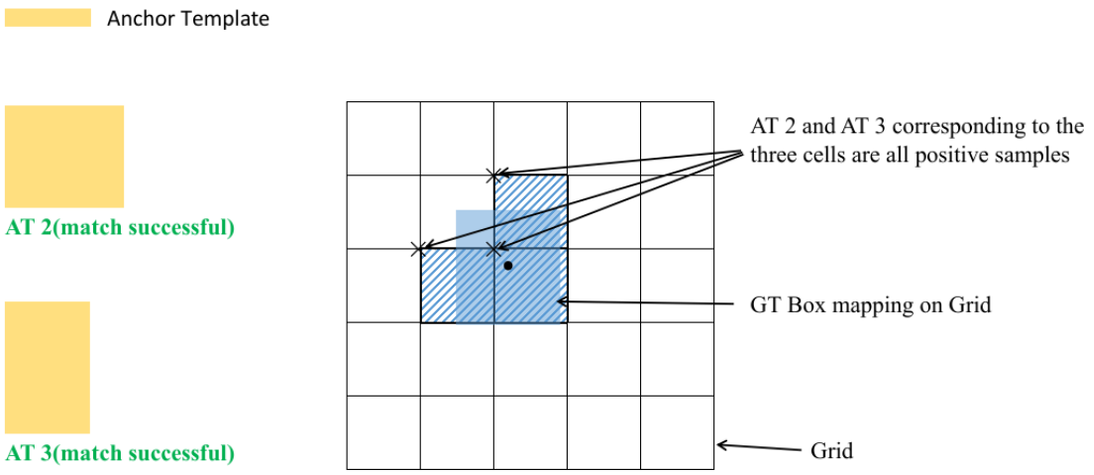
$$\text{BCE}(p, y) = -\delta\big[y \log(p) + (1 - y) \log(1 - p)\big]. \tag{3.11}$$

Here, $p$ is a predicted probability, $y$ is the ground truth label, and $\delta$ takes on a value of either 0 or 1, depending on whether the anchor template was assigned to a ground truth. For the segmentation loss, we crop the predicted and ground truth segmentation masks based on the ground truth bounding box and then apply the BCE loss per pixel between the remaining pixels. The labels $y$ in this case are either 0 or 1, depending on the pixel values in the binary ground truth mask. In cases where the anchor template is not assigned to a ground truth object, we set $y = 0$ and $\delta = 1$ for the associated objectness score. The different cases are given in Table 3.2.

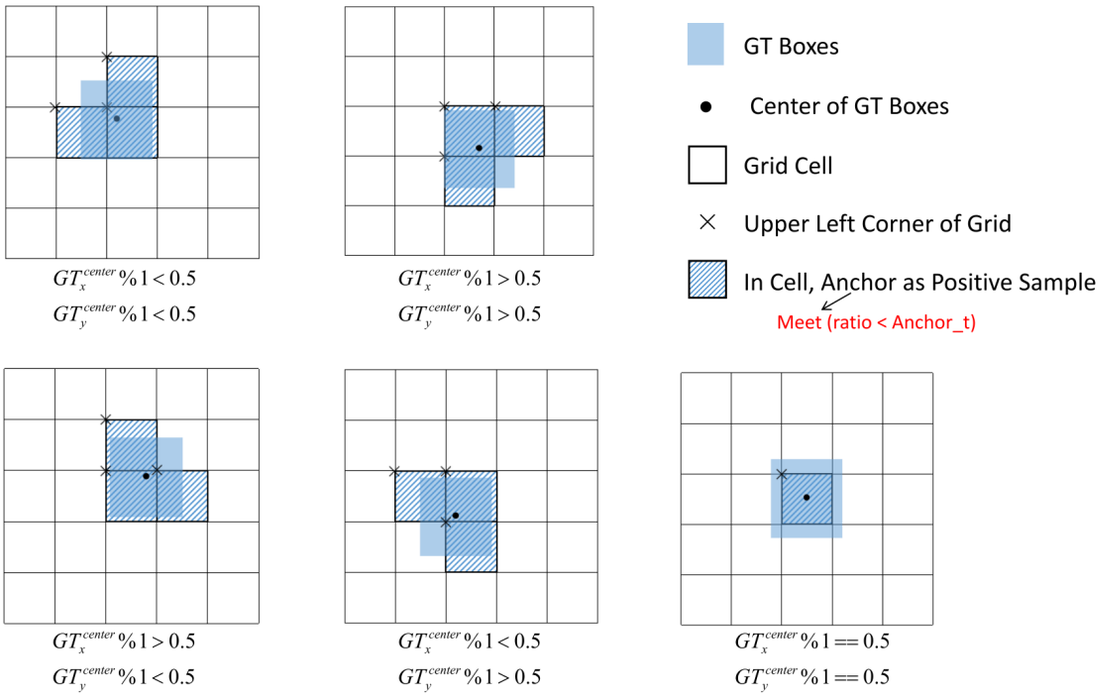
The loss associated with an assigned predicted bounding box is given by the CIoU loss:

$$\text{CIoU} = (1 - \text{IoU}) + \frac{q^2(b, b_{gt})}{c^2} + \alpha v. \tag{3.12}$$

Here, IoU is the intersection over union between the predicted bounding box $b$ and the ground truth box $b_{gt}$, $q(b, b_{gt})$ is the Euclidean distance between the box centers, and $c$ is the diagonal length of the smallest box that can cover both $b$ and $b_{gt}$. The last term in the CIoU loss, $v$, measures the consistency of




**(a)** Anchor templates 2 and 3 from 3 different grids are assigned to one ground truth object. Thus, 6 anchor templates in total are associated with this ground truth object.




**(b)** Each ground truth object gets assigned to 3 grid cells, except in the case where the center of the ground truth object is perfectly in the center of a grid cell, in which case only that one grid cell is assigned to the ground truth object.

**Figure 3.5:** Images (a) and (b) reproduced from [36]. Illustration on how ground truth objects are assigned to anchor templates during tra

aspect ratios:

$$v = \frac{4}{\pi^2}\left(\arctan\frac{w_{gt}}{h_{gt}} - \arctan\frac{w}{h}\right)^2, \tag{3.13}$$

| Type of Output | Assigned | | Not Assigned | |
|---|---|---|---|---|
| | y | $\delta$ | y | $\delta$ |
| Class Score | 1 | 1 | 0 or 1 | 0 |
| Objectness Score | IoU or 1 | 1 | 0 | 1 |
| Segmentation Mask Pixels | 1 | 1 | 0 or 1 | 0 |

**Table 3.2:** The different ground truth labels the outputs can have based on whether they were assigned to a ground truth object. For the objectness scores, we use either the IoU or 1 as the label, depending on how we want to train the model. IoU is a dynamic label that takes on the value of the IoU between the predicted bounding box and the ground truth bounding box.

where $(w_{gt}, h_{gt})$ and $(w, h)$ are the width and height of the ground truth and predicted bounding box, respectively. The trade-off parameter $\alpha$ is given by:

$$\alpha = \frac{v}{(1 - \text{IoU}) + v}. \tag{3.14}$$

The first term in equation (3.12) trains the model to reduce the overlap area, the second term trains the model to reduce the distance between box centers, and the last term, $\alpha v$, trains the model to maintain correct aspect ratios between boxes.

Now, for update iteration $i$ with some batch size $s$, the different loss terms can be written as sums over the number of times a type of output received a ground truth assignment:

$$\text{Loss}_{cls}(i) = \sum_{i_1} \alpha_{i_1} \text{BCE}_{cls}(p_{i_1}, y_{i_1}), \tag{3.15}$$

$$\text{Loss}_{o}(i) = \sum_{i_2} \alpha_{i_2} \text{BCE}_{o}(p_{i_2}, y_{i_2}), \tag{3.16}$$

$$\text{Loss}_{mask}(i) = \sum_{i_3} \alpha_{i_3} \text{BCE}_{mask}(p_{i_3}, y_{i_3}), \tag{3.17}$$

$$\text{Loss}_{box}(i) = \sum_{i_4} \alpha_{i_4} \text{CIoU}_{box}(i_4). \tag{3.18}$$

The weights $[\alpha_{i_1}, \alpha_{i_2}, \alpha_{i_3}, \alpha_{i_4}]$ are computed based on the batch size, the number of terms in the respective sum, which head computes the output, and a set of constants derived empirically on the COCO dataset. The weights are crucial as they determine the relative influence between the four objectives defined by the loss terms. By increasing the weight of a loss term, the model tends to place extra emphasis on learning how to decrease it. If the weight on a loss term becomes too great, the model may neglect learning the objectives set out by the rest of the loss terms in favor of decreasing it. Because of the numerous

ways the weights can change, they will not be elaborated further on, but details can be found in the YOLOv5 code repository [36].

Finally, the full loss for update iteration $i$ is given by:

$$\text{Loss}(i) = \text{Loss}_{cls}(i) + \text{Loss}_{o}(i) + \text{Loss}_{mask}(i) + \text{Loss}_{box}(i). \tag{3.19}$$

### 3.2.2 Training

All models are trained using the default hyperparameter setup and training scheme defined in the YOLOv5 repository [36]. The only hyperparameters that have been changed are the epoch number, which influences training by affecting the learning rate, and the batch size. The numbers of epochs and batch sizes used will be presented alongside the performance of the models in the next chapter.

During training, an exponential moving average of the parameters is kept, which will be used for inference after training is complete. Multi-scale training is used as a regularization technique. This technique involves randomly scaling the training images with bilinear interpolation to be within a range of 0.5 to 1.5 times the original image size. The different scales are drawn from a uniform distribution. Early stopping is implemented with a patience parameter $p = 100$ epochs.

The optimizer used is SGD with Nesterov momentum of 0.937. The learning rate is set according to the linear schedule scheme in equation (2.27) with $(\epsilon_0, \epsilon_\tau) = (0.01, 0.0001)$, and where $\tau$ is set to be the epoch number. The loss is modified as in equation (2.33) by adding an $L^2$ norm weight decay term to the final loss. Weight decay is applied with $\beta = 0.0005$ to all parameters in the network except for those associated with batch normalization and the bias terms. A gradual warm-up schedule is used for every update iteration until 3 epochs are completed. This warm-up scheduling is done using linear interpolation between 0.1 and 0.01 for the bias term learning rate and between 0 and 0.01 for the learning rate of other parameters. Additionally, momentum is warmed up from 0.8 to 0.937, also using linear interpolation.

As image examples are drawn into batches during training, there is a chance that augmentation techniques will be applied to them. The possible augmentations include changes to image hue, saturation, and value (HSV modifications); image translation; horizontal flipping; changes to image scale; image blurring; converting the image to grayscale; CLAHE [68]; and mosaic data augmentation [5]. Mosaic data augmentation involves selecting 4 images at random, resizing

and/or cropping them before combining them into a single image. One of the benefits of mosaic augmentation is that it reduces the variance in batch normalization statistics due to computing the statistics on 4 different images each time (normalization is done per feature map in CNNs) [5]. The different probabilities associated with the data augmentation techniques are given in Table 3.3. The intensities of each data augmentation are set to the default values and can be found in [36]. An example of a batch of training examples after adding data augmentation is shown in Figure 3.6.

| **Augmentation** | **Probability** |
|---|---|
| Hue change | 0.015 |
| Saturation change | 0.7 |
| Value change | 0.4 |
| Image translation | 0.1 |
| Horizontal flipping | 0.5 |
| Image scale change | 0.5 |
| Image blurring | 0.01 |
| Image to grayscale | 0.01 |
| CLAHE [68] | 0.01 |
| Mosaic | 1 |

**Table 3.3:** Probability of applying different augmentations to training examples as they are selected into a batch.

### 3.2.3 Implementation

The code used for this project builds upon the YOLOv5 GitHub repository [36]. All experiments were conducted on Google Colaboratory Pro, where the preferred GPU for training and inference was Nvidia's Tesla T4 GPU.

#### Base Models

In the repository, there are 5 pretrained models on the COCO dataset for instance segmentation: YOLOv5N, YOLOv5S, YOLOv5M, YOLOv5L, and YOLOv5X. The starting architecture used for each model is illustrated in Figure 3.1. However, the number of repeats for each block (in the backbone) and the number of channels in a layer are scaled according to individual depth and width multiple factors, respectively [73, 36]. The layer count, number of trainable parameters, and the amount of FLOPs required for processing a single image are presented in Table 3.4.

