F-SAL: A Framework for Fusion Based Semi-automated Labeling With Feedback

F-SAL: A FRAMEWORK FOR FUSION BASED SEMI-AUTOMATED LABELING WITH FEEDBACK

BY

AHMED ZAIDI, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2021) McMaster University
(Electrical and Computer Engineering) Hamilton, Ontario, Canada

TITLE:	F-SAL: A Framework for Fusion Based Semi-automated Labeling With Feedback
AUTHOR:	Ahmed Zaidi B.Eng. (Electrical Engineering), McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. T. Kirubarajan
NUMBER OF PAGES:	xiii, 108

Abstract

In almost all computer vision and perception based applications, particularly with camera and lidar; state-of-the-art algorithms are all based upon deep neural networks which require large amounts of data. Thus, the ability to label data accurately and quickly is of great importance. Approaches to semi-automated labeling (SAL) thus far have relied on using state-of-the-art object detectors to assist with labeling; however, these approaches still require a significant number of manual corrections. Surprisingly, none of these approaches have considered labeling from the perspective of multiple diverse algorithms. In this thesis a new framework for semi-automated labeling is presented, it is called F-SAL which stands for Fusion Based Semi-automated Labeling. Firstly, F-SAL extends on the idea of SAL through introducing multi-algorithm fusion with learning based feedback. Secondly, it incorporates new stages such as uncertainty evaluation and diversity evaluation. All the algorithms and design choices regarding localization fusion, label fusion, uncertainty and diversity evaluation are presented and discussed in significant detail. The biggest advantage of F-SAL is that through the fusion of algorithms, the number of true detections is either more or equivalent to the best single detector; while the false alarms are suppressed significantly. In the case of a single detector, to lower the false alarm rate, detector parameters must be adjusted, which trade lower false alarms for fewer detections. With F-SAL, a lower false alarm rate can be achieved without sacrificing any detections, as false alarms are suppressed during fusion, and true detections are maximized through diversity. Results on several datasets for image and lidar data show that F-SAL outperforms the single best detector in all scenarios.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Thia Kirubarajan for initially accepting me as an undergraduate researcher in May 2018, then later allowing me to do software development throughout my Bachelor's and Master's degrees. However, most importantly, I would like to thank him for letting me work on a problem I truly enjoyed. Secondly, I would like to thank Dr. Ratnasingham Tharmarasa for his advice and assistance related to software development, and also for his excellent course on multi-target tracking; I learned a great deal in that course. Third, I would like thank the administrative staff in the department of Electrical & Computer Engineering (especially Cheryl Gies and Tracey Coop). Fourth, a special thanks goes out to all the friends I have made at McMaster University over the years, my time here would not have been anywhere as great as it was without these brilliant individuals. Last but not least, a special thanks also goes out to my family.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Abbreviations

AI	Artificial intelligence
SAL	Semi-automated Labeling
F-SAL	Fusion based Semi-automated
PMF	Probability mass function
PDF	Probability density function
MLP	Multilayer perceptron
NN	Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
SSD	Single Shot Detector

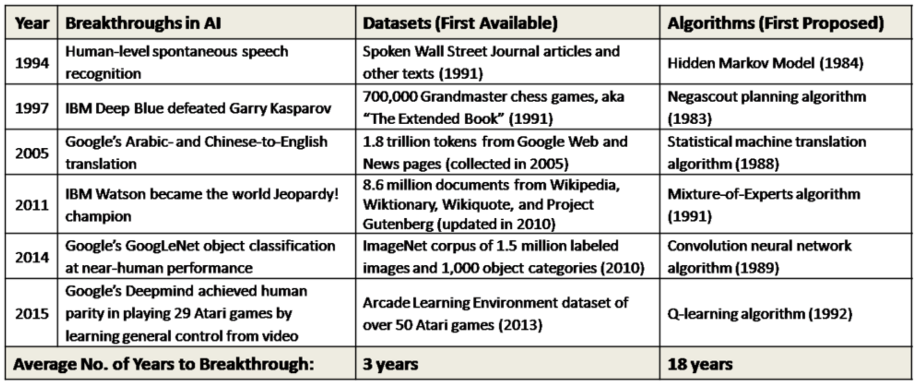
FOV	Field of view
IOU	Intersection over union
FC	Fully connected layer
RPN	Region proposal network
NMS	Non-maximum suppression
LUT	Look-up table

Chapter 1

Introduction

1.1 The Importance of Data

The advancements in artificial intelligence within the past decades have been extraordinary. Similarly, impressive advancements were made in computer vision in the last decade. An algorithm for classification first proposed in 1989; the Convolutional Neural Network (CNN) [15] trained through back propagation, began replacing essentially all of classical computer vision in 2012. However, what is more interesting is why this algorithm's performance went unutilized for 23 years. At the time, only a select few companies had the proper hardware to implement neural networks, and the datasets required to train such algorithms did not exist. It was not until the year 2010, that an enormous dataset called ImageNet [4] was released; which contained 1.5 million labeled images with 1,000 object categories. This dataset, along with the advancements in GPUs, is what allowed for the deep learning revolution to begin and provided empirical proof for deep neural networks. Nonetheless, this trend is not specific to only the CNN model. In fact, in figure 1.1, one can see a general trend throughout history where the average number of years for a breakthrough with respect to datasets is three years, while for novel learning algorithms this is 18 years.



Year	Breakthroughs in AI	Datasets (First Available)	Algorithms (First Proposed)
1994	Human-level spontaneous speech recognition	Spoken Wall Street Journal articles and other texts (1991)	Hidden Markov Model (1984)
1997	IBM Deep Blue defeated Garry Kasparov	700,000 Grandmaster chess games, aka "The Extended Book" (1991)	Negascout planning algorithm (1983)
2005	Google's Arabic- and Chinese-to-English translation	1.8 trillion tokens from Google Web and News pages (collected in 2005)	Statistical machine translation algorithm (1988)
2011	IBM Watson became the world Jeopardy! champion	8.6 million documents from Wikipedia, Wiktionary, Wikiquote, and Project Gutenberg (updated in 2010)	Mixture-of-Experts algorithm (1991)
2014	Google's GoogLeNet object classification at near-human performance	ImageNet corpus of 1.5 million labeled images and 1,000 object categories (2010)	Convolution neural network algorithm (1989)
2015	Google's Deepmind achieved human parity in playing 29 Atari games by learning general control from video	Arcade Learning Environment dataset of over 50 Atari games (2013)	Q-learning algorithm (1992)
Average No. of Years to Breakthrough:		3 years	18 years

Figure 1.1: A comparison of the time required for a breakthrough in artificial intelligence tasks with respect to datasets and algorithms. Image taken from [40]

This is not a coincidence, especially not within the machine learning domain. Classical learning algorithms, which are now mostly referred to as base models; simply were not able to fully utilize even the limited data at the time for tasks such as image classification or speech recognition. Figure 1.2 shows a comparison of classical algorithms and modern deep neural networks. Evidently, the success of deep neural networks is possible because this limitation does not apply to them, more data simply increases their generalization capability.

For researchers who work in pattern recognition and with perception algorithms, many sensors such as camera and lidar now obtain state-of-the-art results for detection and tracking through deep learning models, which can run in real-time and on critical hardware. However, neural network architectures are getting even deeper now, and new novel models are requiring even more data than the enormous architectures from a few years back, and this trend will only continue into the future.

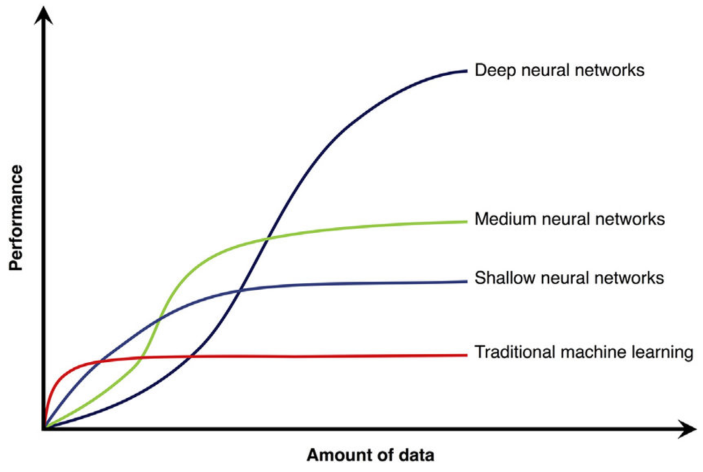


Figure 1.2: Classical learning algorithms plateau after a certain point and cannot make use of the additional data we provide to them. On the other hand, more data leads to deep learning algorithms being able to perform even better. Image taken from [37]

It is evident that data is of critical importance and will play an even bigger role in the future of technology. There seems to be an enormous interest in algorithm design, however the problem of how to obtain labeled data efficiently has not attracted as much attention. In fact, getting data is not always challenging, for example, to obtain data for perception algorithms, the only requirement is to keep the sensor on. However, this raw data is of no importance; labeled data is what is desired.

1.2 The Bottleneck of Modern Algorithms

Unfortunately, labeled data is challenging to obtain, while labeled data will continue to be critical for all future models; obtaining labeled data will be the bottleneck of all future models. Labeling is challenging because it is slow, expensive and requires human labour. Figure 1.3 took the author 2 minutes to label, and the dataset it originates from has hundreds of thousands of such images. Moreover, this is a very simple scene as a rectangle bounding box is used. In some labeling scenarios, polygon bounding boxes are needed, and they contain 100s of vertices; labeling such a scene would take significantly longer. For an application such as autonomous vehicles, the perception algorithms used require a lot of labeled data, and need to be frequently updated to handle edge cases. To maintain robustness, data from multiple cities, diverse climates/weather, different times of the day and with diverse scene types is a necessity. The time commitment for such a task is enormous, and for this reason labeling seems to be the bottleneck when developing learning algorithms.

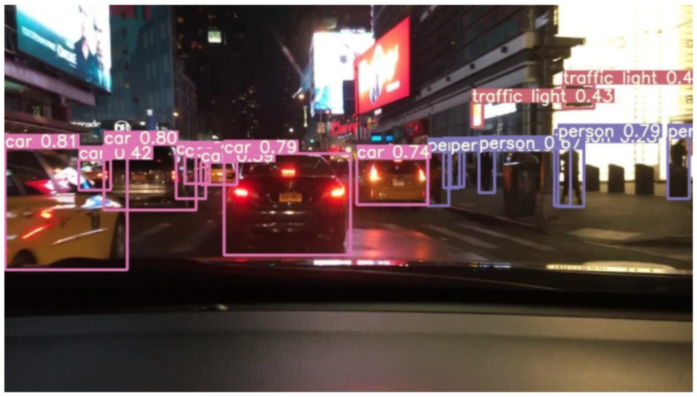


Figure 1.3: A labeled scene with a large number of objects.

1.3 Existing Methods for Labeling

In order to reduce the time spent labeling, two strategies are possible. The first is to somehow reduce the number of training examples needed to train a model without effecting its accuracy. Trivially, fewer examples means less time spent labeling. This is possible through semi-supervised learning approaches such as active learning. The second is to rely on labeling algorithms, and then make corrections on their outputs.

For example, an object detector was able to label the scene in figure 1.3 in only 12ms, and with only 4 false alarms.

1.3.1 Active Learning

Semi-supervised learning combines both unsupervised and supervised learning where there is labeled data and some unlabeled data. If the objective is to generate labels to train a specific algorithm, a form of semi-supervised learning known as Active Learning [27] may be beneficial. When considering training a model, it is often from the perspective of passive learning. In passive learning,there is an implicit assumption that all labels contain the same amount of information. As in, there is no strategy to explicitly decide which labels are useful or which are not to the model being trained. One simply labels some data related to objects of interest, and then trains the model with them. However, active learning is based upon the principle that all labels are not created equally and they contain different amounts of information. When one explicitly selects labels that maximize the information gain, models perform better with less training data. Information gain in this context can be interpreted in many ways, however high uncertainty is the general idea. This strategy is successful because the model focuses on difficult objects, which ultimately leads to it generalizing better with less data. If an algorithm has 95% accuracy with cars but only 60% accuracy with trucks, it does not require more examples of cars, instead it should be provided more examples of trucks so it may generalize better. Figure 1.4 shows a plot which compares active learning and passive learning. Notice how with only 45% of data, an algorithm trained using active learning can achieve 30% more accuracy than a passive learning approach.

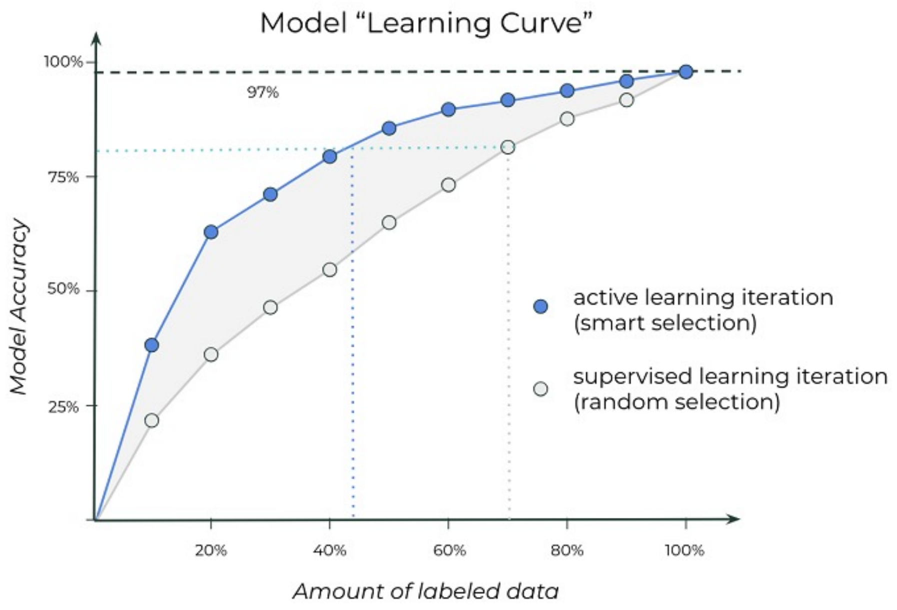


Figure 1.4: A comparison of Active and Passive Learning. Image Source: https://www.kdnuggets.com (permission was obtained from the author)

Active Learning has several variants, but the most popular and successful is iterative pool-based sampling. In this method queries from a set of unlabeled instances Q are selected, and probabilistic techniques are often applied to select some labels from the unlabeled set that the model believes have high uncertainty. The general architecture for this can be seen in Figure 1.5, while the general pipeline is as follows:

1. We initialize by having a small amount of labeled data in a set L with which we train the current model.

2. We then select a random subset of unlabeled examples (Q) and evaluate each element in this set and obtain its corresponding label distribution from the current model.

3. Using an information gain measure on the distributions of each of the elements

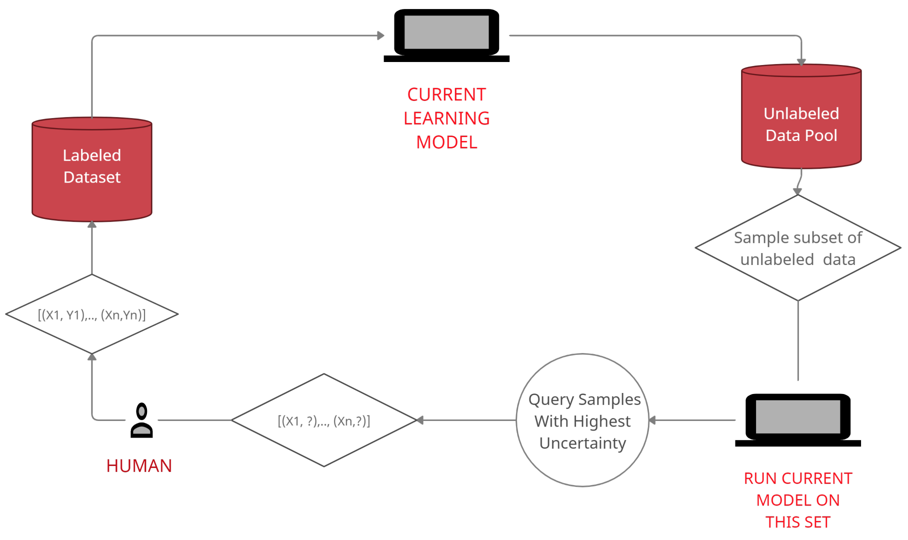


Figure 1.5: Pool-based Active Learning Pipeline

of the set Q, we create a subset of highly uncertain examples that the model was unsure about.

4. The model then queries a human to label these examples, any examples it has strong certainty about are discarded.

5. After a human labels these examples they are added to the set L and we retrain our model and repeat steps 1 – 4 again.

6. We continue this approach until stopping criteria. This usually means we run out of labels, or see a plateau in an accuracy vs data plot.

There are several techniques in the literature for assessing uncertainty with labels. One technique called uncertainty sampling assigns a measure to each candidate object based upon that objects label distribution using an entropy-based measure. For example, in Figure 1.6, the model is certain that object 1 is a car so information gain is minimal (high entropy), however, object 2 has a very large uncertainty with it as both Car and Truck have a similar confidence score (low entropy). There is another strategy called query-by-committee (QBC) which is also based upon probabilistic methods but requires multiple models. It searches for uncertainty through disagreement by experts. Density weighted methods also exist, these use either uncertainty sampling or query-by-committee as a base method but add an additional weight for informativeness of an example using an input distribution related to the set Q.