We can create 5 more unique models by taking the architectures used and

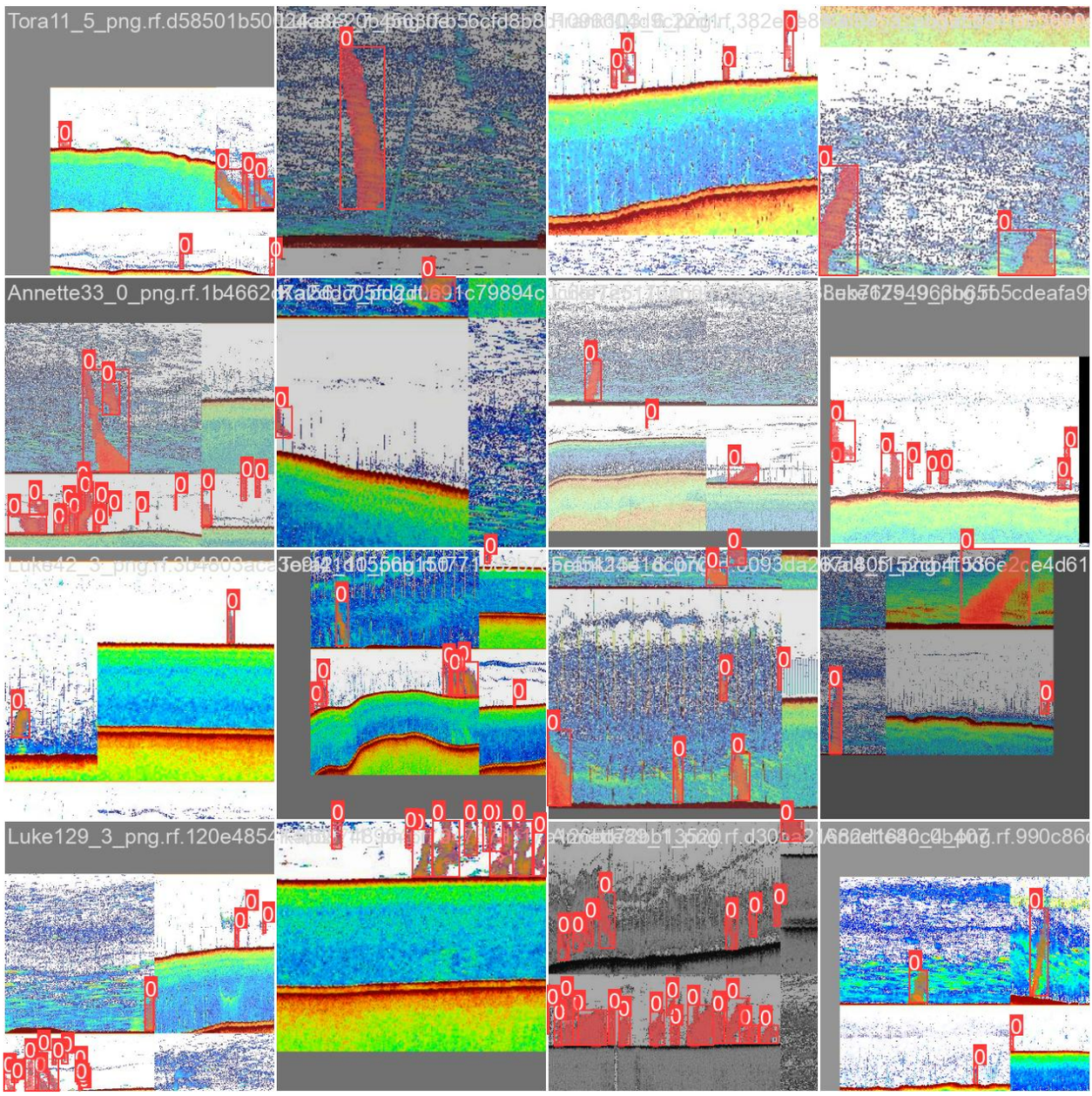



**Figure 3.6:** 16 training images in a batch after applying data augmentation. Note how the mosaic data augmentation, as shown by the fusion of 4 cropped images, is applied to every example in the batch.

replacing the pretrained weights with He initialized weights drawn from a uniform distribution. Furthermore, by using either of the two targets for the objectness score in Table 3.2, we can obtain an additional 1010 unique models. Thus, in total, we have 20 unique models that can serve as the voters in the ensemble model.

| Model | Parameters (M) | Layer Count | FLOPs (B) |
|---|---|---|---|
| YOLOv5N | 1.89 | 225 | 6.9 |
| YOLOv5S | 7.41 | 225 | 25.9 |
| YOLOv5M | 21.7 | 302 | 70.3 |
| YOLOv5L | 47.5 | 379 | 147.0 |
| YOLOv5X | 88.3 | 456 | 264.9 |

**Table 3.4:** Parameter, layer count, and FLOPs comparison of YOLOv5 model variants. Parameters are presented in terms of millions (M) and FLOPs in billions (B). The number of FLOPs needed assumes input images of size $320 \times 320 \times 3$.

### Ensemble Models

We will use either NMS or Weighted Box Fusion (WBF) [70], adapted for instance segmentation, to fuse together the outputs of the individual models. WBF is a method for fusing multiple bounding boxes. It works by assigning boxes to clusters and then computing a fused box for each cluster based on the average position and shape of the individual boxes, weighted by their objectness score. New boxes are added to clusters if their IoU with the fused box for that cluster is above some threshold. Once a new box is added to a cluster, the fused box is recalculated. These steps are repeated until convergence. The final objectness scores for the fused boxes are computed as the averages of the objectness scores within a cluster[2]. The details of the algorithm can be found in [70]. A comparison of how WBF fuses outputs compared to NMS is illustrated in Figure 3.7.

When using NMS, we will pool together all the predicted bounding boxes from all the models for an image and apply NMS as normal to obtain the final predictions. Any time NMS is applied in this project, the hyperparameters used will be 0.0001 and 0.45 for the pruning based on objectness score and IoU, respectively.

WBF was designed to fuse bounding boxes, but we can modify it for instance segmentation. First, we apply NMS to the outputs of the individual models before using WBF on the remaining boxes with an IoU threshold of 0.5 to compute which predictions cluster together. Next, each predicted binary mask in a cluster will be weighted by its objectness score before all the masks are fused by summing them together. The pixels in the fused mask are normalized to values between 0 and 1 by dividing them by the maximum pixel value. Pixel values closer to 0 should be removed, as they are the result of low objectness scores and disagreements between the masks in a cluster. By setting all pixels

2. If the number of boxes in a cluster is fewer than the total number of models used, then the average will be scaled down accordingly.

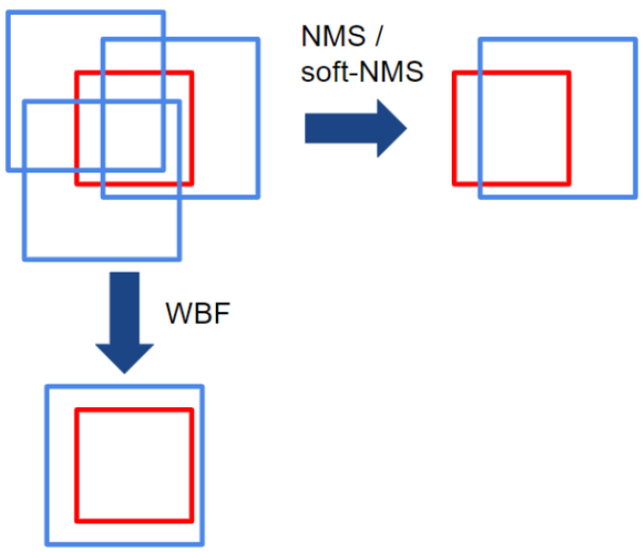
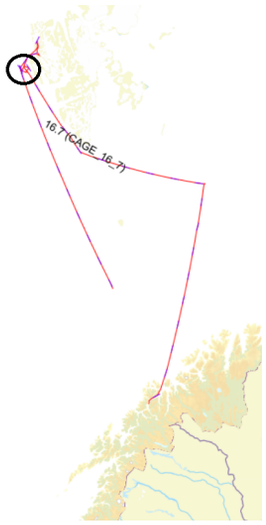



**Figure 3.7:** Image and caption reproduced from [70]. Schematic illustration of NMS/soft-NMS vs. WBF outcomes for an ensemble of inaccurate predictions. Blue – different models' predictions, red – ground truth.

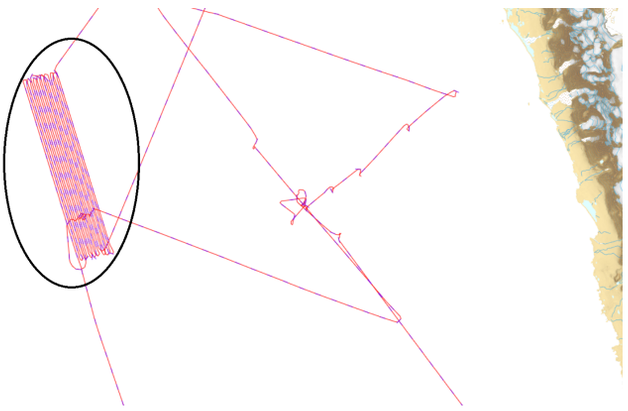
below some threshold to 0 and all above to 1, the final fused mask is created. The threshold used for all ensemble models in this project was the fused objectness score divided by 2. A low flat value such as 0.2 also works, but through experiments the more dynamic threshold based on objectness score was found to work best. Finally, every time a mask ends up with a fused objectness score less than 0.1 it is always removed.

## 3.3 Dataset

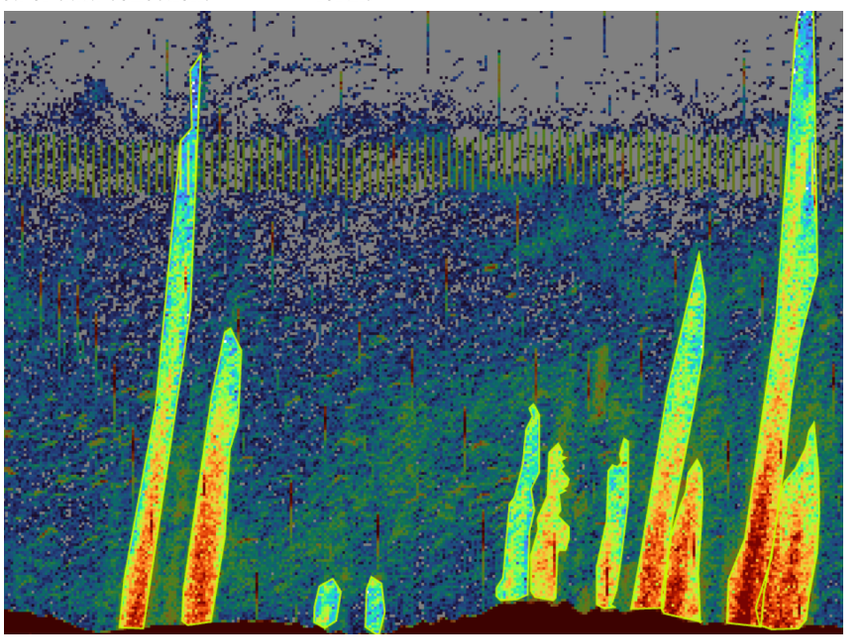
The data were collected from four different cruises to the western Svalbard margin, with the first trip in 2015 and the last in 2017. An example of a trip route is shown in Figure 3.8. The data start as raw echogram files before being processed into image files. We obtain a total of 4,057 grayscale images, each with a width of 320 pixels and a height of 240 pixels. These images are converted to RGB for visual clarity by mapping the grayscale values appropriately. Out of the 4,057 images, 5,142 objects identified as flares were segmented across 1,414 images. Of these, 1,246 images are dedicated to the training set, 138 to the validation set, and 30 to the test set. The selection of images for the various sets was done randomly.




**(a)** Circled in black is the area of data collection.



**(b)** The boat covers a rectangular area by moving back and forth.



**(c)** Example of gas flare segmentations in an image from the training dataset.

**Figure 3.8:** The cruise path shown in (a) and (b) is from the third cruise to Svalbard, which lasted from 16/10/2016 to 25/10/2016.

### 3.3.1 Trained Non-Expert Labeling

Training, validation, and test dataset were created by the author, who can be considered a trained non-expert. To train the non-expert, Expert 1 segmented the first 100 images. These images were carefully studied and used as reference

for the non-expert. The non-expert then segmented the rest of the dataset, having Expert 1 available for discussion whenever the non-expert were in a quandary on how to evaluate a feature in the echogram. Only one class was used for the problem: gas flare or not. Consequently, the labels can be considered to indicate 100% confidence that what is segmented is a true gas flare.

### 3.3.2 Expert Labeling

We recruit six experts to label the same 30 images as in the non-expert's test set. However, instead of giving true or false labels, the experts indicated a confidence score on what was segmented. The possible confidence levels ranged from 0.1 to 1 in increments of 0.1, totaling 10 possible confidences. These labels are more flexible as they can be converted to true or false labels by thresholding and are also more informative compared to binary labeling, which tends to group various thresholds of confidence into one (i.e., grouping together 0.8, 0.9, and 1 confidence all as true).

## 3.4 Evaluation Metrics

To gauge the performance of the model(s), we need some sort of evaluation metric. The metric commonly used for instance segmentation is Mean Average Precision (mAP) [74], which will also be used here. Additionally, we adapt the Brier score [8] for use with instance segmentation due to the continuous nature of labeling by the experts, that is, the ability to place confidence scores between 0 and 1. It is not only of interest to gauge how the model performs on the expert data but also how the experts evaluate each other. To benchmark the experts against themselves, we will consider each expert's ground truth set as the prediction set and then evaluate it against the sets of the remaining experts using mAP and Brier scores. However, we cannot directly use the mAP50 metric on the test set created by the experts because of the soft labels they provide (values between 0 and 1). We worked around this problem by thresholding the experts' predictions into true or false labels. For instance, we can remove all predictions below 0.8, set all above 0.8 to true, and then compute the mAP as usual on the resulting test set.

### 3.4.1 Mean Average Precision

To compute the Average Precision (AP), consider only the ground truths of one class and set IoU and class score thresholds to values such as 0.5 and $\alpha$

respectively. Next, remove all predictions with a class score less than $\alpha$ for the specified class. Compute the IoU between the remaining predictions and the ground truths. A prediction is considered a True Positive (TP) if IoU $\geq 0.5$ and a False Positive (FP) if IoU $< 0.5$. The total number of False Negatives (FN) is defined as the total number of ground truths minus the number of TPs. From these, we can compute Precision (P) and Recall (R) as:

$$P = \frac{TP}{FP + TP} \tag{3.20}$$

$$R = \frac{TP}{FN + TP}. \tag{3.21}$$

Finally, plot $(P, R)$ for $\alpha \in (0, 1]$ and compute the area under the curve to get the AP for one class. Compute the AP for all classes and take the mean to get mAP50, where 50 indicates that an IoU threshold of 0.5 was used. To compute mAP50-95, calculate the mAP between 0.5 and 0.95 in increments of 0.05 and take the average. The mAP50-95 places more emphasis on correct segmentations than mAP50. Thus, we can expect to see lower mAP50-95 than mAP50 scores. Precision and recall are measures of the model's accuracy and its ability to identify ground truths, respectively. There is often a trade-off between the two. For instance, as we increase recall by using more predictions, it is often the case that precision falls because of more FPs. The mAP is a composite measure summarizing the model's performance on both metrics - a mAP100 of 1 would indicate perfect predictions. Both mAP50 and mAP50-95 will be used to evaluate model performances.

### 3.4.2 Brier Score

Another way to benchmark performance on the expert data is to use a metric that allows for comparisons between predictions and soft labels, such as the Brier score. We compute the Brier score by assigning predictions to ground truths if their IoU is greater than a certain threshold, such as 0.5. Then, we compute the squared error between the experts' confidence scores and the model's objectness scores. We only assign the prediction with the highest IoU to a ground truth. This is different than with the mAP calculation where we allow multiple predictions for a ground truth. The Brier score for an IoU threshold of 0.5 is:

$$\text{Brier50} = \sum_{i_1} (g_{i_1} - p_{i_1})^2 + \sum_{i_2} g_{i_2}^2 + \sum_{i_3} p_{i_3}^2. \tag{3.22}$$

Here, $g$ represents ground truth confidence scores, and $p$ are the predicted objectness scores. The first two sums are over the number of ground truths: $i_1$ is the number of ground truths that received a prediction, and $i_2$ is the number of ground truths that did not. The last sum is over all the predictions that

were not assigned to a ground truth and serves to penalize overprediction, similar to precision. We will evaluate both the experts and models Brier50 and Brier50-95.

# 4

# Results and Discussions

In this chapter, we begin by presenting the obtained results. Explanations on how to read the tables and figures that showcase these results are given when they show up. This is followed up by a quantitative and qualitative analysis of the findings, which will conclude the chapter.

## 4.1 Results on Non-Expert Data

First, we will evaluate the base YOLOv5 models on the non-expert dataset using the mAP metric. In total, 25 models were trained with results presented in Tables 4.2 to 4.6. These models were trained using either pretrained weights or weights initialized with He initialization, and with either hard targets or IoU targets for the objectness score, as shown in Table 3.2. By hard targets, we mean labels that are either exactly one or exactly zero. The number of epochs, best epoch, and batch sizes will also be shown in parentheses next to each model. The EMA parameters at the best epoch point are the parameters that are used for evaluating the mAP scores. Additionally, we will denote which models are selected to be part of an ensemble by the number in the superscripted bracket. For example, '**YOLOv5N(500, 280, 160)**$^{(1)}$' indicates that this model was the YOLOv5N model trained with an epoch number of 500, the best stopping point determined by the early stopping algorithm was epoch 280, and that a batch size of 160 was used. Furthermore, this model will be a part of Ensemble1, as indicated by the superscript **(1)**.

Also presented in Tables 4.2 to 4.6 is the performance of the models when ensembled together using either NMS or WBF fusion. This comparison will help determine whether NMS or WBF is the superior fusion choice for this problem. We present the ensembles by first noting which fusion method was used, followed by denoting which base models are part of the ensemble, indicated in parentheses. For instance, '**WBF ensemble(N+L)**' represents the ensemble model formed by YOLOv5N and YOLOv5L, where the fusion of masks is achieved through WBF. Only five combinations of models have been selected for testing, either with NMS or WBF fusion. The total number of possible combinations of models is $\sum_{i=1}^{5} \binom{5}{i} = 31$. While testing all possible combinations and summarizing the performance by averaging could be interesting, this approach has been omitted due to the large amount of training time it would require.

The final ensemble models, which will be used to evaluate performance on the experts' datasets, are shown in Table 4.1. These ensembles all use WBF to fuse predictions together. Ensemble1 and Ensemble2 were created by selecting the models with the best performance on the validation and test sets, respectively. Likewise, Ensemble3 and Ensemble4 consist of the best hard target models, with and without pretrained weights, respectively

| **Set:** | **Validation** | | **Test** | |
|---|---|---|---|---|
| **Metrics:** | **mAP50** | **mAP50-95** | **mAP50** | **mAP50-95** |
| **Ensemble1** | **0.828** | **0.433** | 0.836 | 0.438 |
| **Ensemble2** | 0.808 | 0.42 | **0.882** | **0.457** |
| **Ensemble3** | 0.756 | 0.381 | 0.762 | 0.382 |
| **Ensemble4** | 0.788 | 0.403 | 0.793 | 0.419 |

**Table 4.1:** mAP scores for Ensemble1 to Ensemble4 on the non-expert's validation and test sets. The constituent base models of each ensemble can be found in Tables 4.2 to 4.6. Ensemble1 to Ensemble4 will be further evaluated on the experts' datasets.

## 4.2 Results on Experts' Data using Brier Scores

Next, we will evaluate the four ensembles listed in Table 4.1 on the experts' datasets. In addition to the four ensembles already described, we will also introduce two more ensemble models: Ensemble1B and Ensemble2B. These ensembles are based on Ensemble1 and Ensemble2 but with their fused objectness scores increased (boosted) by 0.1. This adjustment is due to the fact that Ensemble1 and Ensemble2 were created with models trained on IoU targets for the objectness scores. If we interpret the objectness scores as confidence levels, then the models would appear to be perpetually underconfident, as they

| Pretrained and IoU targets | | | | |
|---|---|---|---|---|
| **Set:** | **Validation** | | **Test** | |
| **Metrics:** | **mAP50** | **mAP50-95** | **mAP50** | **mAP50-95** |
| **YOLOv5N(500, 280, 160)**$^{(1)}$ | 0.753 | 0.355 | 0.76 | 0.351 |
| **YOLOv5S(500, 224, 98)**$^{(1)}$ | 0.775 | 0.347 | 0.753 | 0.362 |
| **YOLOv5M(500, 96, 68)**$^{(1)}$ | 0.772 | 0.359 | 0.746 | 0.336 |
| **YOLOv5L(500, 205, 68)**$^{(2)}$ | 0.746 | 0.353 | 0.837 | 0.401 |
| **YOLOv5X(500, 150, 60)**$^{(1)}$ | 0.752 | 0.342 | 0.7 | 0.337 |
| **NMS ensemble(N+L)** | 0.762 | 0.353 | 0.826 | 0.391 |
| **NMS ensemble(X+L)** | 0.772 | 0.357 | 0.765 | 0.375 |
| **NMS ensemble(X+L+M+S)** | 0.783 | 0.353 | 0.79 | 0.38 |
| **NMS ensemble(L+M+S+N)** | 0.781 | 0.353 | 0.8 | 0.38 |
| **NMS ensemble(X+L+M+S+N)** | 0.784 | 0.353 | 0.79 | 0.378 |
| **WBF ensemble(N+L)** | 0.792 | 0.408 | **0.844** | **0.453** |
| **WBF ensemble(X+L)** | 0.792 | 0.395 | 0.81 | 0.42 |
| **WBF ensemble(X+L+M+S)** | 0.814 | 0.416 | 0.818 | 0.427 |
| **WBF ensemble(L+M+S+N)** | 0.813 | 0.428 | 0.799 | 0.429 |
| **WBF ensemble(X+L+M+S+N)** | **0.823** | **0.429** | 0.802 | 0.422 |

**Table 4.2:** mAP scores of the YOLOv5 base and ensemble models on the non-expert's validation and test sets. The base models use pretrained weights from the COCO dataset and are trained using IoU targets for the objectness score. The numbers in parentheses represent the epoch number, best epoch, and batch size, respectively. Models with a superscript of (1) or (2) indicate their inclusion in Ensemble1 or Ensemble2, respectively.

almost never encounter labels above 0.9 during training. Therefore, we have added an extra 0.1 as compensation, to see if a raise in confidence has any impact on the ensembles' performances on the experts' datasets.

We will first evaluate the ensembles on the experts' datasets using the Brier50 and Brier50-95 scores before moving on to the mAP scores in the next subsection. Alongside the ensemble performance, we will also assess the experts on each other's datasets. This assessment is done by considering each expert individually as the predictor and then evaluating their predictions on the datasets of the remaining five experts. Tables 4.7 and 4.9 present the individual results of both the experts and the ensembles. In these tables, the predictors are defined on the rows and the testing sets on the columns. The zeros on the diagonal indicate that the experts have perfect evaluations on their own sets, as expected. Additionally, we observe that the performance of the experts is symmetric, meaning that changing the order of the predictor and the test set does not influence the results. This symmetry results from our method of assigning predictions to ground truths when calculating the Brier scores, specifically by allowing only one prediction per ground truth. Tables 4.8 and 4.10 summarize

| Pretrained and IoU targets | | | | |
|---|---|---|---|---|
| **Set:** | **Validation** | | **Test** | |
| **Metrics:** | **mAP50** | **mAP50-95** | **mAP50** | **mAP50-95** |
| **YOLOv5N(90, 89, 160)** | 0.703 | 0.308 | 0.752 | 0.351 |
| **YOLOv5S(90, 85, 98)**$^{(1,2)}$ | 0.754 | 0.348 | 0.802 | 0.384 |
| **YOLOv5M(90, 88, 68)**$^{(1,2)}$ | 0.758 | 0.359 | 0.804 | 0.376 |
| **YOLOv5L(90, 89, 68)**$^{(1,2)}$ | 0.759 | 0.36 | 0.785 | 0.385 |
| **YOLOv5X(90, 68, 60)**$^{(1,2)}$ | 0.769 | 0.356 | 0.82 | 0.4 |
| **NMS ensemble(N+L)** | 0.769 | 0.355 | 0.801 | 0.385 |
| **NMS ensemble(X+L)** | 0.778 | 0.361 | 0.823 | 0.399 |
| **NMS ensemble(X+L+M+S)** | 0.777 | 0.36 | 0.799 | 0.387 |
| **NMS ensemble(L+M+S+N)** | 0.779 | 0.36 | 0.799 | 0.384 |
| **NMS ensemble(X+L+M+S+N)** | 0.776 | 0.358 | 0.806 | 0.391 |
| **WBF ensemble(N+L)** | 0.779 | 0.385 | 0.802 | 0.414 |
| **WBF ensemble(X+L)** | 0.805 | 0.411 | 0.852 | 0.449 |
| **WBF ensemble(X+L+M+S)** | **0.813** | **0.418** | **0.869** | **0.452** |
| **WBF ensemble(L+M+S+N)** | 0.801 | 0.417 | 0.831 | 0.42 |
| **WBF ensemble(X+L+M+S+N)** | 0.806 | 0.415 | 0.847 | 0.439 |

**Table 4.3:** mAP scores of the YOLOv5 base and ensemble models on the non-expert's validation and test sets. The base models use pretrained weights from the COCO dataset and are trained using IoU targets for the objectness score. The numbers in parentheses represent the epoch number, best epoch, and batch size, respectively. Models with a superscript of $(1, 2)$ indicate their inclusion in Ensemble1 and Ensemble2.

and rank the performance of both the experts and ensembles by averaging across the rows, excluding the experts' performance on their own sets for these calculations. Expert2 emerges as the best performing expert, while Ensemble1 stands out as the top ensemble.

## 4.3 Results on Experts' Data Using mAP Scores

We now showcase the performance of the experts and ensembles on the test sets using mAP scores. The mAP scores are computed by first removing all predictions and ground truths with scores below a certain threshold and setting all remaining scores to one. Next, we construct tables similar to those in 4.7 and 4.8, but with mAP replacing Brier scores. This process is repeated for threshold values ranging from 0.1 to 1, at increments of 0.1. For each threshold, the average mAP is calculated over the datasets of all other experts. These calculations are performed for both mAP50 and mAP50-95. The resulting average mAP scores are then plotted in Figure 4.1. Note that it is impossible for the models to

| Not pretrained and hard targets | | | | |
|---|---|---|---|---|
| **Set:** | **Validation** | | **Test** | |
| **Metrics:** | **mAP50** | **mAP50-95** | **mAP50** | **mAP50-95** |
| **YOLOv5N(300, 146, 100)**$^{(3)}$ | 0.7 | 0.303 | 0.711 | 0.316 |
| **YOLOv5S(300, 260, 200)**$^{(3)}$ | 0.677 | 0.301 | 0.705 | 0.318 |
| **YOLOv5M(300, 189, 130)**$^{(3)}$ | 0.702 | 0.314 | 0.65 | 0.307 |
| **YOLOv5L(220, 206, 72)**$^{(3)}$ | 0.716 | 0.318 | 0.706 | 0.33 |
| **YOLOv5X(220, 180, 48)** | 0.686 | 0.295 | 0.747 | 0.332 |
| **NMS ensemble(N+L)** | 0.732 | 0.321 | 0.727 | 0.333 |
| **NMS ensemble(X+L)** | 0.712 | 0.305 | 0.726 | 0.333 |
| **NMS ensemble(X+L+M+S)** | 0.726 | 0.311 | 0.721 | 0.338 |
| **NMS ensemble(L+M+S+N)** | 0.738 | 0.325 | 0.703 | 0.331 |
| **NMS ensemble(X+L+M+S+N)** | 0.729 | 0.311 | 0.722 | 0.337 |
| **WBF ensemble(N+L)** | 0.744 | 0.359 | 0.716 | 0.365 |
| **WBF ensemble(X+L)** | 0.738 | 0.357 | 0.759 | 0.376 |
| **WBF ensemble(X+L+M+S)** | 0.753 | 0.378 | 0.749 | 0.38 |
| **WBF ensemble(L+M+S+N)** | **0.756** | 0.381 | **0.762** | **0.382** |
| **WBF ensemble(X+L+M+S+N)** | 0.75 | **0.386** | 0.749 | 0.378 |

**Table 4.4:** mAP scores of the YOLOv5 base and ensemble models on the non-expert's validation and test sets. The base models use He initialization for the weights and were trained using hard targets for the objectness score. The numbers in parentheses represent the epoch number, best epoch, and batch size, respectively. Models with a superscript of **(3)** indicate their inclusion in Ensemble3.

predict objectness scores of exactly 1. To address this issue, when dealing with a threshold value of 1, we instead apply a threshold of 0.9 to the models.

In Figure 4.1, we compare individual ensembles against one another and based on this select three ensembles for further comparison with the experts: Ensemble1, Ensemble3, and Ensemble1B. Ensemble1 is chosen because it has the best performance according to the average Brier scores, as well as having the the highest mAP50 on the non-expert validation set. Ensemble3 is selected for having the highest average mAP50 scores among the ensembles, which is evident from Figure 4.1a. Ensemble1B is included to emphasize how Ensemble1 shows underconfidence at higher threshold values.