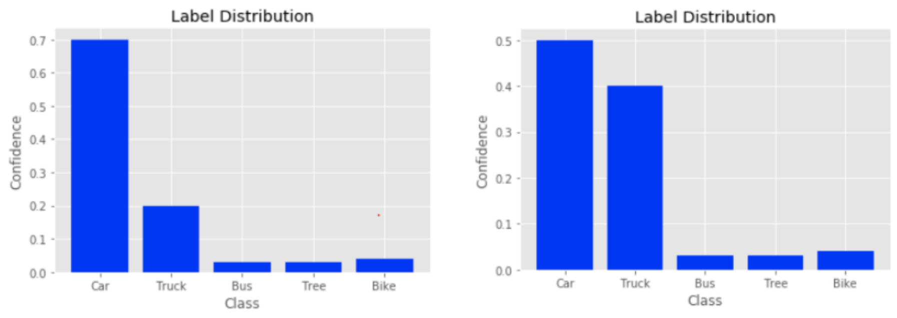


Figure 1.6: **Left**: High certainty so low information gain, **Right**: Low certainty so high information gain

Active learning assumes that training data is easy to obtain with respect to the labeling process itself. For image classification tasks where there is one object in one frame this works well as no localization is needed. However, to train detection and tracking algorithms one must label complex scenes where a single frame contains dozens of objects. This is where the problem occurs, as objects need to be localized in a scene before they can be classified. Therefore, fundamentally, active learning is not applicable to tasks outside of trivial image classification and could not be used to label a scenario such as figure 1.3. Another problem with active learning is that it proceeds in rounds and the model must be retrained at every iteration, this is computationally difficult for deep neural networks (DNN). The above being said, significant research has been conducted over the years to unify DNNs into a framework so that Active Learning may be used with negligible training times [29][26] and more recently [42]. However, the research so far has shown only marginal improvements, and it seems that Active Learning is not as effective for DNNs as it is for classical learning algorithms.

1.3.2 Semi-automated Labeling

Consider a scenario where perception algorithms are unable to generalize to the driving conditions in Northern Canada. In this case, the system must be updated with more training examples pertaining to the winter conditions there. If one is interested in labeling cars, pedestrians, and traffic signs in a new dataset, and has algorithms from a previous one that can detect the same classes to some extent, then using them reduces the time spent labeling. The ideal situation for labeling is when human level ground truth accuracy can be maintained with the lowest possible time commitment. Humans can achieve near perfect accuracy for most objects easily but are too slow. On the contrary, algorithms can label much quicker but have a probability associated with their accuracy. The idea of using both human and algorithm to conduct labeling is referred to as semi-automated labeling (SAL). This approach is far superior to active learning as the algorithms used in SAL can also handle localization.

The most extensive evaluation of SAL so far was conducted by UC Berkeley, Georgia Institute of Technology, Peking University and Uber AI Labs in [43]. The authors had human annotators initially label 55, 000 clips by hand. Then they trained a Fast-RCNN object detector on this data. Subsequently, they asked the human annotators to use this detector to assist in labeling the remaining data. Ultimately, they recorded that through using SAL, the time a human annotator spent labeling and adjusting for corrections was reduced by 60%. Moreover, in [3] the authors were interested in the idea of SAL in industrial robotics. They had a 2D camera mounted on a robot and used it to generate two datasets using a YOLO object detector for one dataset, and an SSD for the other. What would have taken a human 10 hours to label, took their algorithm less than one hour. In [1], SAL was applied for generating polygon labels (see section 2 instance segmentation). Here a human would select regions of interest to apply a polygon segmentation algorithm on, and the algorithm would generate a polygon bounding box around that object. Additionally, whenever the human made corrections on an object, all the vertices would be readjusted. This idea of using algorithms to do initial labeling, then having a human make corrections also seems popular outside of academia, as several popular enterprise labeling tools and open source implementation are based off of this.

1.4 Semi-automated Labeling Motivation

In this thesis, the focus is on improving SAL. As stated previously, labeling algorithms are probabilistic; hence they produce errors. More specifically these errors are:

1. **Inaccurate Localization**: The output position is noisy and needs to be adjusted, but this output is still correct and has the appropriate class.
2. **Incorrect Classification**: An algorithm localized an object correctly, but provided the wrong label (e.g. a car was labeled as a bus).

3. **Missed detection**: Information in the new dataset that the algorithm could not generalize too.

4. **False Alarm**: Algorithm incorrectly localizes and classifies background objects.

These errors can be ranked in order of time required to correct them as: Missed Detections, Incorrect Localization, Incorrect Classification and False Alarms. Missed detections are usually the worst scenario because one must label from scratch – meaning time is spent looking for the object in a scene and placing a bounding box around it. In the other scenarios, the objects are localized, and either the coordinates need to be adjusted, a label needs to be changed, or the bounding box needs to be deleted. Figure 1.7 shows a visual of these errors.

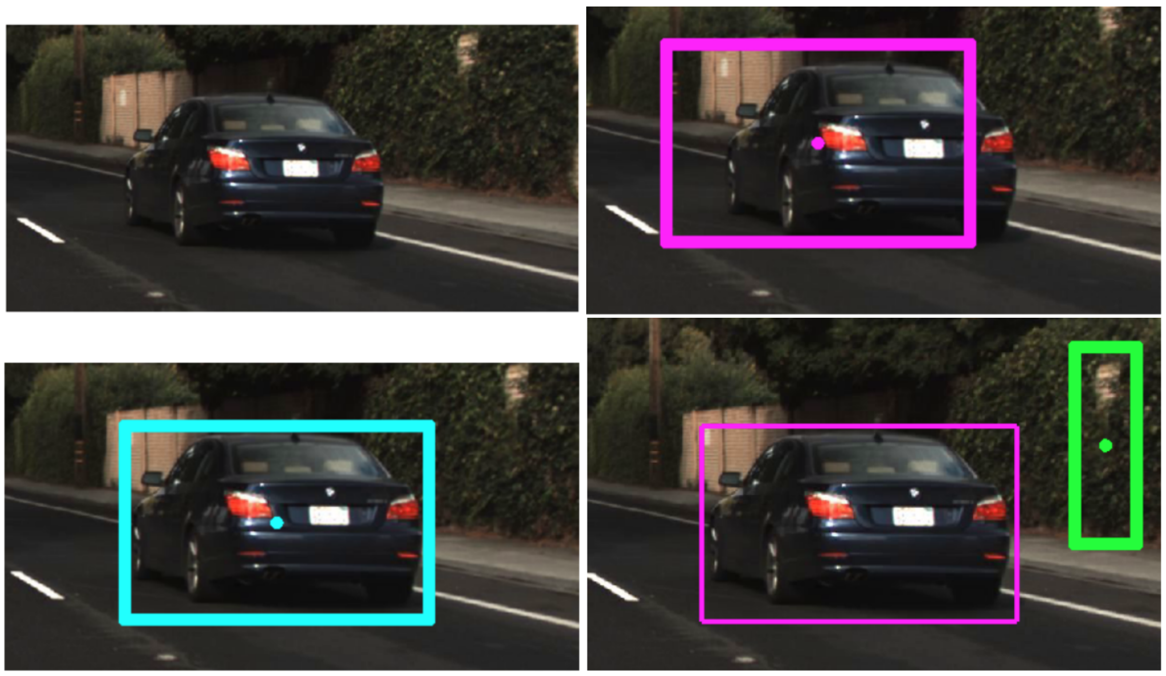


Figure 1.7: From the top, the first image is a missed detection. The second is an incorrect localization, the third is an incorrect classification, and the fourth is a false alarm. A purple bounding box corresponds to a car. A green bounding box corresponds to a pedestrian. A blue bounding box corresponds to a truck.

Denote the # of frames in a dataset as n_{frames}, the average number of objects in a frame as n_{av} and the average time required to label an object as t_{av}; then an expression for how long it takes a human to label a dataset can be written.

$$T_{Human} = \frac{t_{av} * n_{frames} * n_{av}}{3600} \tag{1.4.1}$$

Moreover, an expression for the time taken by a labeling algorithm is trivial and is just the number of frames divide by the algorithm's inference frame rate.

$$T_{Algorithm} = \frac{n_{frames}}{FPS * 3600} \tag{1.4.2}$$

Often a labeling algorithm has between 50% and 65% accuracy on a new dataset, and the time taken to run this algorithm is several magnitudes lower than labeling by hand. However, the fundamental question here is how much time is required to go from this 50-65% accuracy to 100% accuracy through human corrections. This is equal to the time needed for a labeling algorithm to run on the dataset, as well as the time required for a human to correct the four sources of errors mentioned above.

$$T_{SAL} = \frac{n_{frames}}{FPS * 3600} + \frac{(n_{el}t_{el} + n_{ec}t_{ec} + n_{md}t_{md} + n_{fa}t_{fa})}{3600} \tag{1.4.3}$$

However, one should keep in mind that the average time taken to label a bounding box, or correct an existing one, is extremely subjective and depends on what sensor the data was generated from (camera, lidar, etc..), the type of bounding box needed (rectangle, polygon, cuboid, etc..), and even the content of the dataset itself, as labeling frames from a wide angle high resolution satellite as opposed to dash camera data is vastly different.

Using SAL, more time is spent labeling than a stand-alone algorithm as human corrections are needed, however, far less time is spent than a lone human labeler. The advantage with SAL is that human level accuracy is obtained but at a fraction of the time. Table 1.1 compares the accuracy/time trade-off and summarizes these results. The results in the table assume a naive example where an algorithm with an accuracy of 65% and inference time of 5 FPS is used. $N_{frames} = 15000, n_{av} = 10, t_{av} = 7.5s$. For simplicity and just to obtain a ballpark number, assume that all error corrections require the same amount of time as t_{av}. In reality this is not a practical assumption, because t_{av} is the amount of time taken to label an example from scratch, which is normally the most amount of time it would take to apply a correction - therefore the number below actually represents the worst case scenario.

Type	Accuracy	Time Taken	Numerical Example
Ideal Labeler	100%	-	0 hours
Human Labeler	100%	Very Long	312 Hours
Labeling Algorithm	50 - 65%	Relatively Quick	50 minutes
Semi-automated Labeling	$\approx$ 100%	$T_{Algorithm} \leq T_{SAL} \leq T_{Human}$	109 hours

Table 1.1: Accuracy vs Speed comparison of different labelers.

1.5 Contribution and Layout

In this thesis a brand-new framework which incorporates semi-automated labeling is presented. It is fusion based semi-automated labeling or F-SAL. It extends upon the idea of SAL but introduces several new components. From this, a multi-stage end-to-end framework for labeling, which allows for more accurate and robust labels than traditional SAL is derived. Unlike all existing methods, F-SAL allows for the fusion of multiple object detectors as opposed to using just one. The author empirically proves that diversity between object detectors exists in both camera and lidar sensors and can be utilized for object detector fusion. Due to the extension to multiple algorithms, localization and label fusion are introduced. From the general taxonomy of F-SAL, two configurations are presented and each is based off of a different localization fusion algorithm. Likewise, in the case of label fusion, the fact that an algorithm and human label together means that a real-time dataset that reflects the reliability of each individual algorithm (from human corrections) is always available. This is directly used to introduce trainable feedback into the label fusion stage overtime which allows the F-SAL system to label better as more corrections occur. Borrowed from uncertainty sampling techniques from active learning, an uncertainty evaluation stage is introduced which operates directly upon the label distributions from the label fusion stage. This allows the F-SAL system to tag objects with high uncertainty ahead of time and simply query a human labeler to label this example. Finally, since multiple algorithms are used, a diversity evaluation stage is introduced, this is essentially to manage the algorithms optimally. This stage monitors the diversity between algorithms, and if there is low diversity it prompts a human asking if they would like to prune such algorithms. Ultimately, it is shown that F-SAL outperforms stand-alone object detectors in terms of accuracy as well as robustness on all datasets presented.

In section 2 the background knowledge needed to understand the field of labeling is presented. In section 3 the F-SAL taxonomy with each of its individual components and their theoretical pinnings will be discussed in detail. In section 4 two configurations in which F-SAL can be used will be presented. These will be evaluated on two camera and two lidar datasets, and the significance of the results will be discussed. This thesis will conclude with section 5 which briefly mentions future work.

Chapter 2

Background

2.1 Types of Labeling

Labeled data is required to create datasets which are used to train machine learning algorithms. There are several ways in which one can label depending on the problem. More generally, the goal in labeling is to assign a bounding box to each object in a sensor's field of view. A bounding box is a structure which contains both location and classification information about an object. This information is then stored into a dataset, which is ultimately used to train a learning algorithm.

A bounding box can have several topologies. Some common bounding boxes are rectangles, polygons and cuboids. Although any custom bounding box can be used, the theoretical work for loss functions associated with these bounding boxes during model training is very well studied. The simplest, most used and well known one is the 2D rectangle bounding box, which is used on image data. It contains the label, the confidence of the classifier on that label, as well as location information. More specifically, it stores a center vertex denoted by (x,y) with a width and height offset, which gives it a rectangle shape.

$$\text{rect} = \begin{bmatrix} \text{label} & \text{confidence} & b_x & b_y & b_h & b_w \end{bmatrix}^T \tag{2.1.1}$$

Likewise, the next most popular bounding box is the 2D polygon bounding box, which also has a label and confidence score; however, it uses a set of n-connected vertices for its shape. This bounding box is very popular for object segmentation tasks.

$$\text{polygon} = \begin{bmatrix} \text{label} & \text{confidence} & x_1 & y_1 & x_2 & y_2 & \dots & x_n & y_n \end{bmatrix}^T \tag{2.1.2}$$

For an application such as detecting vehicles, 2D bounding boxes are usually used when labeling data from image sensors. However, if working with lidar data, then a 3D cuboid bounding box is needed. A 3D cuboid bounding box is an extension to the 2D rectangle bounding box with an additional dimension and axis of rotation. It has 3 coordinates which describe the center vertex (x,y,z) as well as a length, width and height offset along with an orientation angle θ. This bounding box is very popular in camera-lidar fusion as well, and a 3D bounding box can also be overlaid on an 2D image.

$$\text{cuboid} = \begin{bmatrix} \text{label} & \text{confidence} & b_x & b_y & b_z & b_l & b_w & b_h & b_\theta \end{bmatrix}^T \tag{2.1.3}$$

Depending upon the objective, several labeling scenarios are possible. The four most popular types of labeling scenarios are:

1. Image Classification
2. Object Detection
3. Semantic Segmentation
4. Instance Segmentation

Image Classification is the most straightforward. There exists an image of some scene (as a single frame) and a label must be applied to that entire frame. However, in almost all applications, interest is not only in the label of a single object, but both the locations and labels for several objects in a sensor's field of view. This is referred to as **Object Detection**, and it is mostly done through 2D rectangle or 3D cuboid boxes. **Instance Segmentation** is when a label is represented through distinct polygon bounding boxes. In Instance Segmentation the exact mask (polygon shape) around an object is desired, and it must be separated from the background. Additionally, each object is considered independent from another, i.e., if there are two vehicles, they are considered different vehicles. Object detection and instance segmentation are similar, however in instance segmentation every single object is segmented and is distinct, while in object detection only the number of objects per class are known and they are not considered distinct. Moreover, **Semantic segmentation** is when every single pixel in a sensors field of view is associated to a label (even the background). Figure 2.1 shows a comparison of the different types of labeling scenarios visually.

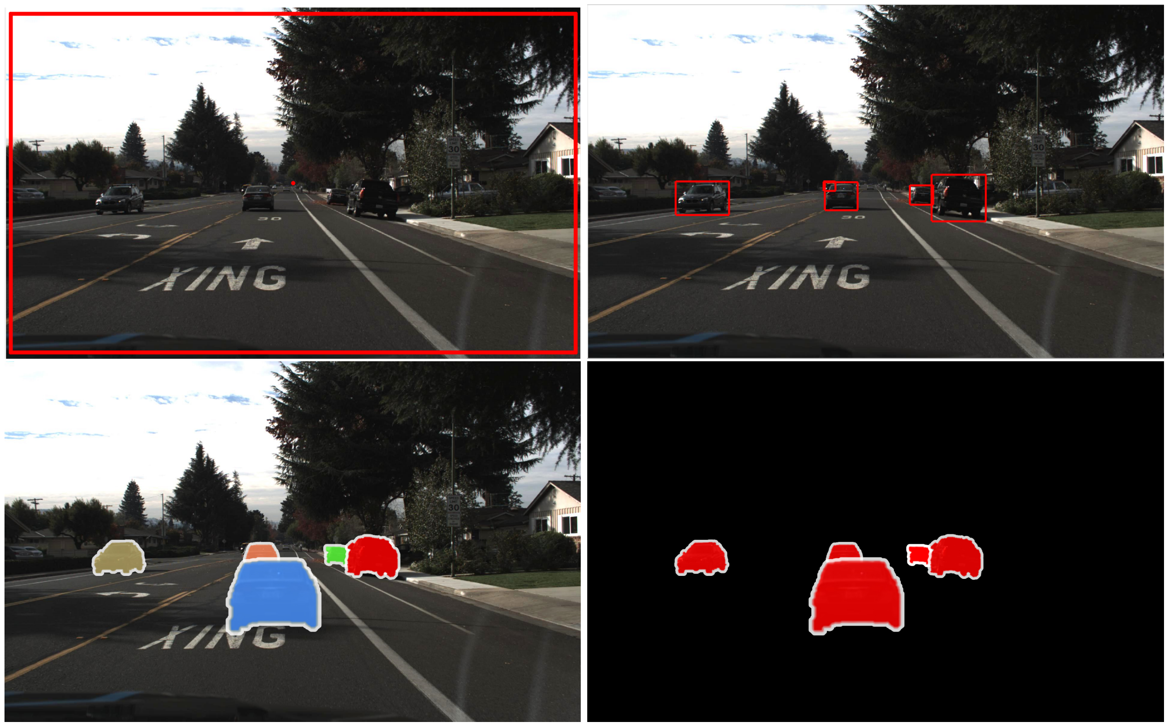


Figure 2.1: Top to bottom: Image Classification, Object Detection, Instance Segmentation, Semantic Segmentation.

Furthermore, labeling can be done with respect to multiple homogeneous (same modality) and heterogeneous (different modality) sensors as well. One very active field of research where heterogeneous labeled data is needed is in autonomous vehicles. In this scenario camera and lidar data is fused to obtain more accurate scene information. Figure 2.2 shows such a scenario.