## 4.4 Discussion of the Quantitative Results

First, we will focus on the performances of the experts, followed by a discussion on the results of the ensemble models.

| Pretrained and hard targets | | | | |
|---|---|---|---|---|
| **Set:** | **Validation** | | **Test** | |
| **Metrics:** | **mAP50** | **mAP50-95** | **mAP50** | **mAP50-95** |
| **YOLOv5N(330, 247, 330)**$^{(4)}$ | 0.689 | 0.29 | 0.733 | 0.334 |
| **YOLOv5S(300, 284, 200)**$^{(4)}$ | 0.718 | 0.336 | 0.715 | 0.346 |
| **YOLOv5M(500, 100, 130)**$^{(4)}$ | 0.743 | 0.344 | 0.696 | 0.316 |
| **YOLOv5L(220, 202, 72)**$^{(4)}$ | 0.712 | 0.34 | 0.756 | 0.366 |
| **YOLOv5X(220, 207, 48)**$^{(4)}$ | 0.686 | 0.323 | 0.698 | 0.343 |
| **NMS ensemble(N+L)** | 0.719 | 0.33 | 0.77 | 0.363 |
| **NMS ensemble(X+L)** | 0.689 | 0.316 | 0.714 | 0.344 |
| **NMS ensemble(X+L+M+S)** | 0.707 | 0.318 | 0.725 | 0.35 |
| **NMS ensemble(L+M+S+N)** | 0.731 | 0.333 | 0.748 | 0.362 |
| **NMS ensemble(X+L+M+S+N)** | 0.709 | 0.317 | 0.725 | 0.347 |
| **WBF ensemble(N+L)** | 0.747 | 0.361 | 0.781 | 0.4 |
| **WBF ensemble(X+L)** | 0.753 | 0.38 | 0.762 | 0.385 |
| **WBF ensemble(X+L+M+S)** | 0.781 | 0.401 | **0.805** | 0.418 |
| **WBF ensemble(L+M+S+N)** | 0.787 | **0.406** | 0.804 | **0.426** |
| **WBF ensemble(X+L+M+S+N)** | **0.788** | 0.403 | 0.793 | 0.419 |

**Table 4.5:** mAP scores of the YOLOv5 base and ensemble models on the non-expert's validation and test sets. The base models use pretrained weights from the COCO dataset and are trained using hard targets for the objectness score. The numbers in parentheses represent the epoch number, best epoch, and batch size, respectively. Models with a superscript of (4) indicate their inclusion in Ensemble4.

### 4.4.1 Interpretation of Quantitative Results from Experts

Not shown in the previous subsections is the variation in flare-picking frequency among the experts. This information is now presented in Table 4.11. The data in Table 4.11 might help explain some of the quantitative differences observed when evaluating the experts on each other's datasets.

However, before discussing the results, we must first address Expert5's performance. In both the average Brier scores (Table 4.8 and Table 4.10) and the average mAP scores (Figure 4.1), Expert5 consistently performed the worst. This discrepancy in performance was primarily due to a different flare picking strategy compared to the other experts. Specifically, for certain flare clusters that other experts would segment individually, Expert5 would segment the entire cluster as one. Although this situation occurred infrequently, it was prevalent enough to consistently lower their performance. Furthermore, these instances typically involved clusters deemed to have high confidences. This explains why their performance did not improve at higher thresholds, as the dubious segmentations were never eliminated. One might consider removing

| Not pretrained and IoU targets | | | | |
|---|---|---|---|---|
| **Set:** | **Validation** | | **Test** | |
| **Metrics:** | **mAP50** | **mAP50-95** | **mAP50** | **mAP50-95** |
| **YOLOv5N(300, 265, 330)** | 0.717 | 0.325 | 0.783 | 0.377 |
| **YOLOv5S(400, 329, 200)** | 0.752 | 0.325 | 0.758 | 0.357 |
| **YOLOv5M(400, 336, 130)** | 0.721 | 0.329 | 0.741 | 0.331 |
| **YOLOv5L(220, 183, 70)** | 0.747 | 0.359 | 0.772 | 0.378 |
| **YOLOv5X(220, 171, 48)** | 0.731 | 0.348 | 0.792 | 0.398 |
| **NMS ensemble(N+L)** | 0.761 | 0.354 | 0.785 | 0.376 |
| **NMS ensemble(X+L)** | 0.75 | 0.357 | **0.823** | 0.4 |
| **NMS ensemble(X+L+M+S)** | 0.752 | 0.345 | 0.819 | 0.396 |
| **NMS ensemble(X+L+M+S+N)** | 0.759 | 0.347 | 0.821 | 0.396 |
| **NMS ensemble(L+M+S+N)** | 0.769 | 0.35 | 0.793 | 0.368 |
| **WBF ensemble(N+L)** | 0.772 | 0.392 | 0.814 | 0.428 |
| **WBF ensemble(X+L)** | 0.775 | 0.398 | 0.786 | 0.422 |
| **WBF ensemble(X+L+M+S)** | 0.8 | 0.416 | 0.807 | **0.439** |
| **WBF ensemble(L+M+S+N)** | 0.8 | 0.407 | 0.797 | 0.434 |
| **WBF ensemble(X+L+M+S+N)** | **0.808** | **0.42** | 0.801 | 0.434 |

**Table 4.6:** mAP scores of the YOLOv5 base and ensemble models on the non-expert's validation and test sets. The base models use He initialization for the weights and were trained using IoU targets for the objectness score. The numbers in parentheses represent the epoch number, best epoch, and batch size, respectively. None of the base models in this table are used in any of the final ensembles.

Expert5's data entirely due to these dubious segmentations; however, this would be a mistake. If we assume that all other experts fail to segment these specific dubious segmentations made by Expert5, then they would all incur the same penalty, effectively maintaining the relative difference between the experts. Also, the remaining predictions by Expert5 are still informative and keeping them will help contribute to uncovering the absolute ground truth[1].

Now, the results in Table 4.8 and 4.10 show that Expert2 was the best in terms of the Brier score. One could explain this by simply conjecturing that Expert2 was the superior segmenter, managing to capture the essence of what constitutes a good segmentation and confidence score for a flare. While possibly true, another explanation is that the Brier metric, as defined in Subsection 3.4, has a flaw. If we consider the different frequencies of flare picking, shown in Table 4.11, it is clear that Expert2 was more conservative in generating segmentations for flares deemed unlikely to be true flares. Indeed, if we consider the cumulative

1. By 'absolute ground truth', we refer to a ground truth defined by averaging or otherwise summarizing across the distribution of predictions from all possible expert labelers.

| | | Test set | | | | | |
|---|---|---|---|---|---|---|---|
| | | **Expert1** | **Expert2** | **Expert3** | **Expert4** | **Expert5** | **Expert6** |
| **Predictor** | **Expert1** | 0 | 24.94 | 35.02 | 53.3 | 50.96 | 76.7 |
| | **Expert2** | 24.94 | 0 | 36.46 | 51.88 | 36.74 | 64.52 |
| | **Expert3** | 35.02 | 36.46 | 0 | 42 | 68.1 | 73.48 |
| | **Expert4** | 53.3 | 51.88 | 42 | 0 | 66.56 | 76.46 |
| | **Expert5** | 50.96 | 36.74 | 68.1 | 66.56 | 0 | 60.38 |
| | **Expert6** | 76.7 | 64.52 | 73.48 | 76.46 | 60.38 | 0 |
| | **Ensemble1** | 43.27 | 35.23 | 43.51 | 49.43 | 48.43 | 73.06 |
| | **Ensemble2** | 41.13 | 41.41 | 46.39 | 51.07 | 51.89 | 80.59 |
| | **Ensemble3** | 49.77 | 45.94 | 52.53 | 54.76 | 52.80 | 77.58 |
| | **Ensemble4** | 49.37 | 50.55 | 53.93 | 56.02 | 57.51 | 83.93 |
| | **Ensemble1B** | 50.09 | 46.45 | 52.11 | 55.54 | 58.18 | 82.99 |
| | **Ensemble2B** | 48.95 | 52.87 | 56.11 | 58.07 | 60.26 | 88.19 |

**Table 4.7:** Comparison of Brier50 scores for expert and ensemble predictors on test sets. On the rows are the predictors and on the columns are the possible test sets.

| Predictor | Rank | Avg. Brier50 |
|---|---|---|
| 1 | Expert2 | 42.908 |
| 2 | Expert1 | 48.184 |
| 3 | Ensemble1 | 48.222 |
| 4 | Expert3 | 51.012 |
| 5 | Ensemble2 | 52.008 |
| 6 | Ensemble3 | 55.563 |
| 7 | Expert5 | 56.548 |
| 8 | Ensemble1B | 57.560 |
| 9 | Expert4 | 58.040 |
| 10 | Ensemble4 | 58.552 |
| 11 | Ensemble2B | 60.742 |
| 12 | Expert6 | 70.308 |

**Table 4.8:** Ranking the experts and ensembles based on their average brier50 score.

sum of flares from 0.1 to 0.6 confidence, then Expert2 only segmented 27 flares. This is 13 less than the expert with the second least low confidence predictions: Expert2 at 40, who coincidentally is the second best at the average Brier50 score.The problem arises when someone excessively segments low confidence flares not recognized by other experts, thereby accruing errors. A possible remedy for this flaw is to use the same procedure as when computing the average mAP scores in Figure 4.1. That is, removing flare segmentations by thresholding them away in intervals of 0.1 until one arrives at 1 confidence, however, this was not tested in the present study. Also of note is how the

| | | Test set | | | | | |
|---|---|---|---|---|---|---|---|
| | | **Expert1** | **Expert2** | **Expert3** | **Expert4** | **Expert5** | **Expert6** |
| **Predictor** | **Expert1** | 0 | 76.202 | 83.348 | 101.02 | 102.02 | 137.19 |
| | **Expert2** | 76.202 | 0 | 76.432 | 87.712 | 68.816 | 101.08 |
| | **Expert3** | 83.348 | 76.432 | 0 | 90.052 | 102.94 | 136.17 |
| | **Expert4** | 101.02 | 87.712 | 90.052 | 0 | 108.66 | 139.69 |
| | **Expert5** | 102.02 | 68.816 | 102.94 | 108.66 | 0 | 114.92 |
| | **Expert6** | 137.19 | 101.08 | 136.17 | 139.69 | 114.92 | 0 |
| | **Ensemble1** | 84.16 | 70.16 | 83.11 | 91.21 | 86.62 | 125.48 |
| | **Ensemble2** | 86.58 | 78.24 | 88.23 | 95.36 | 93.06 | 134.10 |
| | **Ensemble3** | 100.49 | 86.09 | 101.82 | 108.83 | 97.62 | 140.84 |
| | **Ensemble4** | 103.28 | 93.43 | 104.37 | 111.51 | 104.87 | 147.13 |
| | **Ensemble1B** | 96.73 | 85.92 | 96.33 | 103.93 | 101.48 | 140.59 |
| | **Ensemble2B** | 100.37 | 94.63 | 103.36 | 110.04 | 109.07 | 148.63 |

**Table 4.9:** Comparison of Brier50-95 scores for expert and ensemble predictors on test sets. On the rows are the predictors and on the columns are the possible test sets.

| **Rank** | **Predictor** | **Avg. Brier50-95** |
|---|---|---|
| 1 | Expert2 | 82.0484 |
| 2 | Ensemble1 | 90.123 |
| 3 | Ensemble2 | 95.928 |
| 4 | Expert3 | 97.7884 |
| 5 | Expert5 | 99.4712 |
| 6 | Expert1 | 99.9560 |
| 7 | Ensemble1B | 104.16 |
| 8 | Expert4 | 105.4268 |
| 9 | Ensemble3 | 105.95 |
| 10 | Ensemble4 | 110.77 |
| 11 | Ensemble2B | 111.02 |
| 12 | Expert6 | 125.8100 |

**Table 4.10:** Ranking the experts and ensembles based on their average brier50-95 score.

relative rankings change from Table 4.8 to Table 4.10. As mentioned, Brier50-95 places greater emphasis on correct segmentations than Brier50, meaning that a relative drop in rankings can be interpreted as being due to poorer segmentations. 'Correct' in this context means segmentations that are more aligned with those of other experts. Thus, if some experts have personal quirks in how they segment, then the scores of other experts will drop if they cannot precisely mimic these interpretations of segmentations.

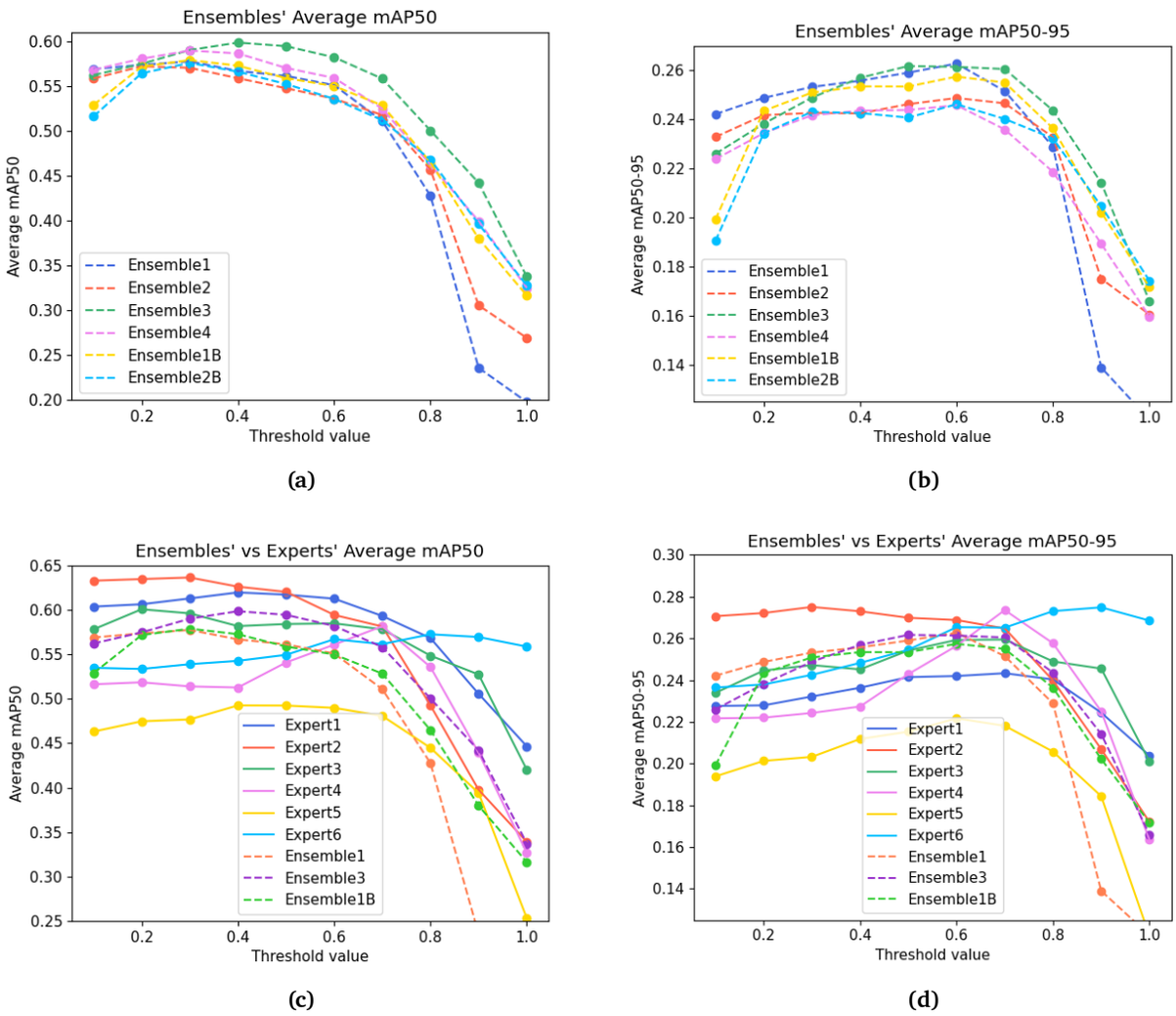



**Figure 4.1:** Comparison of performance between experts and ensembles using the mAP metric. For each predictor, the mAP scores at each threshold are summarized by averaging the mAPs obtained from the experts' test sets.

Next, we will examine the performances based on the average mAP scores, as illustrated in Figure 4.1. Here, we might again be able to explain some of the results by referring to Table 4.11. Generally, the scores of the experts follow similar trends across each threshold value, albeit with some variations in rankings. Notably, Expert2 consistently ranks as the best at low threshold values, only losing the top rank as threshold values exceed 0.5. Particularly interesting is Expert6's performance, ranking as the best for thresholds between 0.8 to 1. When examining flare picking frequencies, Expert6 clearly segments the most flares at higher confidence levels. For instance, at a confidence level of one, Expert6 segments 44 more flares than Expert2. This might suggest

| Confidence | Expert1 | Expert2 | Expert3 | Expert4 | Expert5 | Expert6 | Non-expert |
|---|---|---|---|---|---|---|---|
| 0.1 | 0 | 0 | 14 | 18 | 18 | 6 | 0 |
| 0.2 | 3 | 3 | 10 | 15 | 12 | 8 | 0 |
| 0.3 | 7 | 4 | 8 | 14 | 10 | 8 | 0 |
| 0.4 | 1 | 12 | 7 | 19 | 7 | 10 | 0 |
| 0.5 | 21 | 2 | 8 | 16 | 10 | 25 | 0 |
| 0.6 | 8 | 6 | 8 | 30 | 16 | 8 | 0 |
| 0.7 | 9 | 15 | 7 | 16 | 10 | 25 | 0 |
| 0.8 | 9 | 11 | 7 | 20 | 18 | 15 | 0 |
| 0.9 | 11 | 7 | 8 | 12 | 16 | 20 | 0 |
| 1 | 39 | 19 | 16 | 28 | 22 | 63 | 116 |
| Sum | 108(50) | 79(24) | 93(24) | 188(40) | 139(38) | 188(83) | 116 |

**Table 4.11:** Frequency of flare picking for the experts and non-expert. In parentheses is the sum of the flares from 0.9 to 1 confidence. Experts were allowed to give a confidence score between 0.1 and 1, while the non-expert had to use true or false labels. The data of the experts was collected after the non-expert and it was decided that we wanted to allow for more informative labels.

that Expert6 effectively segments nearly all flares identified by other experts at high confidence levels. Moreover, for each confidence 1 label created by Expert6 that is not recognized by other experts, the scores of these experts drop due to missing it. This effect is more pronounced given the limited number of experts, only six, and a test set of only 30 images. With a larger pool of experts and more test images, such effects would likely be less significant and more evenly distributed. One asymmetry that arises is the fact that Expert6's performance doesn't seem to drop to the same extent as the other experts. We might expect to see Expert6's score drop, perhaps even more than the others, precisely because they tend to over-segment. This over-segmentation could result in a high number of false positives when calculating the mAP. If we again consider the precision and recall measures in the Subsection 3.4 then for high confidence flares, this over segmentation strategy might be considered as having a preference for higher recall values than precision. While it is not the only reason, of course, the oversegmenting of high confidence flares might help explain Expert6's strong performance. Also, while we do not consider the mAP metric to be flawed, it has been suggested that mAP is not a perfect metric for object detection, despite it being the gold standard [60].

Before moving on to the ensemble results, we once again note the variation in the number of objects considered by the experts as high confidence flares, that is, flares from 0.8 to 1, or 0.9 to 1 confidence. It is to be expected that the number of low-confidence flares will vary greatly, as these are by nature more subjective, and the interpretation of a correct confidence score is harder to ascertain. However, we should expect to see agreement on the correct total

number of high confidence flares, particularly the ones one would assign more than 0.8 confidence to. In Table 4.11, we see that Expert2 and Expert3 counted the same total number of flares from 0.9 to 1 confidence, while Expert4 and Expert5 counted 40 and 38, respectively. Thus, it might be the case that Expert2 and Expert3, and Expert4 and Expert5, had similar strategies for selecting high confidence flares. Expert1 and Expert6 determined that there are more high confidence flares than the rest the of the experts, at 50 and 83 respectively. It is clear that there is significant variation among the experts regarding the number of ground truth high confidence flares. These findings highlight the difficulty in determining an absolute ground truth for gas flares datasets, as even trained experts will vary in their opinion on what should and shouldn't be included. This was also a key result found in [18], where instead of selecting flares and giving them a confidence score, there were four discrete classes which flares could fall into. Additionally, it was even found that an expert who segmented and classified a dataset twice, with the first being months away from the second, would vary significantly in how they approached the labeling both times.

The variation between experts presents both a major challenge and a major motivation for adopting a standardized machine learning algorithm for flare segmentation: firstly, the subjectivity in the task of segmenting flares can introduce considerable expert/human-injected bias in training and validation data sets. This may indeed explain differences between some of the expert and machine generated results in the present study. Secondly, it shows that labeling of gas flares need standardization. If datasets are labeled uniquely by different experts due to personal interpretations, then this also changes the underlying distribution of the labels. These slight variations in the underlying distributions of the datasets can affect both local environmental assessments and add noise to global carbon budget estimates and higher order climate prediction models. Addressing the effect of ergonomic factors and differences in expert background can help explain, but also improve consistency between human labelers (see e.g. [56]). This would also contribute to identify and mitigate noise in training datasets. Finally, standardization and widespread adoption of an automated labeling process using deep learning based models, such as the ones introduced in this thesis, would solve the problem of non-standardized flare labeling. This of course requires a sufficiently good, unbiased model, which can segment and detect flares at or beyond human level.

### 4.4.2 Interpretation of Quantitative Results from Ensembles

From Tables 4.2 to 4.6, we observe several notable points. The first is that ensemble models are superior to base models. This superiority is evident from

the consistently better mAP50 and mAP50-95 scores on both the validation and test sets. Although some base models occasionally outperform certain ensemble models, the general trend indicates that ensembling models improves performance. The second observation is that ensembles formed with WBF outperform those with NMS. Across both metrics and on both sets, WBF ensembles achieve the highest scores, except for a singular case in Table 4.6, where an NMS ensemble attains the highest mAP50 score on the test set. The final observations are twofold: pretrained weights are superior to weights initialized by He initialization, and IoU targets are more effective than hard targets, once again shown by the mAP scores. Using pretrained weights is not always necessary for problems involving large datasets. In such scenarios, a model trained with pretrained weights might not perform any better than one without. However, as noted in Subsection 2.5.1, pretraining often benefits smaller datasets. The dataset used for this problem was relatively small, consisting of only 1414 images, which likely explains a significant portion of the performance differences between pretrained and non-pretrained models.

Table 4.1 shows the ensembles which we use for comparison with the experts. Of note are Ensemble1 and Ensemble2, which were crafted for the explicit purpose of maximizing the mAP50 scores on the validation and test sets, respectively. These two ensembles were created by simply combining the best base models on the two sets. This type of combination can be considered a form of hyperparameter tuning, where the hyperparameter represents the specific combinations of base models. Tuning hyperparameters on the test set is considered a cardinal sin in deep learning, the reason being that hyperparameter optimization can be considered a form of overfitting: the hyperparameters found are fitted to the particular set used to tune them. Thus, if we tune hyperparameters on the test set, then the performance found might overestimate the models actual performance. Fitting hyperparameters on the test set becomes less of a problem as its size grows, because it will progressively become a better approximation of the underlying distribution that generated the dataset. The test set size used in this problem consists of only 30 images; thus, we can reasonably assume that the remarkably high mAP50 of 0.882 achieved on the non-expert test set is overstated and will not hold for new non-expert test examples.

Next, we consider the ensemble performances on the Brier scores. In particular, it is interesting to note how Ensemble1 achieves the rank of 3 and 2 on the average Brier50 and Brier50-95 scores, respectively. Ensemble2 also ends up having good scores in this metric. However, due to the reasons outlined earlier in this section, we should not place too much emphasis on how well the ensembles performed here.

We now turn our attention to the average mAP scores, shown in Figure 4.1. But

first, we again point out how Ensemble1 and Ensemble2 tend to be underconfident, as measured by their objectness score, and that Ensemble3 and Ensemble4, as well as Ensemble1B and Ensemble2B, had changes explicitly made to make them more confident. Ensemble3 and Ensemble4 do this by training with hard targets, and Ensemble1B and Ensemble2B have had their scores boosted by 0.1. Now, Figure 4.1a and 4.1b show that Ensemble3 performed the best on the mAP metric. There is no particular reason why Ensemble3 should outperform Ensemble4, as the only difference in the training process between them was having pretrained weights or not. Indeed, Ensemble4 performs about as well as the other ensembles, which leads us to believe that Ensemble3's standout performance from the rest is no more than random luck, and that if we were to retrain it, it likely would not be the best performer again. Further, we see that the boosting trick in Ensemble1 B and 2B seems to have had its intended effect, as for the higher thresholds we can see that they greatly outperform Ensemble1 and Ensemble2, whose performances conversely drop drastically.

Finally, we compare Ensemble1, Ensemble3, and Ensemble1B with the experts in Figures 4.1c and 4.1d. The qualitative performances of Ensemble3 and Ensemble1B appear to be largely indistinguishable from those of the experts, a trend that is also observed for Ensemble1 before its score drops at the higher thresholds. Additionally, we note how the ensembles seem to have more correct segmentations than the experts, as exemplified by their relative increase in performance from average mAP50 to average mAP50-95 over the experts. However, as will be shown in the quantitative analysis, there are considerable differences between the ensembles and experts, which still make them less desirable than a human segmenter.

## 4.5 Qualitative Analysis

Quantifying what constitutes human level performance is hard, and developing sufficient conditions for exactly what would convey human level performance is beyond the scope of this thesis. However, we can discuss what some necessary conditions are and note that if one refines these necessary conditions enough, then one will likely find conditions which can be considered sufficient.

We consider achieving performance at or beyond what the average human expert can do on some quantitative metric, such as the average Brier and average mAP used in this thesis, as a necessary condition for human level performance. Based on the results shown in Subsection 4.2 and 4.3, we consider this condition to be fulfilled or close to being fulfilled. However, despite performing quantitatively well on the chosen metrics, the models still perform noticeably worse in a qualitative manner; therefore, we do not yet consider

them good enough to replace human experts. Thus, we add achieving human level qualitative performance as another necessary condition.

The qualitative analysis is done by comparing images from the test set for an ensemble model together with the experts. More specifically, we will use Ensemble1 as the ensemble of choice, which consists of eight models in total. In Subsection 3.2.3, it was described how the model outputs were fused together using WBF. Because we are doing a purely visual comparison, it is now not necessary to prune away pixels with high uncertainty in the predicted mask, like how we would normally do when evaluating on a test set. This will allow us to visually assess the epistemic uncertainty[2] in mask segmentations. Additionally, since multiple objectness scores contribute to the fused objectness score, we will calculate and display its standard deviation in the figures. These standard deviations, indicated in parentheses next to the fused scores, represent the epistemic uncertainty associated with the objectness scores.

To allow for a succinct comparison with the experts, we will summarize the ground truths from the experts by using WBF instead of presenting them one by one. The discussion of ensemble outputs above will then also apply to the fused predictions by the experts. In [70], it is noted that fusing predictions of an expert panel in this way can lead to more accurate ground truth labels. Additionally, it also gives us access to ground truth uncertainties, which can be of interest if one wants to benchmark the performance of uncertainty estimation methods, such as Bayesian deep learning. Finally, we also include the non-expert ground truth for each presented image, which may help in understanding the rationale behind the ensemble model's predictions. Four images where the ensemble fails qualitatively are shown and described first, followed by five images considered as successes.

The first figure, Figure 4.2, displays an echogram depicting two disjoint objects that, according to the experts' fused ground truth shown in Figure 4.2a, likely form a single flare. However, the ensemble fails to recognize this, as it was not trained on expert ground truth sets but on the non-expert's data. In terms of predicting on the the non-expert test set labels, the ensemble's behavior is actually appropriate. This accuracy in reflecting the non-expert ground truth is to be expected, given its exceedingly high mAP50 score of 0.828 on the validation set, as shown in Table 4.1.

In Figure 4.3, the second example is presented where the ensemble qualitatively

2. In deep learning, we consider two types of uncertainty: epistemic (model) uncertainty and aleatoric (data) uncertainty [20]. Briefly, if we can reduce the uncertainty by gathering more data, then it is epistemic. Conversely, if the uncertainty cannot be reduced with more data and is simply some sort of inherent randomness to the problem, then it is aleatoric.

fails, this time in regards to both the experts' and the non-expert's ground truth. In the image, there is one object that can be confidently identified as a flare. This particular flare is distinctively unique in its form, and among the 1414 images comprising the test data, no other flares resemble it. The flare's uniqueness lies in its strong signal at the bottom, which weakens abruptly yet extends quite far upwards. The ensemble's failure to accurately segment this flare can be attributed to a lack of similar data in the dataset – with more examples of this type of flare, the ensemble might have succeeded in correctly segmenting it. Interestingly, the ensemble does recognize the flare's unique nature, assigning relatively low objectness scores to the two parts.