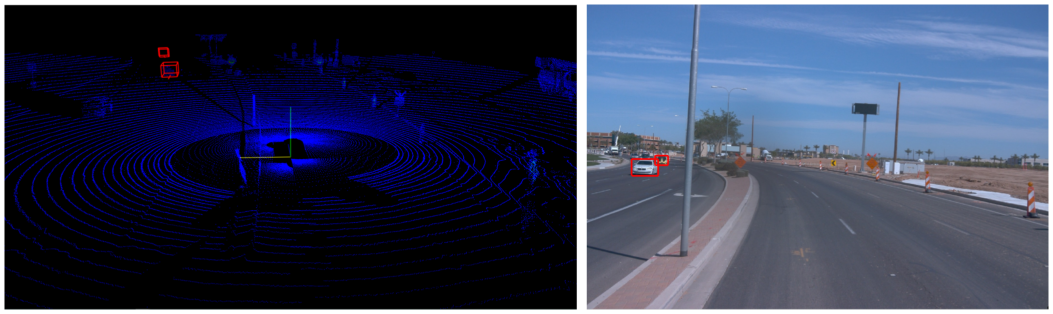


Figure 2.2: Synchronized lidar and camera scan.

2.2 The Theory of Labeling (Bayesian)

As algorithms which conduct labeling have a probability associated with their performance, the problem of labeling can be formulated as a probabilistic theory. Careful attention should be directed towards the definitions introduced here because they are often used interchangeably throughout the rest of this thesis, and the correct distinction depends on the context. The Bayesian framework presented here is from [11].

The task of labeling objects is equivalent to the task of recognizing an object from some information source such as a camera or lidar sensor. Objects may be recognized based upon their features. For example, in most camera sensors, a frame is discretized into a set of pixels over the red, green and blue colour channels. These pixels are referred to as features; while they are initially in a matrix form, they are flattened into a single set of size n that is represented as a feature vector, and the set these features span is called the feature space. All elements in the feature space are numerical values, any non-numerical values must be encoded. Some examples of features are size, shape and colour.

The set of features can be denoted as:

$$X = \{x_1, x_2, \dots, x_n\} \in \mathbb{R}^n \tag{2.2.1}$$

For image sensors, the feature space spans the pixel range 0 to 255 normally, so each element in (2.2.1) would be a value on that range. In scenarios with heterogeneous sensors, the set of features may be partitioned for each sensor and can be denoted as:

$$X_{heterogeneous} = \{X_1, X_2, \dots, X_m\} \tag{2.2.2}$$

Each $X_i \in \mathbb{R}^m$ is a distinct set of features from a different sensor modality, and the values these features can span are dependent on that modality. The terms object and feature are used interchangeably because features physically represent and distinguish objects of different classes.

A class can then be defined as a grouping of objects, which have similar features. Examples of classes are cars, trucks and pedestrians. The terms class and label refer to the same phenomenon, however the term label is used within the context of assignment. In any labeling task, a set of possible labels may be assigned to an object, the elements of this set are referred to as classes. In most recognition tasks there can only be a one-to-one assignment between an object and label. The set of classes can be denoted as:

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\} \tag{2.2.3}$$

As mentioned earlier, in order to train algorithms for inference, labeled data is needed, and to store labeled data, a structure called a dataset is used. A feature matrix is a structure where the number of rows correspond to the number of objects that have been labeled, and the number of columns correspond to the number of features that represent that object (e.g. an 1920 x 1080 image contains 1920 x 1080 x 3 features). Hence, a single row contains all the features belonging to a single object, and each object in a dataset is called a training sample. Next, we define a label vector. The feature matrix has a one-to-one mapping with this vector, which contains the labels associated with every object in this dataset. Side-by-side these two structures form what we call a dataset. The feature matrix is denoted as Z and the associated labels are denoted as Y.

$$\mathrm{Z} = \begin{pmatrix} z_{11} & z_{12} & \ldots & z_{1n} \\ z_{21} & z_{22} & \ldots & z_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ z_{N1} & z_{N2} & \ldots & z_{Nn} \end{pmatrix} \quad \mathrm{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}. \tag{2.2.4}$$

This matrix can also be rewritten in set notation as follows:

$$Z = \{z_1, z_2, \ldots, z_N\}, z_j \in \mathbb{R}^n, y_j \in \Omega, \forall j = 1, 2, ..., N \tag{2.2.5}$$

Here N represents the number of objects in the dataset, n represents the number of features, and the labels are from a predefined set of class labels Ω.

As mentioned earlier, in labeling the objective is to assign labels to objects. A classifier D is defined as a mathematical function that assigns a class label from Ω to some object X based upon this objects n-feature vector.

$$D\colon \mathbb{R}^n \to \Omega \tag{2.2.6}$$

However, to understand this more conceptually, the idea of discriminant functions must first be introduced. Any feature space can be partitioned into C classification regions denoted by:

$$\mathbb{R}^n = \{R_1, R_2, \ldots, R_c\}$$

Through training a classifier on labeled data, these classification regions can be learned. Each classification region has a corresponding discriminant function, which can then be used to assess the likelihood that any new object is associated to it. To classify an object, the object is passed through these C discriminant functions, and the classification region, which maximizes some score (probability) is selected.

$$R_i = \{x \mid x \in \mathbb{R}^n, g_i(x) = \underset{k=1,\ldots,c}{\arg\max} g_k(x)\}$$

Then a label w_i that corresponds to the optimal region R_i is assigned. Therefore, a classifier D is actually a set of C discriminant functions which take a feature vector as input, and output a label distribution. From this distribution, the highest probability output is selected, and all other labels are pruned.

The goal in labeling when using classifiers is to find the true label of an object from the features one believes it contains. Unfortunately, neither a perfect classifier nor dataset exists. A classifier can only be trained with a dataset that contains a finite number of training samples. Additionally, it is possible that two similar objects (car and truck) may never be distinguishable from one another perfectly because the overlap in their feature space is too significant. Therefore, the true goal is to find the optimal label given uncertainty with imperfect features and an imperfect classifier. To achieve this, the problem of labeling can be formulated through a probabilistic framework such as Bayesian estimation. In this case, the posterior probabilities of each class can be viewed as discriminant functions. The advantage of a Bayesian approach is that the error is minimized in an optimal sense when the label with the maximum posterior probability is selected.

Within this framework, the true class label w of an object X is a discrete random variable that may take a value from the set of classes Ω. Assuming a classifier is trained on labeled dataset, one can obtain a prior probability for each label from a distribution based upon the labeled dataset:

$$P(w_j) \in \{p(w_1), ...p(w_c)\}$$

Furthermore, assuming that at most one label may be assigned to a single object, and accounting for all candidate labels $j = 1, \ldots C$ a PMF can be derived for this random variable under the following conditions:

$$0 \leq p(w_i) \leq 1$$
$$\sum_{n=1}^{C} p(w_i) = 1$$

Similarly, a likelihood measure $p(X|w_j)$ based upon the distribution information of the N objects in the labeled dataset can also be obtained. Then from both the prior and likelihood, the posterior probability for a label can be calculated using Bayes theorem.

$$p(w_j|X) = \frac{p(w_j)p(X|w_j)}{\sum_{i=1}^{C} p(w_i)p(X|w_i)} \tag{2.2.7}$$

This represents the probability that the true class label of an object X is w_j. This calculation is done for every candidate label from the set of class labels, and ultimately the label which has the highest posterior probability is selected.

$$P(w_j{}^*|X) = \underset{j=1,\ldots,c}{\arg\max}\{p(w_j)p(X|w_j)\} \tag{2.2.8}$$

We can then substitute the full posterior as the argument, and the denominator in equation 2.2.7 is just a normalization constant that all candidates are scaled by so it has no effect on selecting the optimal label and can be dropped. Alternatively, each posterior probability for a label candidate is actually just a discriminant function, therefore the term inside the maximizing argument can also be rewritten as:

$$P(w_j{}^*|X) = \underset{j=1,\ldots,c}{\arg\max}\{g_i(X)\} \tag{2.2.9}$$

For labeling scenarios in image and lidar sensors, interest is directed towards object detection where there are several objects in a scene. However, the above posterior calculation applies to a single object and assumes it is already localized. Additionally, in its current form, this framework selects its decision from a single algorithm perspective. In F-SAL, multiple algorithms are used, so this framework has to be extended. This is done in the F-SAL label fusion section (3.4) later in this thesis.

2.3 Algorithms for Labeling

In section 2.1 the different types of labeling scenarios were discussed, and in section 2.2 theory of labeling from a mathematical perspective was presented, specifically the idea of a classifier was presented. In this section, a state-of-the-art classifier called the Neural Network is presented along with its theoretical pinnings. The success seen in object detection tasks in camera and lidar within the past decade are attributed to a variant of this classifier called the Convolutional Neural Network [15]. From this one can introduce the theory behind object detectors, which are labeling algorithms in SAL. This knowledge is fundamental in order to understand the theoretical pinning's of the F-SAL system we present in section 3.

2.3.1 Neural Networks

Neural networks are powerful because of the universal approximation property [25], which states that any classification boundary irrespective of complexity can be approximated to any desirable precision with a finite neural network. Likewise, it is mathematically proven that the set of discriminant functions that are learned by minimizing 2.3.1, approach the posterior probability for that class as $n \to \infty$ [23] - meaning that the error is the guaranteed minimum the larger the dataset.

The fundamental building block of a neural network is a neuron - which can be thought of as a single information source, which takes a set of inputs, applies a weight to each input and then takes their summation. This result is then passed into a non-linear function called an activation function, which produces a single output that is propagated forward. Similarly, a perceptron [24] is a model for a neuron. Through connecting a set of perceptrons, a feed-forward structure called the multi-layer perceptron (MLP) can be created, this is a very famous neural network. When using the term 'multi-layer', each layer refers to a collection of neurons, which are adjacent to each other but are not connected. There are a few types of layers in a MLP, however the fundamental layers that all networks must include are the input layer, hidden layers, and an output layer. The input layer is directly connected to the feature vector of an object X (one would like to either train or inference) in a one-to-one manner. Therefore, there are n neurons in the input layer, where n is the number of features an object has. The input layer's only function is to transmit the features into the hidden layers. In the hidden layer, all neurons from the previous layer are passed as inputs to each neuron in this new layer in a one-to-all manner.

Hidden layers can be connected subsequently. Finally, the output layers correspond to the set of discriminant functions, therefore there are C neurons in the output layer, where C is the number of classes in the dataset. A visual can be seen in Fig 2.3.

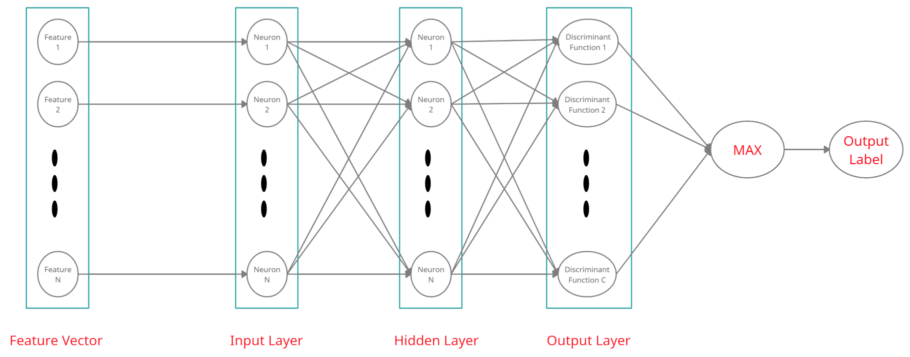


Figure 2.3: A standard Neural Network architecture

The input information for any given neuron in any layer l is denoted:

$$U_l = \{u_l,^1 u_l^2, \ldots, u_l^N\}$$

The associated weights with each information source is denoted as:

$$W_l = \{w_l,^1 w_l^2, \ldots, w_l^N\}$$

The output of a single neuron (input into next layer) then becomes:

$$U_{l+1} = G(\sum_{i=1}^{N} w_l^i u_l^i)$$

Here the symbol G represents a non-linear activation function.

Assume a dataset of N objects denoted Z exists, and each object has n features. This dataset contains objects belonging to the set of classes Ω. Like any other classifier, a neural network is trained using the dataset with its N objects and their features, and learns a set of C discriminant functions - each corresponding to a class from the set Ω. The classifier training is done through minimizing square error over the training set Z, using a cost function such as:

$$Error\ Cost = \sum_{j}^{N}\sum_{i}^{C}(g_i(z_j) - Sim(w_i, y_j))^2 \tag{2.3.1}$$

Where the $Sim(w_i, y_j)$ is a binary function, which equals 1 when the predicted label from the network $w_i = y_j$, and is 0 when the prediction is incorrect. The network is trained through error backpropagation.

Through training a neural network on a dataset, a set of weights are obtained, which are associated with each neuron, and these can be used for inference. The number of hidden layers, the number of neurons and the choice of activation function are hyper parameters that are hand-picked before training. If there are many hidden layers, a neural network is referred to as a deep neural network (DNN) [14]. In DNNs, each subsequent hidden layer has the ability to learn higher level feature representations than previous layers. With DNNs, we are not required to explicitly tell the algorithm what features to learn, there are enough parameter weights that simply based upon the training examples, the network is able to learn the features required to classify objects on its own. All state-of-the-art object detectors are based upon deep neural network designs.

2.3.2 Convolutional Neural Networks

However, traditional neural networks cannot be used when working with very high-resolution feature data often found in camera and lidar – as the number of parameters required for learning is far too great. Fortunately, there exists a very special neural network that is designed for classification in such high correlation scenarios. It is called a Convolutional Neural Network (CNN) and its neuron model is a filter. This neural network is the backbone of all object detectors and is the state-of-the-art for image and point cloud classification, detection and segmentation.

Traditionally in image classification, if one wanted to extract objects from an image, they would have to hand-pick filters to extract feature information. A filter in this context is simply a matrix which one convolves an input image with to extract information. For example, one could have filters to extract vertical and horizontal lines, and even specific edges.

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

If one convolves an image with the filter on the left they would extract all the vertial lines in that image. Likewise, with the filter on the right they would extract all the horizontal lines. This always begged the question, is it possible that an entire class of objects (car, dog, human, cancer cell, etc..) can be decomposed into a set of filters which may be used for inference. In 2013 CNNs proved they were able to achieve this task [9]. Through training on a dataset using back propagation, a CNN was able to learn decomposition patterns to inference on objects in the ImageNet dataset, achieving human level performance.

Unlike classical NNs, in CNNs the neurons are in the form of these filters, and instead of choosing handpicked filters, each filter has a n x n x d (n is the size of the filter and d is the dimension) set of weights, which are learned through training.

$$\begin{pmatrix} w1 & w2 & w3 \\ w4 & w5 & w6 \\ w7 & w8 & w9 \end{pmatrix}$$

The number of filters one may select and their size are chosen hyperparameters when designing the network. For simple tasks smaller and fewer filters are required, however for complex objects many more are needed. This idea of representing filters as learnable weights is the most influential idea in all of computer vision. Through simply training on a dataset, the classifier figures out on its own what filters are needed to learn the representation of an object.

As mentioned previously, in deep neural networks as we go deeper, the network is able to learn more complex features. In the earlier layers of a CNN, the network learns small colour, texture and orientation patterns, in further layers this information is fused to create more complex features such as contours, finally in the deepest layers the network is able to obtain detailed feature maps which uniquely identify objects belonging to classes. This simplification is only possible because image and point clouds have a lot of features which are highly correlated. Allowing CNNs to take advantage of two ideas: Parameter Sharing and Sparsity of Connections. In parameter sharing, each Neuron (filter) is special because it has the exact same weight. For example, if a vertical edge is learned by a CNN network, it can be propagated to different layers in the network and is just computed once. By sparsity of connections, we mean that in each layer, each output value depends only on a small number of inputs around it. Intuitively in images when searching for an object, the pixels that border an object are dependent on the neighbouring pixels and differ from pixels outside the border. Ultimately, these two ideas allow a CNN to achieve with fewer than 100 filters, what a traditional neural network may require several million parameters to achieve. A CNN has several new layers it adds to the traditional NN introduced previously, the most important are the convolutional and pooling layers. The **convolutional layer** is where the filters are, and the output of these layers are often called **feature maps**. **Pooling layers** are used to reduce computational costs and make features more robust. When using CNNs, a few standard NN layers are needed at the end, and these are referred to as **fully connected layers**. The actual classification task is done by these layers while the convolutional layers are for feature extraction.

CNNs are the backbones of all object detectors and over the years there have been several famous CNN architectures that have performed very well and most CNN designs are based off of these. Some of these popular architectures are VGG-16 [33], ResNETS [6], and Inception [36].

2.3.3 Object Detectors

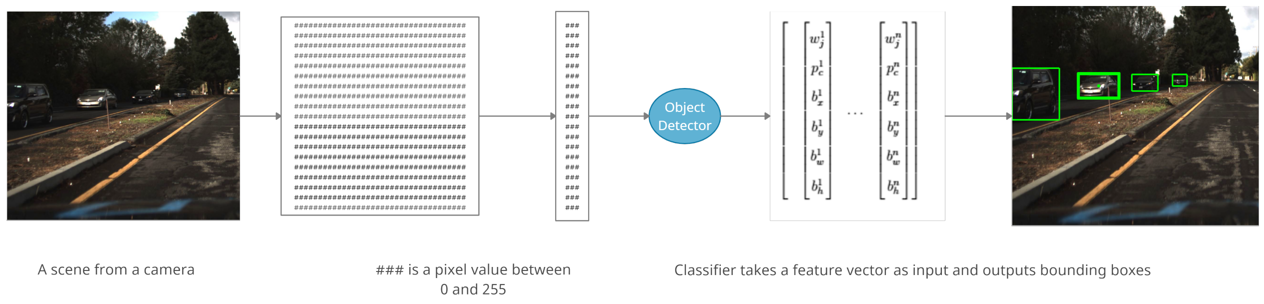

Object Detector
A scene from a camera
### is a pixel value between 0 and 255
Classifier takes a feature vector as input and outputs bounding boxes

Figure 2.4: An image based object detector.