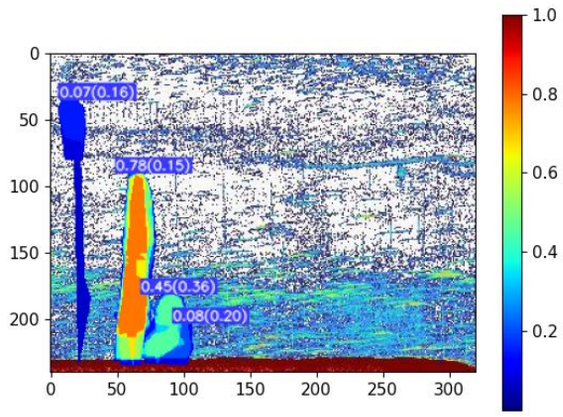
In Figure 4.4, the third comparison is presented, showing an area with lots of obvious flare activity, recognized by both the experts and the non-expert. We particularly highlight the leftmost flare, which is uniquely thin and tall. While humans can easily discern this as a true flare, the ensemble fails to do so. Similar to the situation in Figure 4.3, we attribute this failure to a lack of data coverage — the dataset contains exceedingly few thin and long flares. The ensemble performs better with the other tall flares in the picture, which has thicker roots and other characteristics commonly found in the dataset, as reflected in the higher objectness score it receives. Additionally, it is noteworthy that the ensemble demonstrates more underconfidence in its predictions for most of the 100% confidence flares in the image compared to the experts. This tendency towards underconfidence is a notable weakness of the ensemble when compared to human judgments.

The final example of qualitative failure by the ensemble is presented in Figure 4.5. This example is included to emphasize the presence of what may be considered labeling errors in the dataset created by the non-expert. As illustrated in Figure 4.5c, three objects are identified as flares and assigned a 100% confidence score by the non-expert. However, a comparison with the experts' assessment in Figure 4.5a reveals an extremely low probability of actual flares being present in the image. While the ensemble model accurately captures the non-expert ground truth, in this instance, it leads to a significant misjudgment. The training dataset contains a relatively high prevalence of such labeling errors, and it is likely that rectifying these errors could substantially improve the ensemble's performance on the experts' test sets.

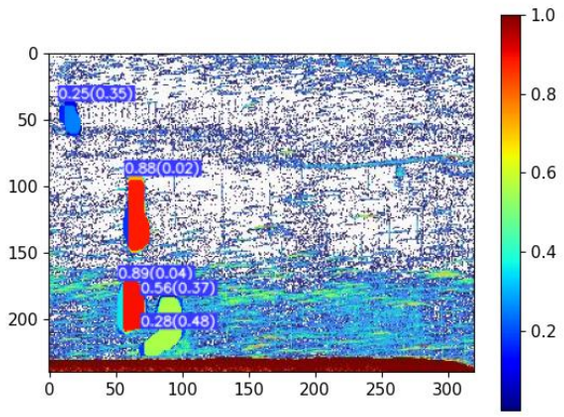
Generally speaking, it is interesting to note how the ensemble not only identifies many high confidence objects but also recognizes many of the same low confidence objects as the experts. This is achieved without having been explicitly trained on low confidence labels, such as those provided by the experts. This ability arises precisely because we are ensembling models together. As mentioned, the training dataset contains considerable noise. Because of this, models may sometimes segment objects that are extremely unlikely to be flares,

but assign them high objectness scores. However, the likelihood that all of the base models will learn the same incorrect predictions is exceedingly low. These erroneous predictions are averaged out in the ensemble model, resulting in very low confidence predictions. In this way, the ensemble is able to pick out a clear signal from all the noise, that is, the objects most likely to be actual flares.

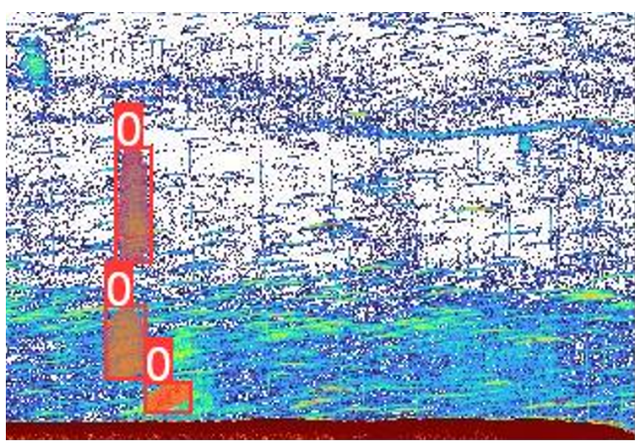
We end this chapter by showcasing examples where the ensemble's performance is considered qualitatively on par with the experts. These are given in Figures 4.6 to 4.10, where a short description is given for each.




**(a)** The experts' ground truth labels after fusing them with WBF.

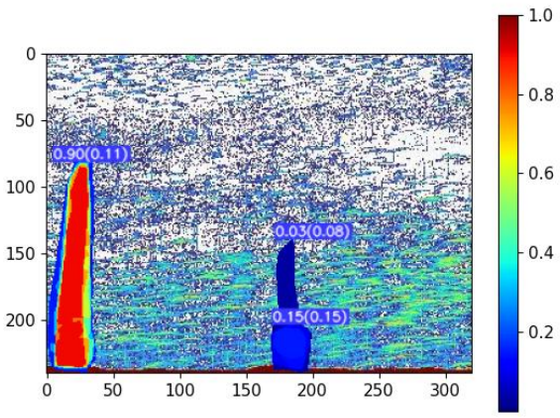



**(b)** Ensemble1's predictions.

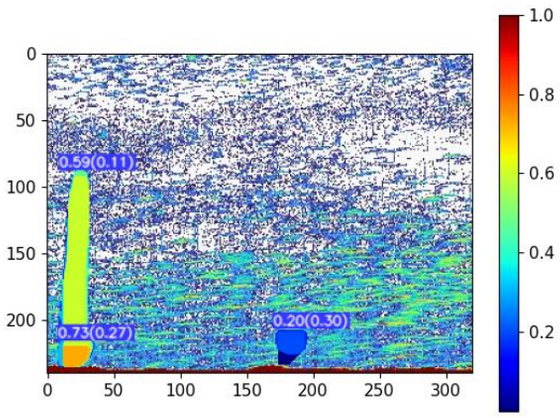



**(c)** The ground truth on the non-expert test set.

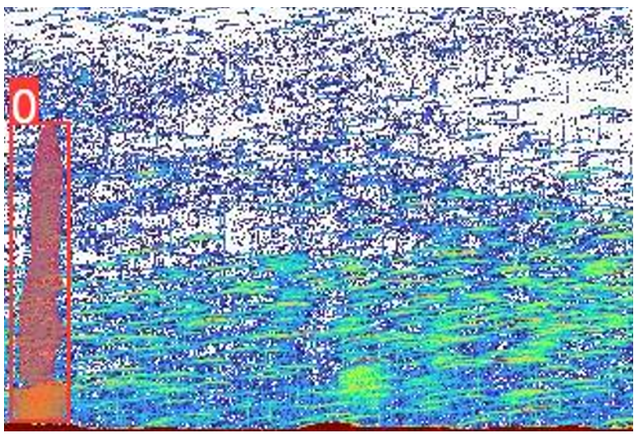
**Figure 4.2:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. We observe that the ensemble fails to replicate the behaviour required by human experts, that is, segment the object likely to be a flare as a whole, instead of two parts. However, the ensemble is successful in terms of predicting the non-expert ground truth.




**(a)** The experts' ground truth labels after fusing them with WBF.

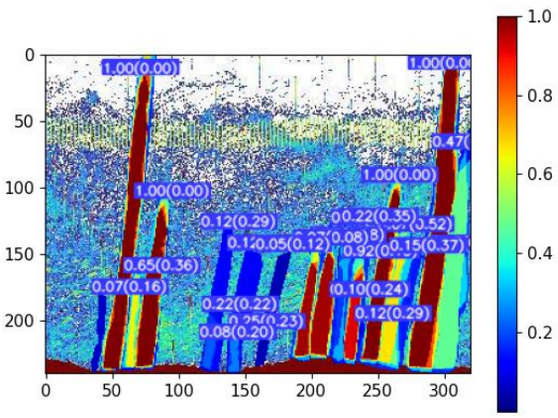



**(b)** Ensemble1's predictions.

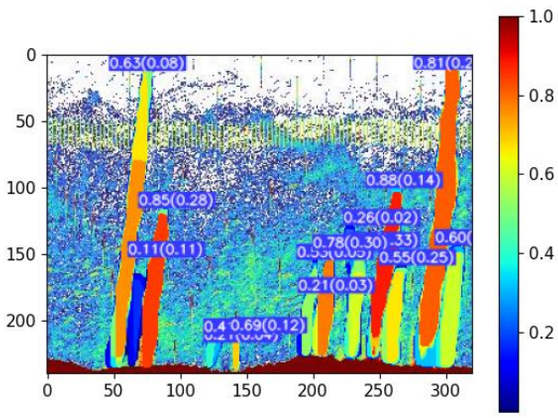



**(c)** The ground truth on the non-expert test set.

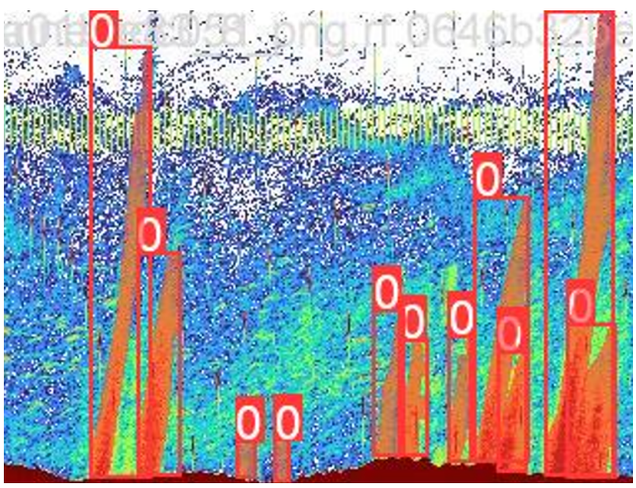
**Figure 4.3:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. We observe that the ensemble fails to replicate the behaviour required by both the experts and the non-expert. However, the ensemble recognizes the unique situation and gives a low objectness score on the two parts which should have been segmented as a whole.




**(a)** The experts' ground truth labels after fusing them with WBF.

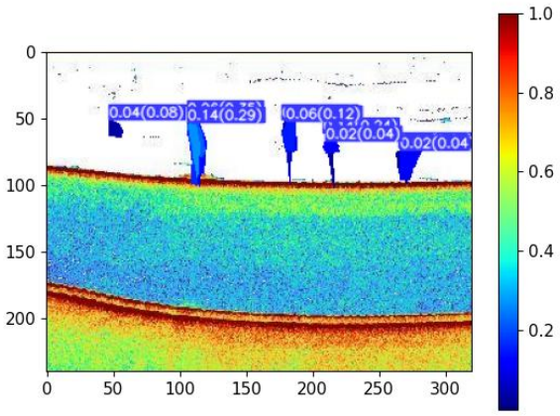



**(b)** Ensemble1's predictions.

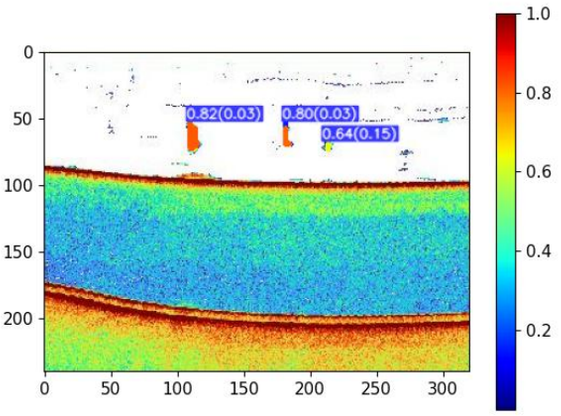


**(c)** The ground truth on the non-expert test set.

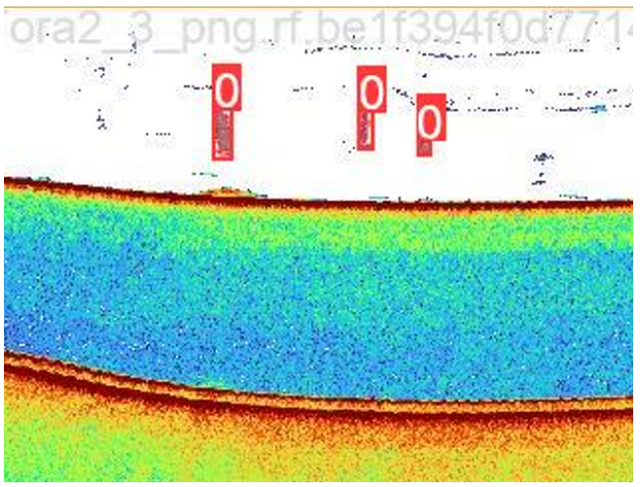
**Figure 4.4:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. We observe that the ensemble fails to give high confidence scores as the human experts would. In particular, we draw attention to the leftmost flare, which got a too low score. We attribute this to a lack of data coverage in the space of training examples where this flare resides.




**(a)** The experts' ground truth labels after fusing them with WBF.

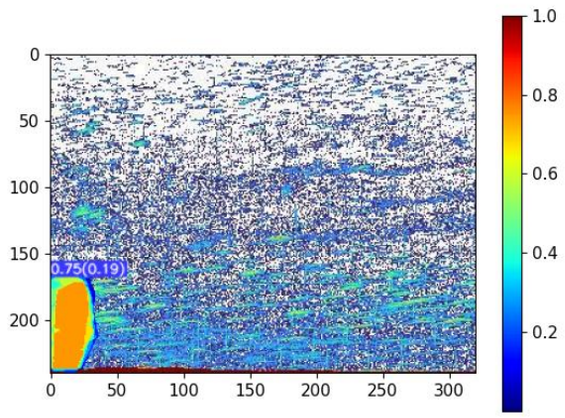



**(b)** Ensemble1's predictions.

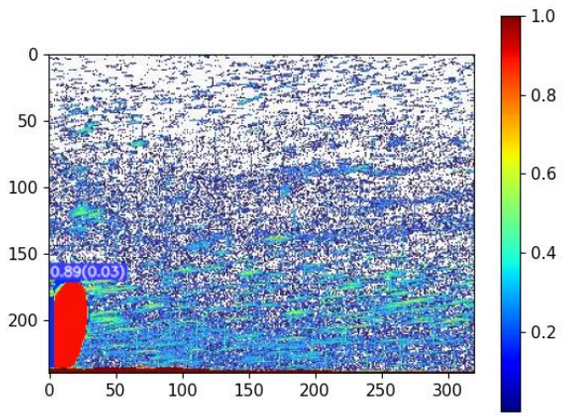


**(c)** The ground truth on the non-expert test set.

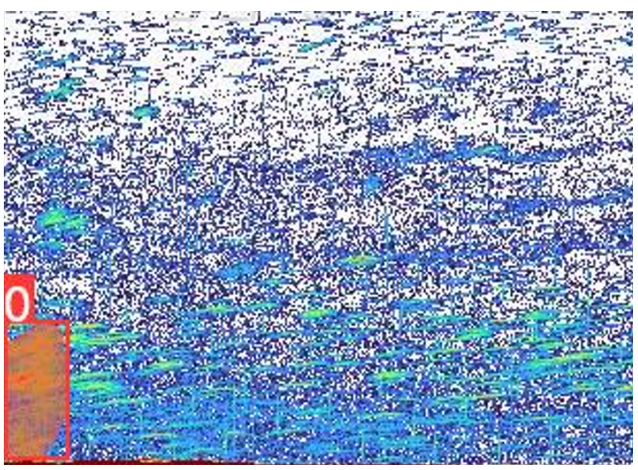
**Figure 4.5:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. The ensemble successfully identifies the non-expert's test set ground truth. However, in this instance, this identification is a significant error in terms of the expert test sets. From (a), it is clear that the experts almost unanimously agree that there are no flares in the image. The dataset created by the non-expert is full of such labeling errors, which contribute to the ensemble's erroneous predictions in this case.




**(a)** The experts' ground truth labels after fusing them with WBF.

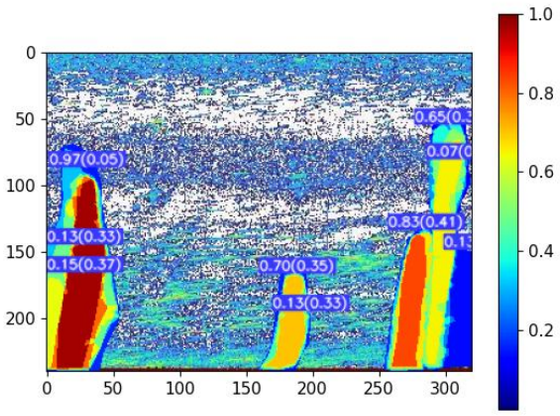



**(b)** Ensemble1's predictions.

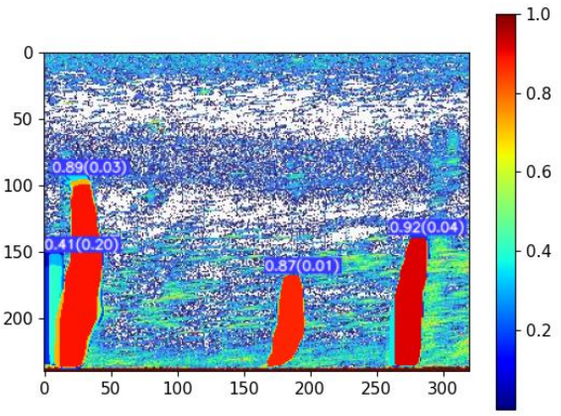



**(c)** The ground truth on the non-expert test set.

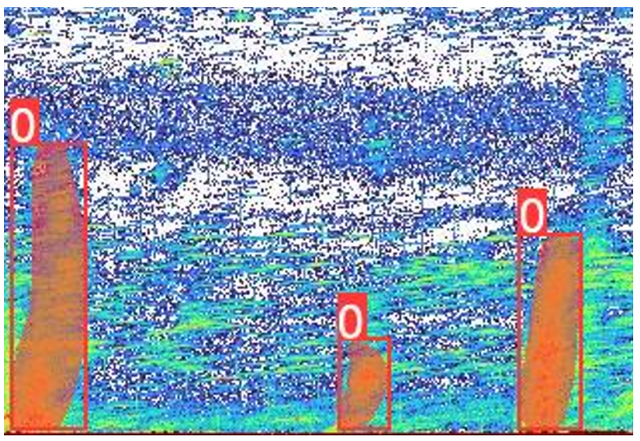
**Figure 4.6:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. The ensemble correctly mimics the behavior of the experts, albeit with slightly too high confidence.




**(a)** The experts' ground truth labels after fusing them with WBF.

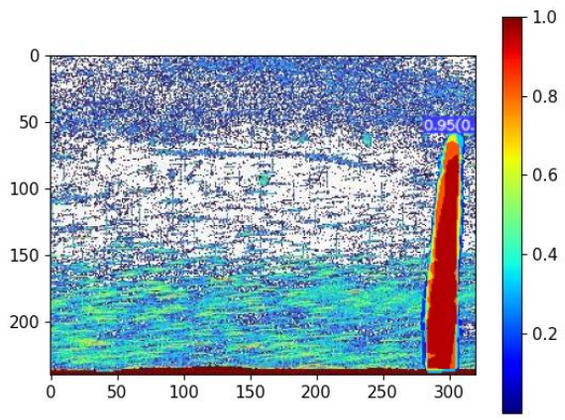



**(b)** Ensemble1's predictions.

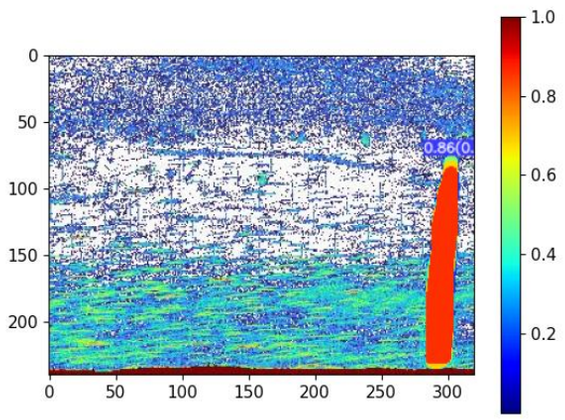



**(c)** The ground truth on the non-expert test set.

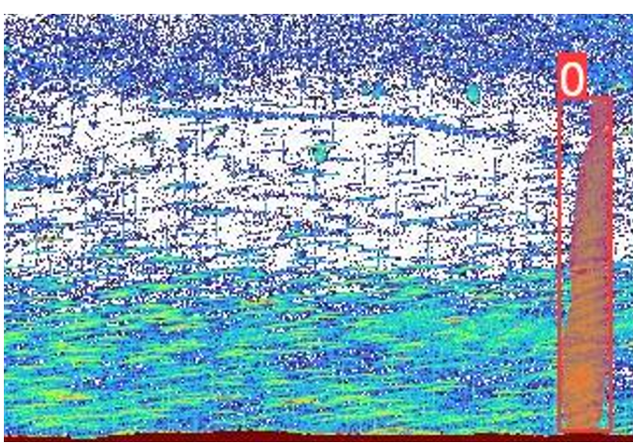
**Figure 4.7:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. Although the experts make some predictions that the ensemble does not share, these predictions are of low confidence. Thus, we regard this as a success for the ensemble. Furthermore, it is notable how the uncertainty associated with the segmentation mask is greater for the experts than for the ensemble.




**(a)** The experts' ground truth labels after fusing them with WBF.

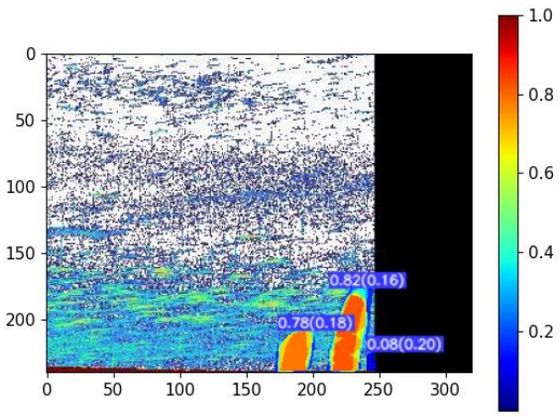



**(b)** Ensemble1's predictions.

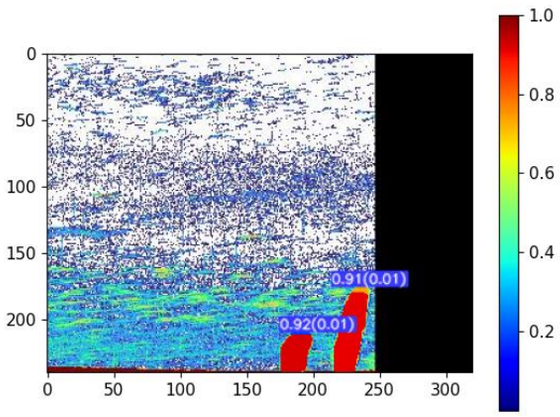



**(c)** The ground truth on the non-expert test set.

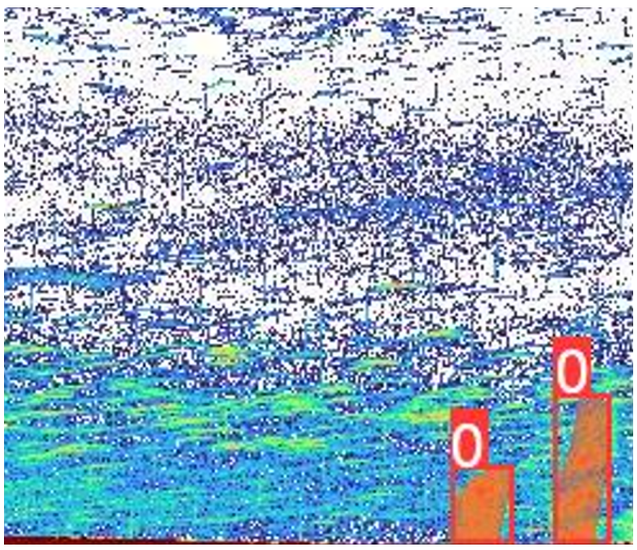
**Figure 4.8:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. The ensemble correctly mimics the behavior of the experts, albeit with slightly less confidence.




**(a)** The experts' ground truth labels after fusing them with WBF.

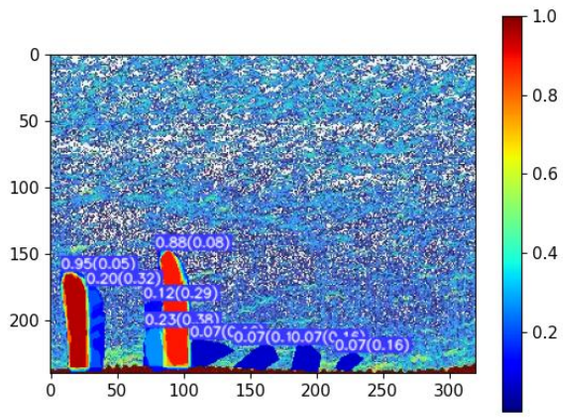



**(b)** Ensemble1's predictions.

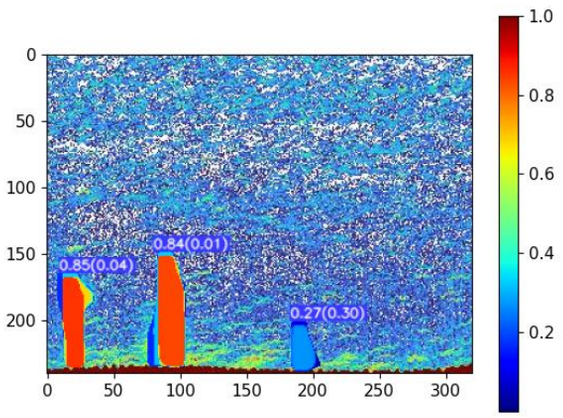



**(c)** The ground truth on the non-expert test set.

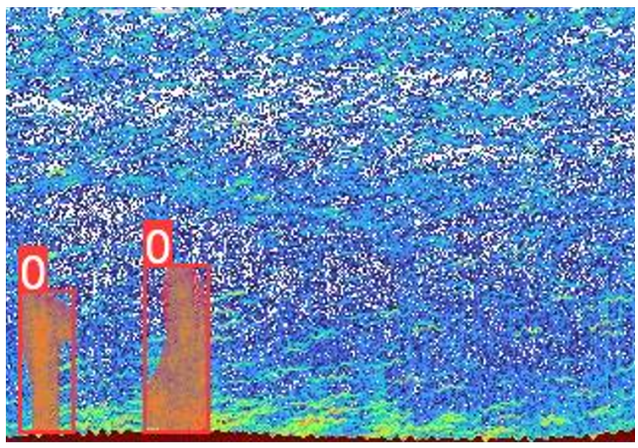
**Figure 4.9:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. The ensemble correctly mimics the behavior of the experts, albeit with slightly too high confidence.




**(a)** The experts' ground truth labels after fusing them with WBF.




**(b)** Ensemble1's predictions.




**(c)** The ground truth on the non-expert test set.

**Figure 4.10:** Visual comparison between Ensemble1 and the experts on one of the 30 test images. The ensemble successfully identifies the two flares in the image, notably assigning high epistemic uncertainty to the protruding parts of the flares. During test set benchmarking, we would remove these areas of high uncertainty, resulting in segmentations that seemingly align perfectly with those of the experts

# 5

# Conclusion and Future Work

## 5.1 Improving on the presented results

There are areas of obvious improvement, which, if implemented, will likely improve or significantly improve on the results already shown in this thesis. The first and most obvious one is improving upon the dataset used. It is clear from the results presented in Chapter 4 that there is a significant amount of noise in the training dataset labels, which adversely affects the model's output. While ensemble models help mitigate this issue, directly reducing label noise by having an expert redo the dataset could lead to more substantial improvements. Furthermore, an upgrade to the dataset could involve collecting new labels. Transitioning from the hard labels used by the non-expert to the soft labels used by experts could be particularly beneficial. Soft labels, which provide a spectrum of certainty rather than binary classifications, are especially useful in the challenging task of segmenting flares. They offer the model richer information about the varying confidence levels associated with flare signals. Furthermore, they offer a more straightforward approach to labeling: when in doubt, it is easy to precisely reflect the degree of doubt in the label, rather than being limited to a binary true or false label. In the training process, these labels could be used as targets for either the objectness score or for the class scores. This approach gains relevance considering that the class scores in this thesis were not a primary focus, as the model was dealing with only

one class and thus learned to always predict it correctly. Giving the class scores actually useful targets during training is therefore interesting. Future work could explore the implications of using soft labels on model training and performance, potentially leading to more accurate flare detection. Also, we remark on the exceedingly high mAP50 found on the non-expert's validation and test set, even in the face of noisy labels, which leads us to believe that the same performance can be replicated on datasets from other labelers. The strong quantitative performance indicates that the ensemble is able to replicate the behavior of the human who produced its training set, a point we confirm to qualitatively in Section 4.5.