To detect objects in a scene such as a lidar scan or an image from a camera, a class of algorithms called object detectors are used. When using the term labeling algorithms within the context of this thesis, this is a direct reference to object detectors. Classically, object detection was done using a sliding window and template matching approach, where one would sequentially run a template match over all areas in an image and then try to classify regions that have a high overlap. Overtime, the literature moved away from template matching and instead tried to use feature extraction with classification. Two famous detectors which were popular for a while were the Viola-Jones Detector, and the famous Histogram of Oriented Gradients (HOG) with a Support Vector Classifier. However, all modern object detectors are now based off of the CNN architectures mentioned in last section. When CNNs were introduced in the last section, they were described from the perspective of passing in a single image and obtaining a classification output. However, CNNs can be transformed to also handle localization, which is not a classification but a regression problem, through introducing another loss function. This is needed for object detection. Recall that in object detection, multiple objects may exist in single frame, so both localization and classification problems need to be solved (see figure 2.4). To solve this problem, modern object detectors apply a CNN classifier to different regions in an image to attempt to localize and classify multiple objects. These detectors are known as fully convolutional. A fully convolutional object detector is an object detector that has been designed to integrate the entire detection pipeline into a CNN. There are two philosophies for how to approach this problem and they are split into one-stage and two-stage detectors (see Figure 2.5). In one stage detectors, something similar to a sliding window approach is used, however the implementation is completely different and convolutional operators within the CNN are used. The advantage with this approach is that instead of having to re-run the CNN sequentially on some n x n region in a frame (traditional sliding window), it can be done in one-go through a single network pass - this allows for an incredible speed up in computational time. In two-stage detectors, the CNN has been designed to detect region proposals and then classify them.

2.3.4 CAMERA based detectors

One-stage detectors are incredibly fast and can be implemented in real-time. Some famous state-of-the-art one-stage detectors are YOLO [21], SSD [17], RetinaNET[16]. One problem with one-stage detectors is that a lot of regions in which a CNN may be executed are empty (see RetinaNET section below for details). Two-stage detectors in general obtain greater performance because they extract regions with a high probability of containing an object and then evaluate those regions only. Additionally, they obtain much better performance on smaller objects. The downside is that two-stage detectors are much slower and often cannot be used in real-time. The most popular state-of-the-art two-stage detector is Faster-RCNN [22]. Figure 2.5 illustrates the difference between one-stage and two-stage detectors

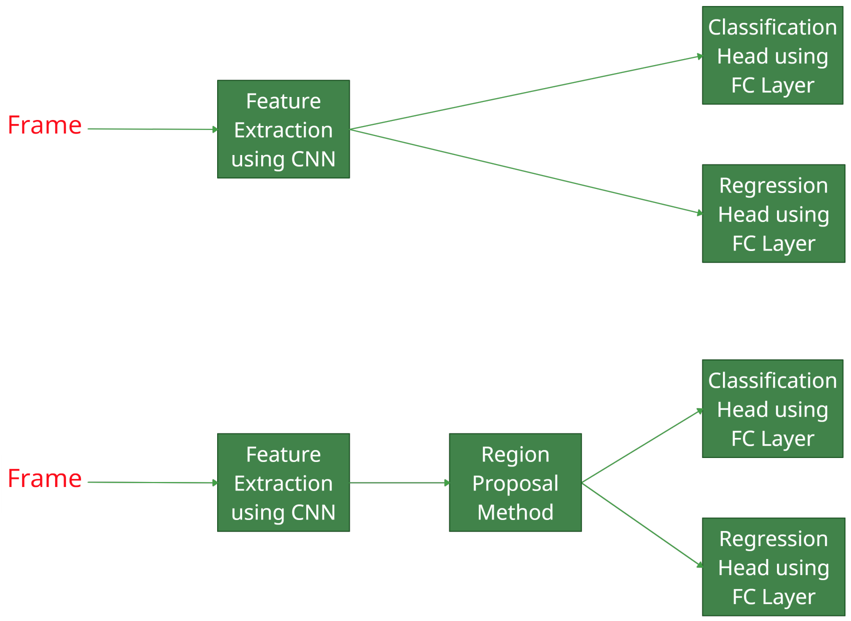

Frame
Feature Extraction using CNN
Classification Head using FC Layer
Regression Head using FC Layer
Frame
Feature Extraction using CNN
Region Proposal Method
Classification Head using FC Layer
Regression Head using FC Layer

Figure 2.5: One-stage vs Two-stage detectors. FC stands for fully-connected layer.

In the YOLO (you only look once) algorithms, we split an image into a grid of n x n cells, and we run a CNN on each of those cells (using convolutions this is done in a single step and NOT sequentially). A small n x n size is used to ensure situations where two objects are inside the same candidate cell are limited. Each cell produces a hypothesis (m hypotheses if using anchor boxes, see below) of what it believes is in that region. to reduce redundancy, which arises in such a situation, a non maximum suppression (NMS) step is applied to prune overlapping bounding boxes that belong to the same object. Another problem in object detection is that there may be overlapping bounding boxes for objects from different classes,for example, a human standing infront of their car. To handle such issues, an anchor box can be used. An anchor box is a template bounding box that is learned by the network and is class specific. It allows for the detection of class specific overlapping objects to some degree using similarity measures such as intersection over union (IOU). Therefore, for each grid cell we predict a total of m hypotheses if using anchor boxes. The fundamental weakness of this detector is that it runs into trouble when there are small objects or closely spaced objects. This is because we divide an image into a n x n grid where each cell can predict either a single object or multiple objects. If more objects make it into these regions and anchor boxes cannot detect them, then we get missed detections. A YOLO detector is fully convolutional, and includes a CNN backbone such as ResNET, and uses fully FC layers for classification (bounding box label) and regression (bounding box location).

SSDs (single shot detectors) are similar to YOLO detectors, however, instead of relying on detections from a single feature map, they perform them at various resolutions. The idea is that often in CNN architectures we start with a feature map, which slowly gets fine tuned to a smaller one as we fuse more and more information down the layers. Unfortunately, it is possible that for small objects that this resolution becomes so small that classification is no longer accurate on that feature map and may result in a missed detection. With SSDs, instead of detecting only at the final map, we also detect at other larger resolutions (layers before the final). With this approach, this detector can detect smaller objects in the earlier layers, and much larger objects in the deeper layers. Ultimately, this allows for much more robust detections. Any SSD is comprised of three components, like YOLO it has a standard CNN backbone as well as the fully connected regression and classification heads. However, in addition to these, it has a feature pyramid network which is directly connected to the CNN backbone and this is what allows for detections at multiple resolutions.

One major drawback of one-stage detectors is that they suffer from severe foreground-background class imbalances [16]. In object detection, there is a natural imbalance between foreground objects and background objects in a scene. Often we have a lot of background noise and fewer foreground objects. In two-stage detectors such as Faster-RCNN, a region proposal network (RPN) selects the top 2k candidate regions to evaluate. By only selecting the top 2k regions, we filter out majority of these background objects. Therefore, the Faster-RCNN algorithm can obtain more robust results and true detections because it is only focusing on important foreground objects during training. In the case of YOLO and SSD they focus on background regions equally as much as foreground regions, and these background regions either generate false alarms or are uninteresting and do not benefit training. The idea here is that these regions require focus from the network but have a negative contribution, this leads to object detectors not being trained properly because they should be instead focusing on feature rich foreground regions. A special SSD called RetinaNET addresses this problem through changing the loss function for standard SSDs using a novel focal loss. This loss function was able to resolve the class imbalance problem and led to RetinaNET surprisingly outperforming the two-stage Faster-RCNN.

The original RCNN (regions with CNN) object detector was based upon the two-stage detector paradigm. It used a region proposal method called selective search to segment and crop 2000 candidate regions which may contain objects. Then it applied a CNN on these proposals to extract features, finally the extracted features were classified using an SVM. Although this method worked well it was extremely slow (inference time of 49s per frame), and this algorithm was not fully convolutional. The slowdown in speed was due to the fact that each of the 2000 candidate regions had to be cropped and passed to the classifier sequentially. In 2015, Fast-RCNN was proposed. The detector still relied upon a selective search style region proposal strategy, however it added a new module, which allowed for the network to be fully convolutional. The selective search unlike before was now applied on convolutional feature maps and not a raw image. Additionally, like the single-stage detectors we mentioned earlier, fully connected layers with regression and classification heads were incorporated and convolutional operators could be used to implement a sliding window approach. This allowed for an enormous improvement in speed since we were no longer sequentially evaluating CNNs on 2000 region proposals. However, this approach was still not adequate because the selective search step still took quite some time (inference time of 2.3s per frame). Luckily in the same year, Faster-RCNN was proposed. This was also a fully convolutional network; however, the novelty was that the region proposal strategy, which was previously a bottleneck, was now incorporated through a region proposal network (RPN)and in turn allowed for an incredible speed up in performance (inference time of 0.2s per frame). Although this detector cannot be used for real-time inference, it is still an extremely good detector which is particularly useful for F-SAL systems. While its performance is sometimes worse than RetinaNET, it allows for very diverse results when conducting object detector fusion, as will be seen in the results section.

2.3.5 LiDAR based detectors

Object Detectors in lidar are similar to those in camera, as both one-stage and two-stage detectors can be used, however the outputs are usually 3D bounding boxes. Additionally, lidar outputs are point clouds which are sparse and unstructured, and fundamentally different than image pixels.

In a standard camera sensor, the coordinates for an object are with respect to the top left corner of an image. The camera physically represents an image as pixels, and pixels have an ordered structure (x,y) over colour channels. This implies that if one were to change the location of any set of pixels in an image, one would obtain a completely different image and objects in that image would be altered, therefore it is important to maintain this order. In lidar, instead of pixels, a point cloud representation is used. Points clouds are an unordered set of points that are obtained when a rotating laser scans the environment. Each such point is a tuple which contains 4 values: (x, y, z, intensity). The lidar coordinate frame is also different and has the center as the origin unlike a camera. More importantly, changing the order of these points has no effect on the objects these points represent.

For object detection in images, one could simply provide raw images to a detector and it could detect objects in that frame. However, the CNN backbones presented in the previous section, all made an implicit assumption that the raw pixel data provided was always ordered. In the literature, this is called the permutation invariance property. It is known that CNNs perform well due to sparsity of connections and parameter sharing, both of which rely on strong local structures. Since point clouds are unordered, CNN based object detectors cannot be used directly on them. Luckily, there are a few alternatives which allow for the use end-to-end NNs on point clouds.

One strategy is to design new NN Architectures, which can directly process raw point cloud data while still satisfying the permutation invariance property. These are called point-based methods. Examples of such an architecture is PointNet [19]. This is primarily based upon the theorem that a function of a set of variables called $f(x)$ is symmetric, if in its product of a function γ and composite function $g(h(x))$; γ is symmetric.

$$f(x_1, x_2, ..., x_n) = \gamma * g(h(x_1), h(x_2), ...h(x_n))$$

With this in mind, an end-to-end architecture that takes raw point clouds and outputs detections can be designed. Specifically, PointNet uses max-pooling as its symmetric function to combat permutation invariance. Additionally, to handle geometric transformation invariance it applies a separate transformation using another network. Unfortunately, PointNET was not too successful because it was unable to extract local structures, however PointNET++ [20] improved on this and provided much better results.

Alternatively, there are grid-based methods. Here raw point clouds are encoded into an ordered representation, these are called pseudo-images. Through this transformation, point cloud detectors are no longer pinned by the permutation invariance property and can make use of CNNs. A popular method to achieve this is voxelization. A voxel can considered as a pixel generalized to a volume. In voxelization, a 3D region is essentially split into subspaces or sub-cuboids of equal size, which are called voxel cells. However, due to computational reasons, we create cells only along the x-y plane and not the z dimension. Although we do not select a z value, the voxel cells still span this region with a fixed height - in the literature, 2D voxel cells are referred to as pillars. Each pillar is a vector which stores different points. Pillars allow for the representation of point clouds as ordered 3D structures so CNNs may be scaled to point cloud data. These pillar encoders can be learned using an end-to-end architecture during detector training on labeled data. A well-known end-to-end network that incorporates this idea is PointPillars [13] which is based upon a one-stage detector.

Presently, some state-of-the-art lidar detectors are Point-RCNN [31], Part-A2Net [32] and PV-RCNN [30]. These are all built upon the idea of two-stage RCNN detectors. However, there are other architectures as well such as SECOND [41], and even detectors which fuse image and lidar data together [2]. However, in the case of Fusion, the aforementioned detectors actually surpass performance from even camera-lidar fusion for 3D object detection tasks.

Chapter 3

F-SAL Taxonomy and Design

3.1 Proposed Method

The general taxonomy for F-SAL (sensor invariant) can be seen in Figure 3.1. This is a framework where both a human and multiple object detectors may label together frame-by-frame in an iterative manner. The taxonomy of the proposed SAL system is as follows: First, a localization proposal strategy that is based upon data association measures is introduced. this is to evaluate what bounding boxes from each algorithm are likely to be true objects. Secondly, a label fusion strategy that is based upon feedback is included, and this determines what each localized candidate's label will be. Third, an uncertainty evaluation (similar idea to uncertainty sampling used in Active Learning) step is applied on the label distribution of each object after label fusion - this is to tag high uncertainty labels. After these steps have concluded, a human validates the results and makes any needed corrections. This stage contains a learning step where after each frame is labeled, the ground truth and the algorithms performance are compared and weights are assigned to algorithms that perform better on respective classes. Finally, a diversity evaluation node logs the performance of each algorithm against the ground truth and calculates the diversity, determining if fusion is effective, or whether algorithms are not contributing any new information and can be pruned.

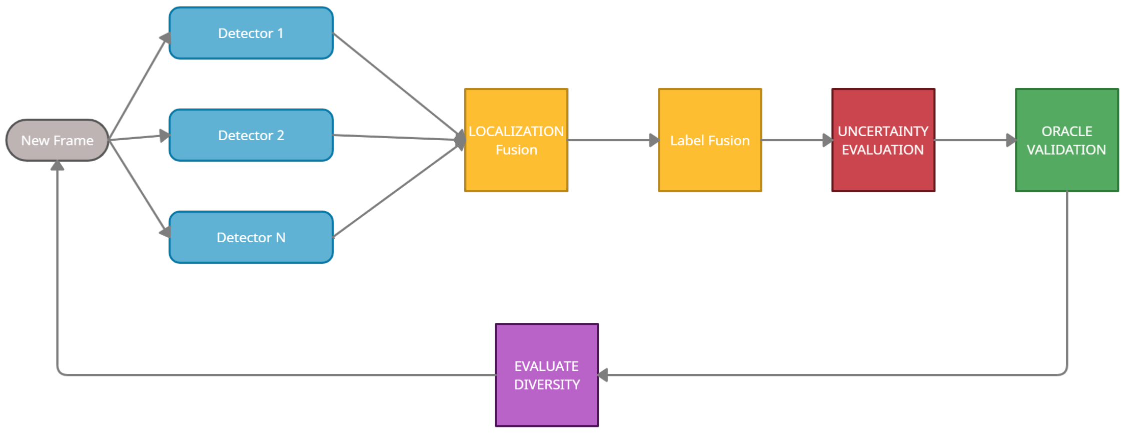


Figure 3.1: The proposed F-SAL System.

With respect to errors, the localization fusion nodes allow for a reduction in missed detections, incorrect localizations and false alarms, while the label fusion node improves incorrect classifications. Likewise, the uncertainty evaluation step further improves on incorrect classifications by tagging objects that have high uncertainty so a human may correct them more easily. Diversity evaluation does not contribute to any error reduction but ensures the efficient use of computational resources. These are the fundamental building blocks for F-SAL, and from these, several architectures can be derived for specific scenarios. Each block will now be discussed in detail from its theoretical pinnings before being applied to camera and lidar data.

However, before continuing, the work in this thesis mostly addresses problems in labeling with respect to autonomous driving. While the general framework of F-SAL is applicable to all problems, for certain disciplines such as bioinformatics where labeling scenarios are vastly different; information fusion approaches from [34] [38] may be more beneficial in both the localization and label fusion blocks.

3.2 Diversity and The Motivation for Fusion

One of the novelties in this thesis is to use several labeling algorithms as opposed to one, however, to understand why an approach like this may work the idea of diversity must be discussed. Diversity is a fundamental requirement when fusing several sources of information. All labeling algorithms have a probability associated with them, and as previously described, they make errors. However, if using multiple algorithms, one is interested in whether they make the same errors together. If these errors are not made in unison or are not correlated, then fusion is possible, and the fused output shows an improvement over the independent outputs. Therefore, diversity can be defined as a measure of how often the algorithms are correct amongst one another with respect to the ground truth and to the extent their mistakes are uncorrelated. More formally: if one algorithm makes a mistake, will other reliable algorithms be able to compensate for this mistake.

Figure 3.2 represents a scenario where there are three algorithms and five objects, and the assumption is that all the algorithms are equally reliable experts. Notice how in the left image algorithm 1 only detects three objects, algorithm 2 detects four objects, and algorithm 3 detects three objects. However, for every mistake made by an algorithm, 2 other algorithms exist to compensate for it. Ultimately, although none of the algorithms could generalize to all five objects, their fusion can. On the contrary, the image on the right illustrates a scenario where there is no diversity at all. All the algorithms make the same correct predictions and the same mistakes. Whether fusion is used or not, the output will be the same.