In Section 4.5, we also highlight a data coverage issue. The remedy for this issue is simple: collect more data. Expanding the dataset is also expected to enhance the overall performance of the models by providing a more comprehensive training base. Additionally, the test datasets used consist of only 30 images each; this is a very small number and does not well represent the underlying probability distribution of the data, potentially biasing the results either negatively or positively. Therefore, we see it as crucial to recruit more experts and use a larger test set to increase the certainty and robustness of our findings.

We have only used one model architecture for this problem, namely the YOLOv5 architecture. It would be interesting to try out other architectures to see how they perform in comparison. Additionally, ensembling more diverse architectures than just different scales of YOLOv5 might lead to further improvements. In this regard, architectures from both two-stage and transformer-based methods are worth exploring. Furthermore, more one-stage architectures, such as YOLOv7 and YOLOv8, also represent potential alternatives to test.

Finally, we did not do an extensive hyperparameter search for the problem, opting instead to use the default hyperparameters for YOLOv5 as found in [36]. An extensive search, using methods such as grid search or genetic fitness algorithms [36], could further improve the results shown in Tables 4.1 to 4.6. In particular, we note the different hyperparameters associated with the data augmentation techniques used, shown in Table 3.3. Of note is the the mosaic data augmentation, which is always applied to every example during training. It is clear that this augmentation will significantly affect training due to always being applied. While it is a good form of augmentation [5], it might not be appropriate for the problem at hand, especially to the degree which it is used here. Developing new data augmentations techniques specifically for the task being addressed represents another direction for future research.

## 5.2 Uncertainty based direction

A promising direction for advancing the present work involves further development of the uncertainty estimates associated with the model's output. Currently, our model quantifies only the epistemic uncertainty in the objectness score and the pixels in the segmentation mask. By employing Bayesian deep learning techniques, we could not only refine these epistemic uncertainty estimates but also acquire estimates of aleatoric uncertainty. Improving on the uncertainty estimates of the model is useful, as it would allow for a more symbiotic relationship between the humans and machine when it comes to detecting and segmenting flares. The deep learning based method would handle the bulk of the work, identifying and segmenting obvious flares, while instances marked by high uncertainty estimates would signal the need for expert human review. Furthermore, we show how to obtain ground truth uncertainties on segmentation masks and confidence scores on the experts' predictions in Section 4.5. It is uncommon to have good ground truths on uncertainties in deep learning, but through this method we could obtain quantities which we could use to benchmark the performances of the Bayesian based deep learning model.

## 5.3 Discerning useful flares

As mentioned in the introduction, there are two goals of interest for researchers: detecting whether the flare is useful or not, and if useful, segment it. As a recap, a useful flare is a flare which has roots to the seafloor, or more specifically, it has a strong target strength (more reddish color in the echogram) in the 5 to 10 meter layer above the seafloor. We show an illustration of a useful or not flare in Figure 1.2. In this thesis we focused on the detection of flares part, but now consider ways to incorporate a distinction between the two types of flares.

We first consider the simplest method to discern between types of flares: calculating the average pixel value of the flare in the 5 to 10 meter layer and comparing it with a threshold. If the average is sufficiently high, it indicates that the target strength is high, suggesting the presence of useful data in the 5 to 10 meter layer. However, there are edge cases where this method might fail, such as when a fish or other disturbance enters the 5 to 10 meter layer of the flare, resulting in a falsely high average pixel value. To address this, we could implement additional checks following the average pixel value check. One such check could involve analyzing the average variation of neighboring pixels within the 5 to 10 meter layer of the flare. In instances of disturbances, like a fish, there would likely be higher variation among the pixels. Oppositely,

a pure flare is expected to exhibit smoother transitions between pixels.

Another approach to differentiating between flare types involves explicitly labeling the flares as either useful or not useful and then training the model to predict the correct class in addition to performing segmentations. An alternative variation involves a post-processing check, similar to the method described in the previous paragraph, but in this case, the check is learned by a smaller CNN. The input to this CNN would exclusively be the data from the 5 to 10 meter layer of the flare, where detecting and segmenting the flare is learned by another model. Due to its smaller scope, this CNN could be relatively lightweight and require significantly less inference time compared to the larger flare detection and segmentation network. Of the two deep learning based methods, this post-processing CNN appears to be the better option. It allows us to directly inject the knowledge that only the 5 to 10 meter layer of the flare contains relevant information for determining its usefulness, which a model trained to simultaneously output classes and segmentations cannot know. Because we have seen that interpreting and labeling data is highly subjective, we would need to be extra careful in the case where we collect data on useful versus not useful flares.

## 5.4 Concluding remarks

To our knowledge, this thesis presents the first work on instance segmentation for gas flares in single beam echosounder data. To this end, we have created a dataset consisting of 1,414 images, with 5,142 objects identified as flares and segmented. We demonstrate how to adapt the Brier score and WBF for use in instance segmentation. A quantitative analysis is performed on ensemble models using the Brier and mAP metrics, where we find that the ensemble models perform as well as the average human expert. However, our qualitative analysis uncovers remaining challenges with the ensemble models that need improvement to achieve human-level performance. Finally, we detail potential directions and improvements for the current work and note that, if implemented, these could bridge the gap between human and machine-level performance.

# Bibliography


[1] Ethem Alpaydin. *Introduction to Machine Learning*. 3rd ed. MIT Press, 2014. ISBN: 9780262028189.

[2] Sebastian Bach et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation." In: *PLOS ONE* 10.7 (July 2015), pp. 1–46. DOI: 10.1371/journal.pone.0130140. URL: https://doi.org/10.1371/journal.pone.0130140.

[3] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG].

[4] Yoshua Bengio et al. "Greedy Layer-Wise Training of Deep Networks." In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. NIPS'06. Canada: MIT Press, 2006, pp. 153–160.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[6] Navaneeth Bodla et al. *Soft-NMS – Improving Object Detection With One Line of Code*. 2017. arXiv: 1704.04503 [cs.CV].

[7] Daniel Bolya et al. *YOLACT: Real-time Instance Segmentation*. 2019. arXiv: 1904.02689 [cs.CV].

[8] GLENN W. BRIER. "VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY." In: *Monthly Weather Review* 78.1 (1950), pp. 1–3. DOI: https://doi.org/10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2. URL: https://journals.ametsoc.org/view/journals/mwre/78/1/1520-0493_1950_078_0001_vofeit_2_0_co_2.xml.

[9] George Casella and Roger L. Berger. *Statistical Inference*. 2nd ed. Pacific Grove: Duxbury Press, 2002.

[10] Anna Choromanska et al. *The Loss Surfaces of Multilayer Networks*. 2015. arXiv: 1412.0233 [cs.LG].

[11] Floriana Ciaglia et al. *Roboflow 100: A Rich, Multi-Domain Object Detection Benchmark*. 2022. arXiv: 2211.13523 [cs.CV].

[12] Yann Dauphin et al. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*. 2014. arXiv: 1406.2572 [cs.LG].

[13] J. Deng et al. “ImageNet: A large-scale hierarchical image database.” In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. Miami, FL, USA, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[14] K. O. Dølven. “Methane in the Arctic Ocean: Legal and Scientific Aspects of Seabed Seepage.” Doctoral dissertation. UiT The Arctic University of Norway, 2023. URL: https://munin.uit.no/handle/10037/24357.

[15] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[16] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[17] B. Dwyer, J. Nelson, J. Solawetz, et al. *Roboflow (Version 1.0)*. https://roboflow.com. Computer Vision Software. 2022.

[18] J. Friedrich. “Seasonal Variability of Methane Seep Distribution and Intensity Offshore Western Svalbard at the Edge and Outside the Gas Hydrate Stability Zone.” Master’s Thesis. MA thesis. Kiel University, 2021.

[19] M.A. Ganaie et al. “Ensemble deep learning: A review.” In: *Engineering Applications of Artificial Intelligence* 115 (Oct. 2022), p. 105151. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2022.105151. URL: http://dx.doi.org/10.1016/j.engappai.2022.105151.

[20] Jakob Gawlikowski et al. *A Survey of Uncertainty in Deep Neural Networks*. 2022. arXiv: 2107.03342 [cs.LG].

[21] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[22] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks.” In: *International Conference on Artificial Intelligence and Statistics*. 2010. URL: https://api.semanticscholar.org/CorpusID:5575601.

[23] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[24] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[25] Priya Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2018. arXiv: 1706.02677 [cs.CV].

[26] Moritz Hardt and Tengyu Ma. *Identity Matters in Deep Learning*. 2018. arXiv: 1611.04231 [cs.LG].

[27] Nazanin Sadat Hashemi et al. *Template Matching Advances and Applications in Image Analysis*. 2016. arXiv: 1610.07231 [cs.CV].

[28] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[29] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[30] Kaiming He et al. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].

[31] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 346–361. ISBN: 9783319105789. DOI: 10.1007/978-3-319-10578-9_23. URL: http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[32] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[33] Shuyu Hu et al. "Underwater gas leak detection using an autonomous underwater vehicle (robotic fish)." In: *Process Safety and Environmental Protection* 167 (2022), pp. 89–96. ISSN: 0957-5820. DOI: https://doi.org/10.1016/j.psep.2022.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0957582022007662.

[34] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[35] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[36] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. May 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[37] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[38] Rohan Kashyap. *A survey of deep learning optimizers – first and second order methods*. 2023. arXiv: 2211.15596 [cs.LG].

[39] Nitish Shirish Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. arXiv: 1609.04836 [cs.LG].

[40] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[41] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[43] Chuyi Li et al. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. arXiv: 2209.02976 [cs.CV].

[44] Hao Li et al. *Visualizing the Loss Landscape of Neural Nets*. 2018. arXiv: 1712.09913 [cs.LG].

[45] Hongzhou Lin and Stefanie Jegelka. *ResNet with one-neuron hidden layers is a Universal Approximator*. 2018. arXiv: 1806.10909 [cs.LG].

[46] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[47] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[48] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[49] Shu Liu et al. *Path Aggregation Network for Instance Segmentation*. 2018. arXiv: 1803.01534 [cs.CV].

[50] Wei Liu et al. "SSD: Single Shot MultiBox Detector." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2016, pp. 21–37. ISBN: 9783319464480. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[51] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[52] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG].

[53] Wenyu Lv et al. *DETRs Beat YOLOs on Real-time Object Detection*. 2023. arXiv: 2304.08069 [cs.CV].

[54] M. E. Mann. *Greenhouse Gas*. https://www.britannica.com/science/greenhouse-gas. Accessed: December 6, 2023. 2023.

[55] Dominic Masters and Carlo Luschi. *Revisiting Small Batch Training for Deep Neural Networks*. 2018. arXiv: 1804.07612 [cs.LG].

[56] D. McKay and A. Kvammen. "Auroral classification ergonomics and the implications for machine learning." In: *Geoscientific Instrumentation, Methods and Data Systems* 9.2 (2020), pp. 267–273. DOI: 10.5194/gi-9-267-2020. URL: https://gi.copernicus.org/articles/9/267/2020/.

[57] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[58] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[59] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[60] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[61] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[62] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[63] Yazhou Ren et al. *Deep Clustering: A Comprehensive Survey*. 2022. arXiv: 2210.04142 [cs.LG].

[64] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[65] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: 1805.11604 [stat.ML].

[66] Robin M. Schmidt, Frank Schneider, and Philipp Hennig. *Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers*. 2021. arXiv: 2007.01547 [cs.LG].

[67] Andrew Shepley, Greg Falzon, and Paul Kwan. *Confluence: A Robust Non-IoU Alternative to Non-Maxima Suppression in Object Detection*. 2022. arXiv: 2012.00257 [cs.CV].

[68] Manu Siddhartha and Avik Santra. *COVIDLite: A depth-wise separable deep neural network with white balance and CLAHE for detection of COVID-19*. 2020. arXiv: 2006.13873 [eess.IV].

[69] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[70] Roman Solovyev, Weimin Wang, and Tatiana Gabruseva. "Weighted boxes fusion: Ensembling boxes from different object detection models." In: *Image and Vision Computing* 107 (Mar. 2021), p. 104117. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2021.104117. URL: http://dx.doi.org/10.1016/j.imavis.2021.104117.

[71] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].

[72] Ilya Sutskever et al. "On the Importance of Initialization and Momentum in Deep Learning." In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–1139–III–1147.

[73] Mingxing Tan, Ruoming Pang, and Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2020. arXiv: 1911.09070 [cs.CV].

[74] Juan Terven and Diana Cordova-Esparza. *A Comprehensive Review of YOLO: From YOLOv1 and Beyond*. 2023. arXiv: 2304.00501 [cs.CV].

[75] Y Tian, Y Zhang, and H Zhang. "Recent Advances in Stochastic Gradient Descent in Deep Learning." In: *Mathematics* 11.3 (2023), p. 682. DOI: 10.3390/math11030682. URL: https://doi.org/10.3390/math11030682.

[76] J.R.R. Uijlings et al. "Selective Search for Object Recognition." In: *International Journal of Computer Vision* (2013). DOI: 10.1007/s11263-013-0620-5. URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[77] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[78] M. Veloso et al. “A new methodology for quantifying bubble flow rates in deep water using splitbeam echosounders: Examples from the Arctic offshore NW-Svalbard.” In: *Limnology and Oceanography: Methods* 13.6 (2015), pp. 267–287.

[79] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV].

[80] Chien-Yao Wang et al. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. arXiv: 1911.11929 [cs.CV].

[81] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV].

[82] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015. arXiv: 1505.00853 [cs.LG].

[83] Suorong Yang et al. *Image Data Augmentation for Deep Learning: A Survey*. 2023. arXiv: 2204.08610 [cs.CV].

[84] Jason Yosinski et al. *How transferable are features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG].

[85] Kaichao You et al. *How Does Learning Rate Decay Help Modern Neural Networks?* 2019. arXiv: 1908.01878 [cs.LG].

[86] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

[87] Chiyuan Zhang et al. *Understanding deep learning requires rethinking generalization*. 2017. arXiv: 1611.03530 [cs.LG].

[88] Huaqing Zhang et al. *Compiler-Level Matrix Multiplication Optimization for Deep Learning*. 2019. arXiv: 1909.10616 [cs.LG].

[89] Zhaohui Zheng et al. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. 2019. arXiv: 1911.08287 [cs.CV].