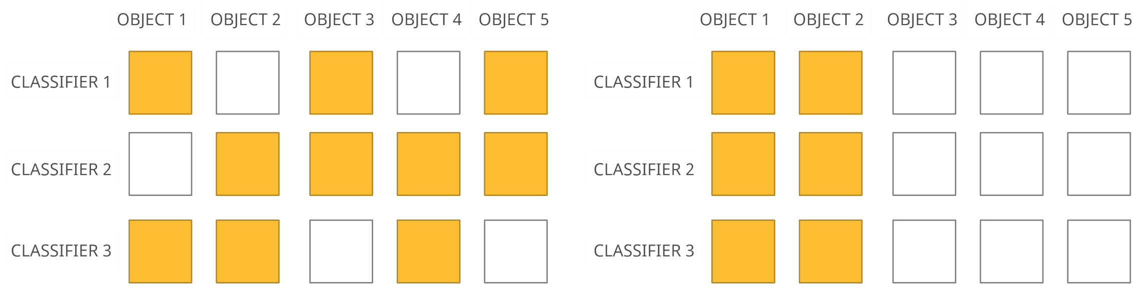


Figure 3.2: **Left:** Optimal Diversity, **Right:** Bad Diversity

Based upon the definition of diversity mentioned above, one can define a pairwise-matrix which assesses the diversity between two object detectors. As interest is in whether an algorithm correctly labels an object, therefore a diversity measure that fits in quite well in the F-SAL framework is the disagreement measure [11].

$$d_{ij} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{3.2.1}$$

This measure is introduced through defining a $2x2$ matrix whose elements are probabilities that must sum to 1, while the subscripts i, j refer to the two algorithms being compared. a is the probability that two algorithms will agree on the same correct objects, b and c are the probabilities that one algorithm was correct while the other was incorrect, and d is the probability that both were incorrect. These values are calculated whenever the detector outputs and ground truth results are compared; with respect to the F-SAL system this is right after a human makes corrections. The disagreement measure tells us the probability that two algorithms will disagree on their decision but still be correct, it is defined as the sum of both b and c.

$$d_{ij} = b + c \tag{3.2.2}$$

Therefore, it sufficient to say that if the diagonals have values greater than 0 then diversity exists to some extent, and fusion of labeling algorithm outputs may increase accuracy. Otherwise, the algorithms are getting the same lahels correct and incorrect, and fusion is of no value. The number of such pairwise-matrices needed for diversity evaluation is $\frac{L(L-1)}{2}$, where L is the number of algorithms being fused. For example, if 3 algorithms are available, diversity values must be evaluated for algorithm 1 and algorithm 2, algorithm 1 and algorithm 3, and finally algorithm 2 and algorithm 3. Likewise, the disagreement measures of each individual algorithm pair can be averaged to get a general consensus. One advantage with this measure is that it also scales to multi-modal scenarios if the sensors are synchronized.

However, to better understand the idea of diversity with multiple algorithms it is more appropriate to plot it. Figure 3.3 shows a plot where the x-axis represents the number of algorithms that correctly agreed on the same objects, while the y-axis represents the number of correct objects those L algorithms labeled. In the optimal diversity case, the number of correct algorithms and all occurrences are all over the majority vote line and a very sizeable number of objects were classified by 2 algorithms, meaning there is diversity as at least one algorithm disagreed. This illustrates that all the information obtained from independent algorithms was fully utilized, and none went to waste. While in the worst-case scenario, there is no diversity at all because either all algorithms agreed or disagreed on the same objects, fusion would not provide any information gain in this situation.

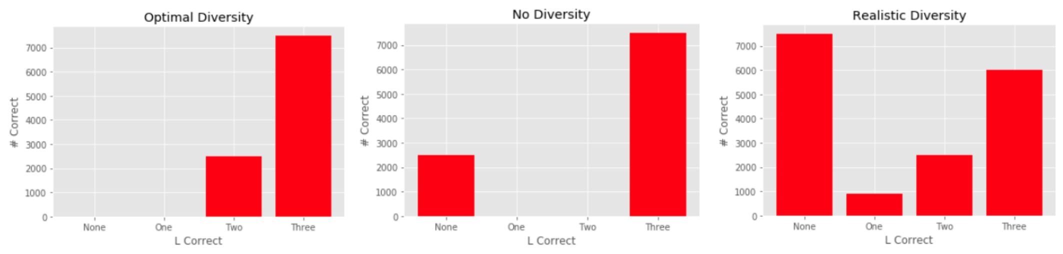


Figure 3.3: **Left:** Optimal Diversity, **Center:** No Diversity, **Right:** Realistic Diversity

Based upon empirical results shown in section 4, more realistically a diversity distribution similar to the far right graph in figure 3.3 is observed when conducting labeling. In this case, there is clearly diversity since there are many objects where only a single or two classifiers agreed on and were correct. However, this also illustrates another important idea with diversity, that, diversity may exist but be unutilizable. Since an implicit assumption was made that all algorithms are equal experts, any output from a single algorithm cannot be utilized unless a second algorithm agrees. In this case, there was a scenario where only one algorithm was correct 900 times, but since a second algorithm did not agree, those 900 objects were treated as false alarms. This also ties into the idea of how many algorithms should be selected for fusion, theoretically as $L \to \infty$ many of correct objects below the majority vote line are able to pass through and be utilized as more algorithms can support their vote.

For the F-SAL system, as fusion is needed, algorithms which are known to be both diverse and accurate are desirable. Fortunately, the backbone of all labeling algorithms in camera and lidar are NN based, and NNs are known to have sensitive classification boundaries making them optimal candidates for fusion tasks [28]. After having mentioned object detectors, their NN backbones and various other design choices in section 2.3, it should come as no surprise that due to so many possible configurations, diversity between object detectors is possible, even if not intentional.

Firstly, there are object detectors that have different region proposal methods. In the YOLO algorithm, an image is split into a n x n grid, in the Faster-RCNN algorithm a RPN detects region candidates, and for RetinaNET a feature pyramid network that allows for merging several feature maps is used. Secondly, all object detectors have CNN backbones, the choice of the CNN backbone may also vary between object detectors. However, another source of diversity is the dataset. Through training on different datasets, object detectors learn different generalization; although this requirement is often hard to meet because in general the best dataset is desirable, and all models are usually trained on it. Additionally, for a sensor like lidar, different pseudo-image conversion methods such as point-based, and grid-based methods exist, which may introduce diversity to an extent. More generally, in the case of lidar detectors there seems to be an enormous difference in the architecture design choices between state-of-the-art detectors. A more general strategy to induce diversity is to vary thresholds of an object detector which control the extent to which low confidence bounding boxes make it through as valid detections. Lowering this threshold for certain scenarios such as small objects allows for true detections to make it through at the cost of also allowing some false alarms through. Luckily, in the localization fusion stage, a vote from multiple experts is required, therefore even through lowering this threshold, false alarms can still be suppressed significantly. Proof of diversity between object detectors under some of these scenarios can be seen in the results section. (Section 4).

3.3 Localization Fusion

While object detectors output both location and classification info together, in the F-SAL system, localization fusion occurs first, and then label fusion is applied on a candidate set of high probability localizations after. This choice is tied to the idea of diversity introduced in the previous section, as the localization stage is actually where diversity can be exploited between object detectors. One method to induce diversity is to select CNNs that have different region proposal strategies, from the theory of CNN based object detection, classification is not difficult for CNNs, localization seems to be the actual problem. Therefore, labeling algorithms that have different region proposal methods are desirable. This claim is proven in section 4.

Consider a frame is passed to 3 object detectors like in figure 3.4. Each algorithm outputs a set of bounding boxes – each corresponding to a candidate object. Now the problem is that each candidate bounding box (from each algorithm) needs to be associated to an actual object, and double counting must not occur. There are 3 objects in this scene but 8 bounding boxes. Firstly, a data association measure along with a decision strategy to determine the criteria for a valid object is needed. State-of-the-art object detectors roughly have the same accuracy, and results on several datasets in section 4 validate this.

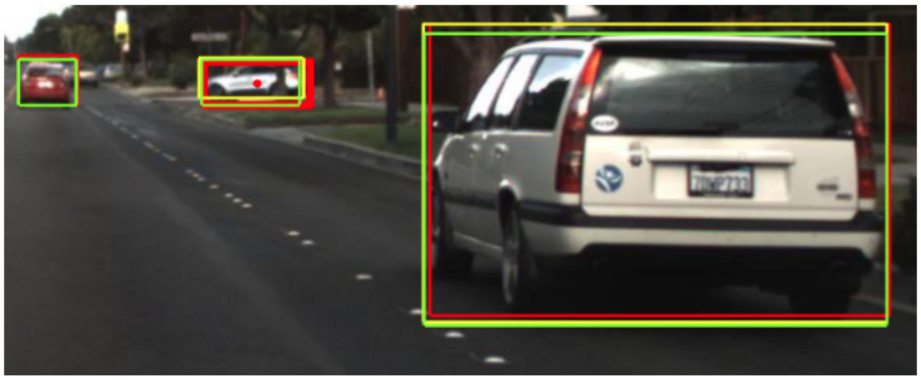


Figure 3.4: There are 3 objects but 8 bounding boxes

Since there is not a large variance, one can assume that all the object detectors being used are roughly experts to the same degree. Based upon this, a majority vote fusion rule can be selected as the decision strategy. From the theoretical perspective, the majority vote fusion rule is the optimal fusion rule when fusing information sources with equal reliability. The criteria then becomes: if $\frac{L(L-1)}{2}$ of the detectors agree on a localization, then it is accepted. As for the data association measure, the best measure for bounding boxes is the IOU measure, which can be generalized to both 2D and 3D bounding boxes [44]. While a weighted vote method is not used, it is possible that if there is a large variance in the object detectors one has available, then weighted methods would be beneficial.

3.3.1 Algorithm for Localization Fusion

A robust algorithm for localization can be formulated as follows:

1. Given one has L object detectors and each outputs a set of candidates bounding

boxes:

$$\begin{aligned}
D_1 &\rightarrow \{bb_1, bb_2, \ldots, bb_n\} \\
D_2 &\rightarrow \{bb_1, bb_2, \ldots, bb_n\} \\
&\vdots \\
D_L &\rightarrow \{bb_1, bb_2, \ldots, bb_n\}
\end{aligned}$$

2. - If labeling 2D objects, compare every pair of algorithms output against one another using 2D IOU, if the two bounding boxes between a pair have a score greater than 0.5 (0.25 if pixel area is less than 2500) then they are valid candidates that may represent the same object and are inserted into a cost matrix with their 2D IOU score. If using L object detectors, then there will be $\frac{L(L-1)}{2}$ cost matrices, one for each algorithm pair.
   - If labeling 3D objects, the steps are identical but use 3D IOU and keep the threshold at 0.5.

3. After exhaustively computing IOU over all pairs, each cost matrix is in the form of standard assignment problem which can be solved using combinatorial optimization algorithms such as Hungarian [10] with a guarantee of $O(n^3)$. The assumption here is that only one-to-one assignments are permitted. The output from this operation would return a set with the optimal assignments, one for each of the $\frac{L(L-1)}{2}$ cost matrices, a sample output from the optimization for an L = 3 example would look like:

$$D1 \rightarrow D2 = \{0:1, 1:0, 2:9, 3:7, 4:4, 6:3, 9:6, 10:5, 11:2, 12:8\}$$

$$D1 \rightarrow D3 = \{0:0, 1:4, 2:8, 3:9, 4:7, 5:5, 6:2, 9:3, 10:6, 11:1, 12:11\}$$

$$D2 \rightarrow D3 = \{0:4, 1:0, 2:1, 3:2, 4:7, 5:6, 6:12, 7:9, 8:11, 9:10\}$$

Here each element in the set $D_1 \rightarrow D_2$ represents the optimal assignment between detectors 1 and 2 for a set of bounding boxes which satisfied both the IOU and majority vote requirement. For example in $D1 \rightarrow D2$, $5 : 11$ implies that bounding box 5 from detector D_1 is associated to bounding box 11 in detector D_2.

4. However, since there are $\frac{L(L-1)}{2}$ hypotheses amongst detectors, all of them need to be merged and duplicates need to be removed. They are then added to an L-tuple. Each element in the L-tuple is the contribution from an object detector. If any pairs of algorithms did not detect an object while the others did, a **-1** is inserted to indicate a missed localization from that detector. From this we obtain the final set of valid candidates. Again, using the L = 3 example from above, the final list of hypotheses looks like:

$$\{(0, 1, 0), (1, 0, 4), (2, 9, -1), ...\}$$

Each L-tuple can be interpreted as "What D1,D2 and D3 call 0,1,0 are in fact the exact same target", In the case of 2,9 -1 this is interpreted as "what D1 and D2 call 2 and 9 are in fact the same target".

3.3.2 IOU Remarks

The IOU threshold for 2D bounding boxes can be adjusted depending on the scenario, however, very good results were obtained by using an IOU measure of 0.5 for all objects that were larger than 2500 pixels, while a score of 0.25 was used for smaller objects as they are very noisy, and it is easy to unintentionally suppress them. However, if one is not interested in labeling objects below 50 x 50 pixels, there is no need to apply this. In regard to 3D IOU, this measure is slightly more robust, however, a slight improvement might be obtainable by lowering the threshold to 0.25 for small objects. For example, on the KITTI dataset, a threshold of 0.25 was applied when the volume of a box was less than 4m, this allowed for 400 more true detections with a negligible false alarm increase.

3.3.3 Computational Cost Remarks

For 2D object detection, calculating IOU is a microsecond operation and is extremely quick, therefore the run-time cost is negligible compared to the cost of detector inference. A 2D assignment problem can be solved in $O(n^3)$, however, due to fusion, this must be solved $\frac{L(L-1)}{2}$ times. Furthermore, all the independent hypotheses which were produced must be merged and any duplicates must be removed, this step requires additional $O(n)$ time but is negligible in comparison to solving the above assignment problems. Altogether, per frame, the end-to-end algorithm is a millisecond operation in the 2D case.

Unfortunately, this is not the case in 3D object detection. The algorithm used to approximate 3D IOU relies on several other algorithms for approximations and is significantly slower than 2D IOU. Having to apply this operation $\frac{L(L-1)}{2}$ times quickly adds up. Yet, practically speaking, it is difficult to obtain more than 3 detectors, and often 5 is the hard limit for any scenario. Similarly, it is rare for the number of detector outputs to be more than 20 for a given scene. In worst case scenarios, the run-time for the above localization algorithm in lidar was up to 3-5 seconds per frame. However, this result can be significantly improved by computing this operation on a GPU if one is available.

3.4 Label Fusion

3.4.1 Extension to multiple algorithms

Since we are now dealing with the fusion of multiple detector outputs, the Bayesian theory of labeling from the previous chapter must be extended to now handle multiple algorithm outputs. The theoretical foundations used here and in label fusion are based upon the work from the book of [11]. When initially discussing classification in section 2, interest was in finding a label through maximizing the posterior based upon some set of features. However, for label fusion, it is assumed that a set of L algorithms inference on an object and each provides a label distribution (hypothesis), Now, given a set of L label distributions (hypotheses), one must decide on how to optimally assign a label.

For a given object X, a set of classifiers D which output labels are denoted as:

$$D = \{D_1, D_2, \dots, D_L\}$$

The set that contains each classifier output is denoted as:

$$s = \{s_1, s_2, \dots, s_L\} \ \forall \in \Omega$$

Each s may either be an abstract label or a soft label (mentioned later). Interest is now in the posterior of the form:

$$P(w_k|s_1, s_2, \dots, s_L) \ \forall k \in \Omega$$

From the assumption that each classifier output is independent:

$$P(s_1, s_2, \dots, s_L|w_k) = P(s_1|w_k)P(s_2|w_k)...P(s_L|w_k)$$

Then from Bayes Theorem this can be written as:

$$P(w_k|s_1, s_2, \dots, s_L) = \frac{P(w_k)\prod_{i=1}^{L} P(s_i|w_k)}{P(s_1, s_2, \dots, s_L)}$$

As before, the goal is to apply an optimization where the posterior is maximized, since the normalization constant does not depend on the label, it can removed.

$$P(w_k|s_1, s_2, \dots, s_L) = P(w_k)\prod_{i=1}^{L} P(s_i|w_k) \tag{3.4.1}$$

3.4.2 Label Fusion Taxonomy

The taxonomy for label fusion is simple, one can have either abstract labels or soft labels, and the combiner used to fuse decisions can be trainable or non-trainable.

An abstract label disregards the confidence score associated with an object. Soft labels on the other hand keep the entire label distribution associated with an object. Non-trainable combiners are usually based upon hardcoded rules such as majority vote or averaging a distribution. While on the other hand, trainable combiners assign weights to each individual algorithm that contributes to the combiner. These weights are learned through training on a dataset. The idea with trainable combiners is that more reliable algorithms should be assigned greater weights than less reliable ones. Fortunately, in F-SAL, a very special scenario exists which allows for trainable combiners to be utilized. This is possible because a human and several algorithms label simultaneously frame by frame. Since the human provides corrections at every time step, there is always a real-time dataset of each individual algorithm's performance on each class with respect to the ground truth. From this data, one can assign weights that reflect the reliability of an individual algorithm on a class. In fact, the true reliability improves and becomes known as the number of corrected objects by a human tends to infinity.

3.4.3 Abstract Label Fusion

All modern object detectors are based upon NN architectures, and generally NNs are very overconfident in their predictions and produce skewed label distributions. Therefore, in a sense it is arguable that abstract labels may produce better results in some scenarios. Additionally, it is not always possible for a detector to output the entire label distribution of an object. If one is designing their own object detectors they are able to do this. However, with several open source implementations, authors apply a threshold and only return the top label.

Weighted Majority Vote (WMV) is a well known trainable fusion method, however, it is not well suited for label fusion in F-SAL. As there is a false assumption that all the information sources being fused have the same probability of classification on each class. Interest is instead in methods which allow for assigning weights not only to a classifier as a whole, but also to its performance on specific classes, as datasets are usually imbalanced and one can obtain a much better representation with such a distribution.

Naive Bayes Combiner

One fusion method that allows for this is the Naïve Bayes Combiner (NBC) [11]. It allows for real-time training in O(1) as only a few matrix operations are required to update a confusion matrix or index it. On the other hand, inference time is O(n), where n is the number priors and likelihoods that must be computed. Often this is small, as n is equal to the number of classes in a dataset. While NBC assumes an independence assumption, empirical evidence in literature and results on two datasets in this thesis have shown that it still performs well even if this is not met.

In this combiner, equation 3.4.1 is maximized directly using approximations for the likelihood and prior:

$$P({w_k}^*|s_1, s_2, \ldots, s_L) = \underset{k=1,\ldots,c}{\arg\max}\{P(w_k)\prod_{i=1}^{L}P(s_i|w_k)\} \tag{3.4.2}$$

Both the prior and likelihood are computed from a confusion matrix. Table 3.1

shows a sample confusion matrix. If there are L classifiers, then there will be L confusion matrices. An element in the confusion matrix can be read as: class ‘i’ (row) was the true label, while the classifier predicted label ‘j’ (column). Therefore, the diagonal elements of the confusion matrix track the occurrences of correct classifications, while all other elements represent the occurrence of misclassification’s. The numbers in a confusion matrix encode all the information needed in order to calculate the exact probability of classification of a classifier on a specific class. It’s dimensions are C+1 x C+1 where C denotes the number of classes in the dataset, and the +1 term is to account for a missed detection where there is no label.

Classifier 1	Car	Truck	Pedestrian	Missed
Car	150	10	0	5
Truck	5	40	0	1
Pedestrian	0	0	50	4
Missed	0	0	0	0

Table 3.1: Confusion Matrix for C = 3

The prior is class specific for all w_k, $k = 1, \ldots, C$ and can be estimated as the ratio between the number of objects that belong to a class k in the dataset divided by the total number of objects in that dataset.

$$P(w_k) = \frac{N_k}{N} = \frac{\sum\limits_{j} CM(k,j)}{\sum\limits_{k}\sum\limits_{j} CM(k,j)} \tag{3.4.3}$$

While the likelihood can be estimated as:

$$P(s_i|w_k) = \frac{CM^i(k, s_i) + \frac{1}{C}}{N_k + 1} \tag{3.4.4}$$

The $\frac{1}{C}$ term is an offset to ensure that a single classifier does not have veto power as this may occur for very small numbers when taking a product.

Finally, the posterior becomes:

$$P(w_k{}^*|s_1, s_2, \ldots, s_L) = \underset{k=1,\ldots,C}{\arg\max}\{\frac{N_k}{N}\prod_{i=1}^{L}\frac{CM^i(k, s_i) + \frac{1}{C}}{N_k + 1}\} \tag{3.4.5}$$

Training

1. Anytime a human validates algorithm outputs in F-SAL, update the L confusion matrices $\{CM^1, CM^2, \ldots, CM^L\}$ corresponding to each specific detector. Each confusion matrix is indexed by the predicted label and the true label (in their numerical encoded form) and the count at that specific location is incremented.

Inference

1. After receiving an object X from the localization fusion block, based upon the output $s_1, \ldots, s_L$ calculate the prior (**3.4.3**) and likelihood (**3.4.4**) for each class $k = 1, \ldots, C$ and for all $i = 1, \ldots, L$

2. Select the label which maximizes the posterior using (**3.4.5**)

Behaviour Knowledge Space Combiner

Bayesian frameworks are optimal under an independence assumption. However, if there is a scenario where there is strong dependence between the outputs, the optimal combiner is the Behaviour Knowledge Space Combiner (BKS) [7]. This is essentially a look-up-table approach, which provides an estimate of the posterior $P(w_k|s)$ for all $k = 1, \ldots, c$ and for every single combination of outputs $s \in \Omega$. Like NBC, BKS training time is O(1). What this means is that to update the dataset BKS uses, the number of operations is always constant. This can be implemented as a hashmap which has O(1) insertion (amortized). Likewise, for inference one only needs to index the hashmap, which again has O(1) (amortized) access. BKS is slightly slower than the NBC because collisions occasionally occur in hashmaps but seems to perform better on the datasets evaluated (see **table 3.2**).

Training

1. Initialize a look-up-table (LUT) of size C^L which contains every possible combination of $s_1, s_2, \ldots s_L$, which are the ordered permutations with repetition.

2. Anytime a human validates algorithm outputs in F-SAL, this LUT is updated. This is done through indexing the LUT by its classifier outputs $s_1, s_2, \ldots s_L$ (key) and inserting the ground truth value as the look-up (value).

Inference

1. After receiving an object X from the localization fusion block, from outputs $s_1, \ldots, s_L$ index the LUT and then select the label from the candidate list that had the most votes.

2. If there is a scenario where no key value has been assigned, apply majority vote on $s_1, s_2, \ldots, s_L$. However, one problem with this approach is ties, in such a scenario a label must be picked at random. However, instead of doing this, since confusion matrices are actually very useful for analytics, they are always

maintained in F-SAL. Therefore, instead of picking at random, a NBC can be used instead.

Machine Learning based Combiners

Additionally, other classifiers can be used as combiners as well, this is called stacking. In this case, the classifier outputs are combined over an intermediate feature space, and the labels are features, which are passed into another classifier for classification. SVM, LDC, and KNN were evaluated on 2 separate datasets using 3 classifiers and 5 classes (Car, Truck, Pedestrian, Traffic Light, and Motorcycle), the results can be seen in Table 3.2. Since machine learning models need initial data for accurate inference, a majority vote strategy for the first 2000 objects was used, and they were only allowed to inference after this point. All these machine learning methods performed worse than both the NBC and BKS, in performance and in training as well as inference times. Although the SVM classifier came very close, having to retrain it every frame takes longer than the other approaches, and becomes problematic for large datasets. Between BKS and NBC, it is better to go with the BKS combiner. It has produced better results on all datasets and theoretically dependency between accurate classifiers is expected. Additionally, both the BKS and NBC do not require any special initialization strategies to be incorporated into our F-SAL system, and unlike machine learning techniques, they do not require any hyperparameter tuning either.

Method	Errors on Dataset 1	Errors on Dataset 2
LDC	8637	10926
KNN	1906	2642
SVM	1600	2356
NBC	1572	2345
BKS	1497	2092

Table 3.2: Label Fusion Performance on Two Datasets

3.4.4 Soft Label Fusion

With soft labels one has access to an object's entire label distribution, in general this is very useful as more features exist to make decisions than with abstract labels. Since there are several distributions, a structure is needed to store all of them. One such structure is the decision profile[11]. This is a matrix that contains the entire ranked output for all classifiers along with their probabilities for a given object X. It is denoted as $DP(x)$. Each row corresponds to a classifiers label distribution, and each column corresponds to the support from classifiers $D_1, \ldots D_L$ for a class w_j. Each element in this matrix, denoted d_{ij}, is an estimate of the posterior $P(w_j|x)$. Since these are probabilities, each row must therefore sum to 1.

$$\text{DP(x)} = \begin{pmatrix} d_{11} & d_{1j} & \ldots & d_{1C} \\ \vdots & \vdots & \vdots & \vdots \\ d_{i1} & d_{ij} & \ldots & d_{iC} \\ \vdots & \vdots & \vdots & \vdots \\ d_{L1} & d_{Lj} & \ldots & d_{LC} \end{pmatrix} \tag{3.4.6}$$

Like abstract label combiners, when training soft combiners, each classifiers set of confidence scores can be used as features to infer what the fused label is. The obvious advantage for soft combiners is that the number of features available for a single object is C X L. The theoretical pinnings for the next few methods are very strong, as fusion rules which involve averages and products on distributions can be shown to minimize metrics such as the average Kullback-Leibler divergence [18]. This is also proven from Bayesian estimation for weighted scenarios [8].

Weighted Average Combiner

As mentioned previously, with trainable combiners the goal is to find weights which reflect the reliability of each classifier on each class. One strategy is to treat this as a regression problem where the objective is to fit to the posterior probabilities. This can be implemented through the weighted average combiner [11]. Weights for a class j are denoted as w_{ikj}, and there are a total of C x C X L + 1 weights (+1 for intercept). This is the same idea as how the NBC had a confusion matrix of weights associated with each classifier.

Training

1. In the F-SAL system, any time a human validates algorithm outputs, a new entry is inserted into a real-time dataset. This dataset is represented by a feature matrix Z and ground truth vector y. The feature matrix Z is constructed by taking the decision profile of an object and flattening it into a row vector, this is done for all objects. The ground truth vector is then simply the true label, which is associated with each object.

$$Z = \begin{pmatrix} d_{11}^{1} & \dots & d_{LC}^{1} \\ d_{11}^{2} & \dots & d_{LC}^{2} \\ \vdots & \vdots & \vdots \\ d_{11}^{N} & \dots & d_{LC}^{N} \end{pmatrix} \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}.$$

2. Train a regression for each class j = 1, ... , C and return the weights from the C regressions using the following function:

$$\mu_j(x) = \sum_{i}^{L} \sum_{k}^{C} w_{ikj} d_{ik}(x)$$

Inference

1. After receiving an object X from the localization fusion block, a decision profile is constructed from the L label distributions generated by L detectors.

2. The confidence associated with each class can then be evaluated using the weights obtained during training. Select the class that maximizes the confidence scores.

$$P(w_k{}^*|DP(x)) = \underset{k=1,\ldots,C}{\arg\max}\{\sum_{i}^{L} \sum_{k}^{C} w_{ikj} d_{ik}(x)\}$$

However, one problem with this approach is that one would have to retrain a regression every frame or every few frames for an accurate representation, as the number of objects increases, the time required to train this regression also increases. Also, for a regression one must carefully select a loss function and well as a regularization term, which requires some hyperparameter tuning. Therefore, this is not a good strategy to use for soft label fusion in F-SAL. Similarly, other machine learning technique can also be applied to this problem, however, the issue is that they too are impractical due to training time requirements and hyperparameter tuning.

Decision Template Combiner

One alternative to weighted methods is the Decision Template Combiner (DTC) [12]. This approach is similar to the BKS approach with abstract labels but is generalized to handle distributions. The idea here is to keep a reference to the average decision profiles for each class w_j through a template decision profile. All the decision profiles over N objects from the dataset are merged, where a specific j was the true label, and this is done for all classes $j = 1, \ldots, C$. Then at inference time, one compares a new objects decision profile to each class template through a distance measure, and then selects the label whose profile minimizes this distance measure (or maximizes a similarity measure). Through averaging several distributions, a single accurate distribution which reflects the consensus from multiple experts is obtained. DTC is an excellent strategy for fusion and feedback in F-SAL as it allows for real-time training in O(1) as only a few matrix operations are required to update the templates.

Training

1. Initialize a set of C matrices which represent a decision template associated with a class label, each decision template DT_j for $j = 1, ..., C$ is the mean of all decision profiles for all instance of w_j observed in the run-time dataset so far.

$$DT = \{DT_1, DT_2, \ldots, DT_C\}$$

2. Anytime a human validates algorithm outputs in F-SAL, the corresponding decision templates are updated by adding the new outputs and reweighing the mean. This can be calculated through:

$$\mu_j(x) = \frac{1}{N_j} \sum DP(Z_k) \; where \; Y_k = w_j \; and \; Z_k \in Z$$

Inference

1. After receiving an object X from the localization fusion block, a decision profile is constructed from the L label distributions generated by L detectors.

2. Compare the object's decision profile to the set of templates and assess the similarity score using square euclidean distance, then select the label as the one which minimizes this distance.

$$P({w_k}^*|DP(x)) = \underset{k=1,\ldots,C}{\arg\min}\{(1 - \frac{1}{L*C}) \sum_{i}^{L} \sum_{k}^{C} (DT_j(i,k) - d_{i,k}(x))^2\}$$

3.5 Uncertainty Evaluation

This idea was inspired from querying techniques in Active Learning [27]. Sometimes classes in datasets will have very similar features (e.g., it is common for trucks, cars, and buses to me mistaken for one another) and may be misclassified. Recall how in all of the label fusion methods mentioned, a label which maximizes a posterior distribution is used. In uncertainty evaluation, if several label hypotheses in the posterior are very close, instead of pruning them, which may induce errors, this object can be flagged for human validation. The idea here is through flagging these uncertain labels, the time spent making corrections is reduced as the human does not need to search for errors. To illustrate, in Figure 3.5, from observing the label distribution, it is clear with a high degree of confidence that the label on the left is a car. However, with the distribution on the right there is uncertainty as it seems there is still a strong chance this may be a truck.

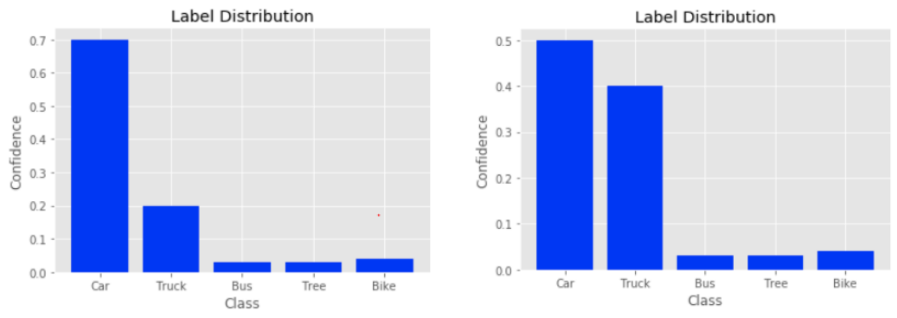


Figure 3.5: The label distribution of two objects

3.5.1 Measures for Soft and Abstract Labels

For soft labels, a decision profile which was introduced in the last section contains the distribution for all L classifiers on an object, this is enough information to assess if there is high uncertainty with a label post fusion. After obtaining the decision profile of an object, the L distributions must be averaged into a single distribution to get a consensus. From this new distribution, several information theory based or thresholding techniques can be used to determine any uncertainty.

$$P_{av}(y|x) = \frac{1}{L}\sum_{i=1}^{L} P_i(y|x) \tag{3.5.1}$$

Similarly, uncertainty evaluation can also be applied to abstract labels if using the NBC or BKS combiners. In the case of the NBC, the posterior from which argmax is taken, is a distribution. Likewise, the same can be for BKS as for an entry in the LUT can have multiple label candidates, and from their frequency of occurrence a distribution can be created. Although both methods require an additional step to apply a normalization to scale the values accordingly.

The most trivial measure that can be applied is a simple threshold condition. For example, if the highest ranked label has a probability lower than say 0.5, then it can be flagged as uncertain. This technique however only relies on the top value from the distribution and ignores everything else.

$$\arg\max_{x}\{P_{av}(y|x)\} > T \tag{3.5.2}$$

A better strategy is margin sampling, where the top two n candidates in the label distribution can be compared, and if their difference is within a threshold, then the label output is flagged as uncertain (e.g. in Figure 3.5 if a threshold of 0.15 was used, and the top two candidates were compared, this would be an uncertain scenario). However, this approach may not be reliable if the number of classes is very large.

$$|(P_{av}(y_1|x)) - (P_{av}(y_2|x))| > T \tag{3.5.3}$$

If the number of classes is very large, an entropy-based measure, which weights in the entire distribution, can be used. Entropy measures the amount of information needed to encode a distribution. Therefore, high entropy means there are several close candidates and therefore an uncertain scenario. However, low entropy means a single component dominates and there is certainty about the prediction.

$$\sum_{i=1}^{C} P_{av}(y_i|x) \log(P_{av}(y_i)) > T \tag{3.5.4}$$

In this section some measures to assess the uncertainty with a label were presented. As mentioned previously, neural network architectures for object detection are extremely overconfident with their label scores, therefore, the measures specified here are sufficient to assess any uncertain scenario if using object detectors with CNN backbones.

Chapter 4

Methods and Results

In the previous sections, the fundamental theory required to construct the proposed F-SAL system was presented. In this section, using the general taxonomy proposed in section 3, some methods which may be useful for labeling both image and point cloud data are presented. Furthermore, the results and their significance will also be discussed.

4.1 Datasets

The following datasets will be used for evaluation of the proposed F-SAL system.

4.1.1 Image Datasets

Firstly, results are presented on the Udacity Autonomous group dataset. This is an open-source dataset that contains 15000 labeled frames with 5 classes (car, person, traffic light, truck, motorcycle). This dataset represents driving conditions for several scenes at different times of the day, although they are bound to the same city and climate.

Secondly, results are presented on a small available subset of the BDD100 dataset [43]. This dataset contains 10000 labeled images and 147350 objects over 5 classes (car, person, traffic light, truck, motorcycle) obtained from Kaggle. This dataset is the most challenging autonomous vehicle dataset to date and has data from dozens of cities, varying climates and times of the day from all over the world.

4.1.2 LiDAR Datasets

The KITTI dataset [5] is a famous multi-modal dataset for perception algorithms, which contains data for both lidar and camera sensors. Each training example in this dataset is a scenario that represents a labeled 3D scene captured from a synchronized camera and lidar sensor. The camera sensor outputs a standard RGB image while the lidar sensor outputs a 360-degree laser scan – this can be thought of as a 360-degree image captured over a small interval of time through a rotating laser. Furthermore, by synchronized, this means that the camera captures an image exactly when the lidar scan is aligned with the center of the camera's field of view. The KITTI dataset contains a second grayscale camera, however, we never make use of this information, so it is omitted. KITTI has become stricter and no longer allows unauthorized access to its test set. Therefore, it has now become common convention to take the training set and split it into a training (3712) and validation set (3769). In the case of lidar detectors, they were all trained on this training set. While inference was done on the validation set which contains 17544 objects over 3 classes: Car, Pedestrian, Cyclist.

Additionally, we also evaluate lidar results on the waymo dataset [35]. This dataset is enormous and due to computational limitations results are evaluated on a single validation set (validation set 0). Unlike the KITTI dataset, the configuration is different and there are several cameras in different directions which are synchronized with central lidar sensors. However, the object detectors (trained on KITTI) used for evaluation are only trained to label objects in the front field of view, so we evaluate only on this range. There are 4898 training samples with 45681 objects over 3 classes: Car, Pedestrian, Cyclist.

4.2 Design Choices

4.2.1 Image Detectors

Machine Specifications for Inference

For image inference, the results were evaluated in a Windows 10 environment, on an Intel Core i7-8750H CPU @ 2.20GHz processor with 6 cores and 12 logical processors. The graphics card is a NVIDIA GeForce RTX 2070 with Max-Q Design and contains 8.0GB of dedicated GPU Memory.

Selected Detectors

For image data, Faster-RCNN, YOLOV5-M and RetinaNET object detectors were used. Each of these detectors were pre-trained on the COCO dataset and are based upon ResNET backbones. The two-stage Faster-RCNN detector is the slowest and has an average inference time of 200ms, while YOLOV5 had an inference time of 12ms, and for RetinaNET this was 70ms.

4.2.2 LiDAR Detectors

Machine Specifications for Inference

For LiDAR inference, the results were evaluated on an Ubuntu 18.04 environment, on an Intel Core i7-10750H CPU @ 2.60 GHz processor with 6-cores and 12 logical processors. The graphics card is a NVIDIA GeForce GTX 1650 Ti Mobile with 4.0GB of dedicated GPU Memory.

Selected Detectors

For LiDAR data, PV-RCNN, Part-A2 Free, PartA2-Anchor, Point-RCNN and SECOND were used. Each of these detectors were trained on the KITTI dataset (subset of it, see dataset section for split). PV-RCNN has an average inference time of 330ms, PartA2-Anchor has 260ms, and SECOND is the quickest at 130ms.

4.2.3 Localization Fusion

For image data, 2D IOU with Majority Vote (see section 3.2 for formulation) was selected as the localization fusion strategy. As the accuracy of the algorithms is roughly the same, no benefit was obtained by using weighted methods in any of the datasets tested. Regarding the IOU value, the best balance between detections and false alarms were obtained when an IOU threshold of 0.5 was set for any object larger than 2500 pixels, and 0.25 for any object less than this. This adaptive value is required because IOU penalizes mismatches very heavily, and small bounding boxes tend to have more variance amongst detectors. Therefore, majority of small targets are filtered out if a smaller threshold is not used. In the case of LiDAR, 3D IOU with majority vote was used. The 3D IOU is to 0.5 for objects that had a volume larger than 4, and 0.25 for objects which had a volume lower than 4. However, these values only apply to the above mentioned object detectors. It is quite likely that future object detectors will be much better at detecting smaller targets and these thresholds may change. In such a scenario, one can calculate the optimal value through evaluating detector performance on a dataset and recording how different IOU values effect false alarms and detections.

4.2.4 Label Fusion

BKS Combiner was chosen as it performed the best on all datasets, and this is in line with its theoretical backing (see section 3.3 for comparison of performance of several combiners on datasets), where it is derived as the optimal abstract combiner when there is heavy dependency between algorithms. Soft Label fusion approaches could not be used as some detector implementations were not able to provide the complete label-score distribution for objects during inference.

4.2.5 Uncertainty Evaluation

Margin Sampling on BKS Combiner Outputs with T = 0.15 (see section 3.4) was selected. If two labels from the distribution of fused label outputs are within a 0.15 margin, then a human is required to label this example. More advanced measures such as an entropy-based measures produced the exact same result and did not provide any improvement. This is in line with the idea that NNs are generally overconfident in their performance and their results are often skewed towards a particular class;this is true even for false alarms. However, it is likely that if we had more classes in our datasets, which had similar features, entropy measures would probably outperform

Margin Sampling.

4.2.6 Diversity Evaluation

This parameter is not needed for simulations and does not effect accuracy. However, when an oracle validates the algorithm outputs, the original outputs as well as any edits made by the oracle are all recorded for diversity evaluation. Here we are tracking the pairwise matrices associated with each algorithm combination. If any two pairs of algorithms are not seeing an increase in their diversity values (pairwise matrix diagonals) for some latency period, a user is prompted and asked if they would like to terminate any redundant algorithms.

4.3 Metrics

The objective in this thesis is to show that the fusion of several strong object detectors with feedback can reduce time spent labeling. Specifically, through minimizing the four sources of error associated with object detectors. Therefore, one only needs to compare those four sources of error with respect to the independent detectors and the F-SAL result. Additionally, the number of true detections is also included to give a clearer picture of performance – since minimizing those four errors is equivalent to maximizing the correct detections.

The metrics used in this comparison are simply the number of:

1. Correct Detections
2. Incorrect Classifications

3. False Alarms

4. Missed Detections

There are dozens of measures that can be evaluated using just these base metrics (e.g. accuracy, precision, recall, etc...), and object detectors are often evaluated using mAP (mean average precision). However, within the context of labeling, relying on these measures does not highlight the improvements associated with a F-SAL system for the task of labeling, therefore they are omitted.

4.4 Evidence of Diversity with Object Detectors

Before discussing specific methods, evidence for diversity will be presented as it is fundamental to F-SAL. The theoretical pinning's of how object detectors contain diversity was mentioned in section 3.2. However, one rule of thumb which is experimentally verified next, is to select detectors that have different region proposal methods - this applies to both camera and lidar based detectors. Additionally, in the case of lidar detectors there seems to be a significant difference between architecture design choices between state-of-the-art detectors.

Diversity in Image based Object Detectors

To prove diversity exists amongst image sensors, YOLOV5, RetinaNET and Faster-RCNN are evaluated - all of which have different region proposal methods - on two datasets. It is reasonable to claim that the results that are about to be obtained are in fact due to each detectors region proposal method, because all three detectors were trained on the same COCO dataset and used the same ResNET backbone. Therefore, the only difference is in how the algorithms detect region proposals. More generally, a good strategy may be to obtain the best one-stage and the best two stage detector and complement them with a 3rd detector similar to RetinaNET, which has a feature pyramid network (most SSDs have this).

Dataset 1

Using a graphical representation for diversity (Figure 4.1), one can see that all three detectors were simultaneously correct on 41, 115 objects, while two out of the three agreed on an additional 13, 737 objects and a further 8406 objects were situations where only one detector was correct. We can see each individual algorithms diversity using pairwise matrices. Faster-RCNN detected 2038 true detections which YOLO did not detect, and YOLO detected 3636 detections which Faster-RCNN Missed. Similar trends can be seen with the remaining two comparisons between Faster-RCNN and RetinaNET as well as YOLO and RetinaNET. The diversity here is good, the average disagreement measure over the pairwise matrices is 9.26%.

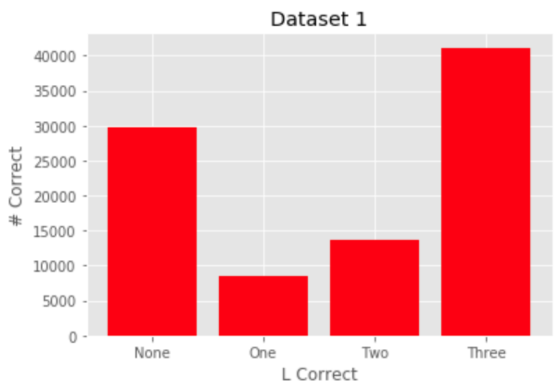


Figure 4.1: The diversity of the three detectors on Dataset 1.

	YOLOV5 Correct	YOLOV5 Wrong
Faster-RCNN Correct	50103	2038
Faster-RCNN Wrong	3636	37309

Table 4.1: Diversity between Faster-RCNN and YOLOV5

	RetinaNET Correct	RetinaNET Wrong
Faster-RCNN Correct	45106	7035
Faster-RCNN Wrong	3515	37430

Table 4.2: Diversity between Faster-RCNN and RetinaNET

	RetinaNET Correct	RetinaNET Wrong
YOLOV5 Correct	46360	7379
YOLOV5 Wrong	2261	37086

Table 4.3: Diversity between YOLOV5 and RetinaNET

Dataset 2

Using a graphical representation (Figure 4.2), one can see that all three detectors were simultaneously correct on 45, 507 objects, while two out of the three agreed on an additional 19,655 objects and a further 11,155 objects were situations where only one detector was correct. Again, from the pairwise matrices, one can view the performance of each algorithm, which shows a pairwise disagreement measure of 10.3%. This is slightly more than the previous smaller dataset. The results obtained are only for a small sample of 147350 objects, however, the famous BDD100k dataset is much larger and has 100, 000 videos, and each video is 40 seconds long at 30fps. Fusion for such a scenario would allow for a significant decrease in labeling time over independent detectors.

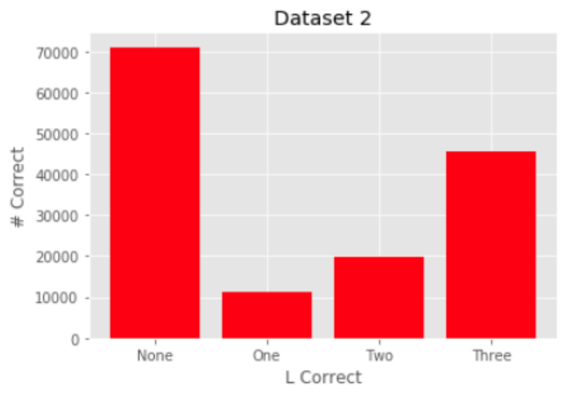


Figure 4.2: The diversity of the three detectors on Dataset 2.

	YOLOV5 Correct	YOLOV5 Wrong
Faster-RCNN Correct	49849	7303
Faster-RCNN Wrong	8939	81259

Table 4.4: Diversity between Faster-RCNN and YOLOV5

	RetinaNET Correct	RetinaNET Wrong
Faster-RCNN Correct	50982	6170
Faster-RCNN Wrong	8920	81278

Table 4.5: Diversity between Faster-RCNN and RetinaNET

	RetinaNET Correct	RetinaNET Wrong
YOLOV5 Correct	52278	6510
YOLOV5 Wrong	7624	80938

Table 4.6: Diversity between YOLOV5 and RetinaNET

Diversity in LiDAR based Object Detectors

Likewise, a similar trend is observed with lidar sensors. However, unlike image based detectors, all of the state-of-the-art lidar detectors are two-stage detectors. From the literature it seems that top state-of-the-art methods are fundamentally very different from the architecture perspective. This is again due to varying region proposal strategies, but in the case of lidar specifically, this is also true because of varying pseudo-image conversion methods. In PointRCNN, region proposals are pooled through segmenting point clouds into foreground and background points in a bottom-up manner. In the second stage, there is point cloud region pooling which allows for further refinement of proposed bounding boxes. In PV-RCNN, the authors actually combined voxel and point based methods to obtain more accurate bounding boxes. The motivation was that point based methods have flexible receptive fields while voxel based methods produce high quality 3D proposals. Similarly, if one compares the frameworks for Part-A2 and SECOND, it can be seen that they too are fundamentally very different.

Dataset 3

Using a graphical representation for configuration 1 (Figure 4.3), it can be seen that all three detectors were simultaneously correct on 10555 objects, while two out of the three agreed on an additional 3882 objects and a further 612 objects were situations where only one detector was correct. Again, from the pairwise matrices, one can see the performance of each algorithm, which shows a pairwise disagreement measure of 14.59%. Moreover, looking at configuration 2 (Figure 4.4), one can see that all three detectors were simultaneously correct on 13484 objects, while two out of the three agreed on an additional 1234 objects and a further 597 objects were situations where only one detector was correct. From the pairwise matrices an average disagreement measure of 3.6% is obtained. In configuration 1, detectors that are based off of completely different architectures that have diverse region proposal strategies are used. In configuration 2, two detectors which are both very precise but at the same time have similar region proposal strategies (Part-A2 Anchor and Part-A2 Free) are intentionally chosen. The result is that less diversity is observed, proving our claim that region proposal methods can induce diversity between detectors. While the gains from diversity do not seem to be much in this case compared to the image detector dataset. This is only because the KITTI dataset is extremely small and the accuracy of these detectors is already very high, therefore only marginal gains are possible in such a scenario.

Configuration 1: PV-RCNN, Point-RCNN and Part-A2-Anchor were used

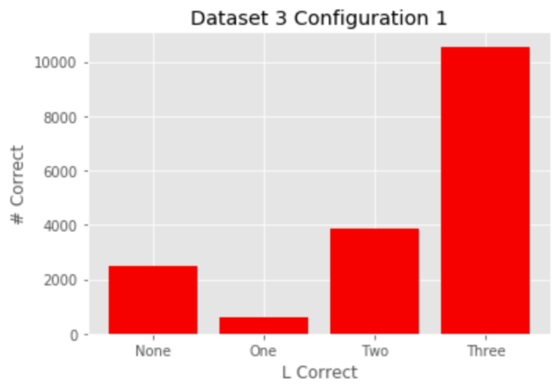


Figure 4.3: The diversity between PV-RCNN, Point-RCNN and Part-A2-Anchor

	Point-RCNN Correct	Point-RCNN Wrong
PV-RCNN Correct	10910	3514
PV-RCNN Wrong	115	3005

Table 4.7: Diversity between PV-RCNN and Point-RCNN

Configuration 2 PV-RCNN, Part-A2-Free and Part-A2-Anchor were used

	Part-A2-Anchor Correct	Part-A2-Anchor Wrong
PV-RCNN Correct	14210	214
PV-RCNN Wrong	116	3004

Table 4.8: Diversity between PV-RCNN and Part-A2-Anchor

	Part-A2-Anchor Correct	Part-A2-Anchor Wrong
Point-RCNN Correct	10814	211
Point-RCNN Wrong	3512	3007

Table 4.9: Diversity between Point-RCNN and Part-A2-Anchor

	Part-A2-Free Correct	Part-A2-Free Wrong
PV-RCNN Correct	13993	499
PV-RCNN Wrong	215	2837

Table 4.10: Diversity between PV-RCNN and Part-A2-Free

	Part-A2-Anchor Correct	Part-A2-Anchor Wrong
PV-RCNN Correct	14238	254
PV-RCNN Wrong	216	2836

Table 4.11: Diversity between PV-RCNN and Part-A2-Anchor

	Part-A2-Anchor Correct	Part-A2-Anchor Wrong
Part-A2-Free Correct	13954	254
Part-A2-Free Wrong	500	2836

Table 4.12: Diversity between Part-A2-Free and Part-A2-Anchor

Dataset 4

Since the detectors were trained on the KITTI dataset, the true cross-domain generalization capability cannot be assessed unless another dataset is evaluated. In this case the Waymo dataset is used. PV-RCNN, Part-A2-Anchor and SECOND detectors were chosen, and using a graphical representation (Figure 4.5), it can be

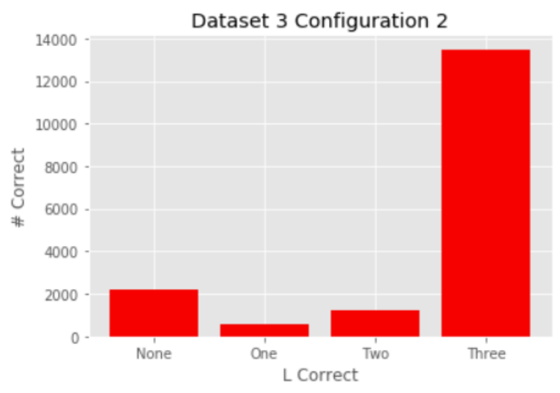


Figure 4.4: Diversity between PV-RCNN, Part-A2-Free and Part-A2-Anchor. Notice how there is much less diversity in this case when two detectors with similar region proposal methods are used.

seen that all three detectors were simultaneously correct on 1462 objects, while two out of the three agreed on an additional 1143 objects and a further 1605 objects were situations where only one detector was correct. Clearly, the accuracy of the detectors was extremely low on this dataset. The result here is unusual and it is discussed in more detail in the discussion section.

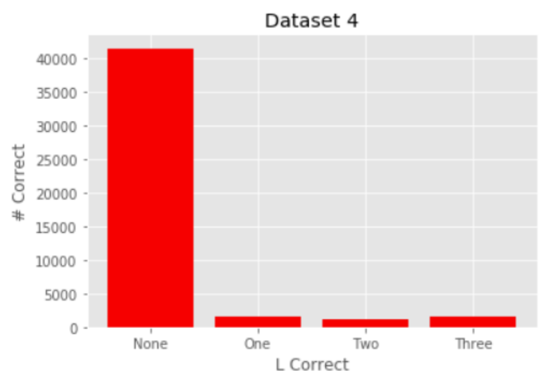


Figure 4.5: Diversity between PV-RCNN, Part-A2-Anchor and SECOND

	Part-A2-Anchor Correct	Part-A2-Anchor Wrong
PV-RCNN Correct	1870	74
PV-RCNN Wrong	803	42934

Table 4.13: Diversity between PV-RCNN and Part-A2-Anchor

	SECOND Correct	SECOND Wrong
PV-RCNN Correct	1704	240
PV-RCNN Wrong	803	42934

Table 4.14: Diversity between PV-RCNN and SECOND

	SECOND Correct	SECOND Wrong
Part-A2-Anchor Correct	2433	240
Part-A2-Anchor Wrong	74	42934

Table 4.15: Diversity between SECOND and Part-A2-Anchor

4.5 Method 1 - Standalone Object Detector Fusion

The most practical method is to simply fuse independent object detectors. The advantage of this approach is that there are many open-source implementations for state-of-the-art object detectors and whenever new research is released it is often accompanied with reproducible code. Additionally, because of transfer learning one can train custom data very easily on these stand-alone object detectors. The general F-SAL pseudocode using this method can be seen in Figure 4.7.

4.5.1 Speed Remarks

The following analysis is based upon the parameters specified in Section 4.2.

Multiple detector inference: The total average time required for inference

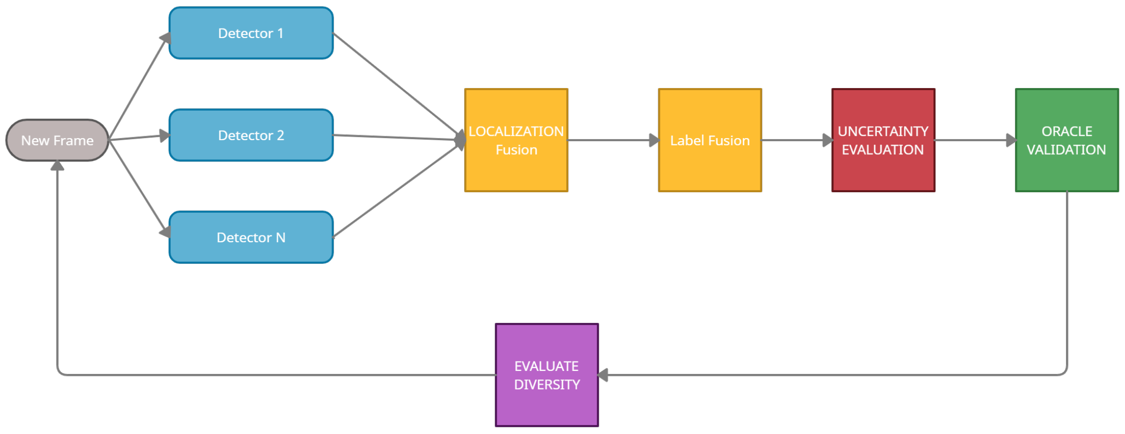
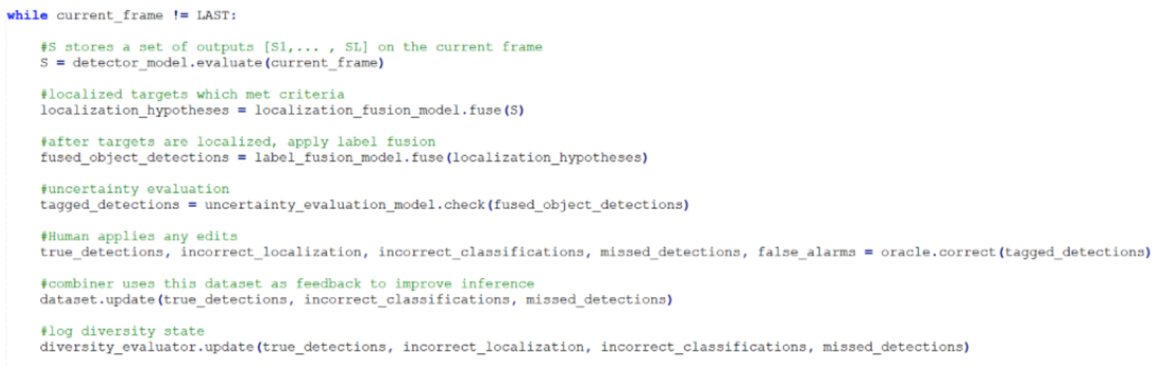


Figure 4.6: The proposed F-SAL System.



```
while current_frame != LAST:

    #S stores a set of outputs [S1,... , SL] on the current frame
    S = detector_model.evaluate(current_frame)

    #localized targets which met criteria
    localization_hypotheses = localization_fusion_model.fuse(S)

    #after targets are localized, apply label fusion
    fused_object_detections = label_fusion_model.fuse(localization_hypotheses)

    #uncertainty evaluation
    tagged_detections = uncertainty_evaluation_model.check(fused_object_detections)

    #Human applies any edits
    true_detections, incorrect_localization, incorrect_classifications, missed_detections, false_alarms = oracle.correct(tagged_detections)

    #combiner uses this dataset as feedback to improve inference
    dataset.update(true_detections, incorrect_classifications, missed_detections)

    #log diversity state
    diversity_evaluator.update(true_detections, incorrect_localization, incorrect_classifications, missed_detections)
```

Figure 4.7: Pseudocode for F-SAL Method 1

in image data is 282ms per frame. In the case of lidar data this is approximately 720ms per frame. real-time performance is not required therefore this result is excellent. However, if resources are available, this multi detector inference can be handled concurrently. In which case, the runtime is no longer $T_1 + T_2, \ldots, +T_L$ but instead $max(T_1, T_2, \ldots, T_L)$ + a small overhead for parallelization.

Localization Fusion: In the case of a highly cluttered 2D scene, this component only took 15ms to execute. Evaluating this algorithm for the worst-case scenario in a highly cluttered 3D lidar scene, the performance was 3-5s per frame (see **section**

3.3 for bottleneck). However, in the case of SAL it takes a human much longer to validate a frame. With that in mind, even the worst-case scenario with lidar is not significant in the SAL process.

Label Fusion + Uncertainty Evaluation: This step takes 30 microseconds per object, making this the fastest component in the entire system.

Human Corrections: This varies and depends on several factors such as if one is interested in simply object detection or instance segmentation, or whether they are working with image or LiDAR data, and even the scene type. For simple scenes this may take only 15 seconds, while for more complex scenes this would take longer.

Diversity Evaluation: Here data is simply being logged to pairwise matrices. This requires a few comparison operations and is a microsecond operation.

4.5.2 Results

Image Datasets

The results on Dataset 1 can be seen in Table 4.16. The number of correct labels are slightly better than the best detector (YOLOV5 in this case). However, what is incredible is the number of false alarms which are suppressed. In terms of robustness, the F-SAL system is much better than any single algorithm. Moreover, with respect to the specific stages in the pipeline, the label fusion stage reduced the number of incorrect labels from 3165 (if simple majority vote was used) to 2107 (using BKS). The uncertainty evaluation stage tagged 548 candidate labels with high uncertainty which would have been incorrect, and the exact diversity values can be seen in section 4.4.

Detector	Correct	Missed	Incorrect Classification	False Alarms
Ground Truth	93086	0	0	0
Best Scenario	63258	29828	0	0
Faster-RCNN	54912	38174	2919	57712
YOLOV5	56334	36752	3911	49283
RetinaNET	50468	42618	2789	43028
F-SAL	**57441**	**35645**	**2107**	**32683**

Table 4.16: Results on Dataset 1. Best Scenario is if full diversity was utilized

Moreover, the results on Dataset 2 (Table 4.17) were also good. F-SAL actually had more true detections than 2 out of the 3 detectors, while RetinaNET did have more true detections, the number of false alarms it allowed through were an excess of 35000 compared to F-SAL. Making the label outputs from F-SAL clearly more robust. With respect to the specific stages in the pipeline, the label fusion stage reduced the number of incorrect labels from 2560 (if simple majority vote were used) to 1833 (using BKS). The uncertainty evaluation stage tagged 1030 candidate labels with high uncertainty which would have been incorrect, and the exact diversity values can be seen in section 4.4. Dataset 2 is considered an extremely challenging dataset so the performance on this dataset highlights the improvement F-SAL brings.

Detector	Correct	Missed	Incorrect Classification	False Alarms
Ground Truth	147350	0	0	0
Best Scenario	76317	71033	0	0
Faster-RCNN	62826	84542	2954	16689
YOLOV5	62649	84701	2508	16207
RetinaNET	71500	75850	3741	46620
F-SAL	**67358**	**79992**	**1833**	**10413**

Table 4.17: Results on Dataset 2. Best Scenario is if full diversity was utilized.

Lidar Datasets

In regard to dataset 3 (Table 4.18 and Table 4.19), the results were good on both configurations. For configuration 1, although the best detector did detect an additional 215 detections, it also generated 23907 additional false alarms. For configuration 2, 168 more detections were obtained, but 3445 extra false alarms were acquired in comparison to configuration 1. Regardless, both F-SAL implementations performed better than any independent detector and are far more robust. Comparatively, almost as much detections as the best independent detector are obtained, while having 60% less false alarms.

However, in both configurations, it seems that label fusion and uncertainty evaluation were not helpful. There was only a single label improvement from majority vote using BKS, and only two labels were tagged by uncertainty evaluation. This makes sense because the KITTI dataset is a benchmark dataset which only contains three classes - Car, Pedestrian and Cyclist. Therefore, as the accuracy is already high, and coupled with the fact that discriminating between three distinct classes is very easy; only marginal returns can be obtained from stages such as label fusion and uncertainty evaluation in such a scenario.

Moreover, the results on dataset 4 (Table 4.20) were not good. What occurred was that the lidar detector trained on the KITTI dataset was unable to generalize to the much more complex Waymo dataset (see discussion section 4.7). This performance had nothing to do with the F-SAL system itself, yet interestingly enough though, even with very poor lidar detectors, the F-SAL system still produced the

Detector	Correct	Missed	Incorrect Classification	False Alarms
Ground Truth	17544	0	0	0
Best Scenario	15049	2495	0	0
PV-RCNN	14703	2841	7	34372
Point-RCNN	11004	6540	4	13127
Part-A2-Anchor	14754	2790	6	39783
F-SAL	**14539**	**3005**	**3**	**15876**

Table 4.18: Results on Dataset 3 - Configuration 1. Best Scenario is if full diversity was utilized.

Detector	Correct	Missed	Incorrect Classification	False Alarms
Ground Truth	17544	0	0	0
Best Scenario	15315	2229	0	0
PV-RCNN	14703	2841	7	34372
Part-A2-Free	14334	3210	9	31659
Part-A2-Anchor	14754	2790	6	39783
F-SAL	**14707**	**2837**	**0**	**19321**

Table 4.19: Results on Dataset 3 - Configuration 2. Best Scenario is if full diversity was utilized.

most robust results. The number of detections were lower than the best detector but the suppression of the false alarms was significant. Similarly, label fusion and uncertainty evaluation made no difference in this case, and the reasoning is the same as in dataset 3.

Detector	Correct	Missed	Incorrect Classification	False Alarms
Ground Truth	45681	0	0	0
Best Scenario	4210	41471	0	0
PV-RCNN	2027	43654	0	11369
Part-A2-Anchor	3593	42088	0	26757
SECOND	3182	42499	32	142060
F-SAL	**2747**	**42934**	**0**	**9156**

Table 4.20: Results on Dataset 4. Best Scenario is if full diversity was utilized.

4.6 Method 2 - Object Detector Fusion with L - 1 validation

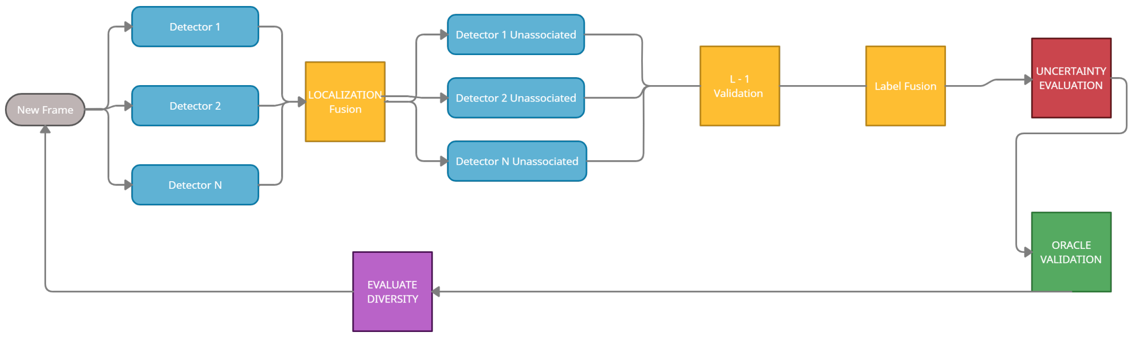


Figure 4.8: The proposed F-SAL System with a post validation stage

```
while current_frame != LAST:

    #S stores a set of outputs [S1,... , SL] on the current frame
    S = detector_model.evaluate(current_frame)

    #localized
    associated_hypotheses, unassociated_hypotheses = localization_fusion_model.fuse(S)

    #preprocessing on unassociated hypotheses
    masks = preprocessUnassociatedHypotheses(unassociated_hypotheses, CROP | CENTER, RESIZE)

    #apply inference again on masks of unassocaited hypotehses which contain modifications
    associated_hypotheses_postvalidation = postValidation(masks)

    #merge the two sets to get the set of associated localizations
    localization_hypotheses = merge(associated_hypotheses, associated_hypotheses_postvalidation)

    #after targets are localized, apply label fusion
    fused_object_detections = label_fusion_model.fuse(localization_hypotheses)

    #uncertainty evaluation
    tagged_detections = uncertainty_evaluation_model.check(fused_object_detections)

    #Human applies any edits
    true_detections, incorrect_localization, incorrect_classifications, missed_detections, false_alarms = oracle.correct(tagged_detections)

    #combiner uses this dataset as feedback to improve inference
    dataset.update(true_detections, incorrect_classifications, missed_detections)

    #log diversity state
    diversity_evaluator.update(true_detections, incorrect_localization, incorrect_classifications, missed_detections)
```

Figure 4.9: Pseudocode for F-SAL Method 2

Normally after Localization Fusion, a set of majority vote detections are allowed into the label fusion block, and a set of unassociated detections are suppressed. However, recall in the diversity plots that there are often values for L = 1, and these represent true detections which were wasted because other detectors did not agree. If one could get some of these detections over the majority vote line, one could obtain even better results. One strategy to achieve this is to further evaluate the unassociated detections to confirm whether they are in fact false alarms. Surprisingly, if one applies minor modifications to some of the true detections which were missed and rerun them through the L – 1 detectors; a portion of them are classified correctly the second time. One advantage with this method is that it does not require searching for any extra object detectors, as the same detectors are used for validation. This method applies only to image data.

From empirical evidence observed on two image datasets, it seems that object detectors sometimes have difficulty with objects that are at the edge of a sensor's FOV, and objects which are small. Specifically, Faster-RCNN is very good at finding small objects, while RetinaNET can still label some objects near the edges of a sensor's FOV. The following operations are designed around this idea and have shown good results.

4.6.1 Post Validation Pre-processing

Assume there is a scenario where there are 3 detectors and one of the detectors located an object that the other 2 did not, in this scenario one would:

1. Apply a mask to crop out this candidate object. Make sure to apply padding to ensure all edges are extracted.
2. Overlay the cropped image onto an empty background.
3. Resize the crop based upon width (maintain aspect ratio). A size of 224 or 112

is good, this value was chosen to match the resolution most neural networks down sample an image to.

4. Translate it to the center.

One thing to note is that the object crop must be overlaid onto a frame of the same size as the original image. This is because if an imaged is cropped with its original dimensions (say 100 x 110) and run directly through an object detector, its resolution will get scaled down and lead to a missed detection.

4.6.2 Post Validation Decision

After preprocessing the images, detectors 2 and 3 can be re-run, if at least one of the detectors agrees that the object has the same label as what detector 1 claimed, this object is allowed through to the label fusion stage.

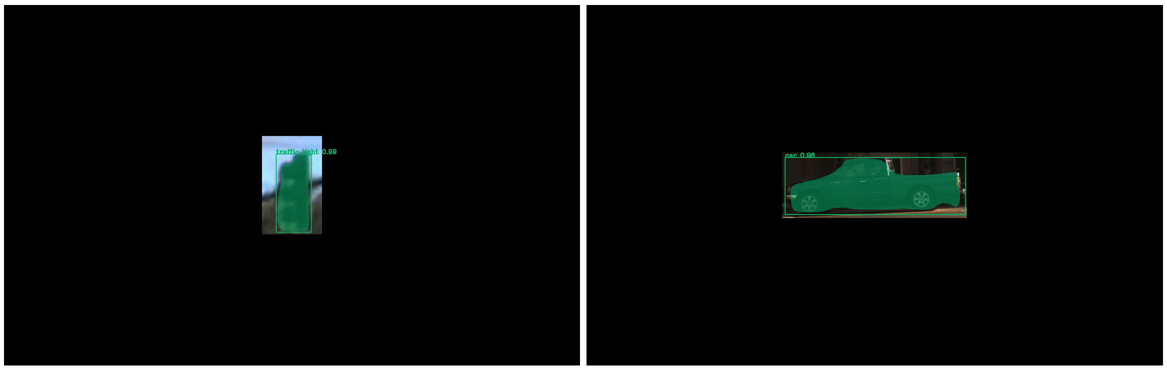


Figure 4.10: **(LEFT)** The first time a detector evaluated this traffic light it was missed, however, after applying preprocessing and running the detector again, surprisingly it is 96% confident this is a traffic light. **(RIGHT)** The first time a detector evaluated this truck it was missed, the second time a detector was 96% confident this was a Truck.

4.6.3 Detections to False Alarms Trade

This approach does allow for more detections but also leads to a significant number of false alarms making it through. It is also computationally more demanding, where most of the errors are due to the following.

Multiple Detections from Single Crop

Multiple detections sometimes arise from a single crop. One strategy to resolve this is to classify these as false alarms as the objects are so small. Additionally, one may be interested in just tagging the region as high uncertainty and notify the human for validation.

Object Splits

Small objects may end up being split between detectors (see Figure 4.11). In this case what happened was that one detector detected one half of a vehicle, and another detector detected the other half. Initially none of these were allowed through, however, through the post validation stage where other detectors revaluated these, they made it through. This problem is challenging to fix because it requires some sort of post processing where merges may be handled, which further increases the computational cost of this method.

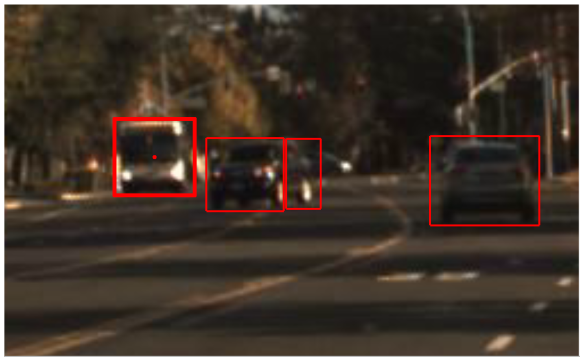


Figure 4.11: A scenario where detectors detected two separate parts of the same vehicle.

Approximations

The fundamental issue with this approach is the number of inference operations, which may be required in the validation step. Fortunately, single-stage detectors have remarkably fast inference times, and one strategy is to only allow single-stage detectors to conduct validation. Additionally, in the case where there are $L > 3$; one strategy is to only conduct validation on candidates that are 1 vote below the majority vote line. For example, in the case where L = 5, one should only apply post validation on candidates that had 2 votes. In the case of L = 7, the requirement would be at least 3 votes. Furthermore, the moment an object has surpassed the majority vote line, there is no need to evaluate the remaining detectors. However, another strategy is to look at how the algorithms are performing, and only allow the algorithms performing well on the ground truth to have their values validated.

Run-time Cost Remarks

This method is much more computationally demanding than Method 1 where we only required L inference operations. Here, at worst $N_{unassociated}(L–1)$ inference operations may be required, where $N_{unassociated}$ is the number of unassociated detections which are 1 vote below the majority vote line. Likewise, the time taken in pre-processing must also be accounted for; while this time is negligible with respect to inference time, it is still substantially more than the time required to evaluate a frame compared with Method 1.

Practical Use Case

Instance Segmentation: This method is useful when a labeler is willing to trade more false alarms for more detections. This sounds very unusual, but this can be beneficial in a scenario where a labeler must label complex objects that may be 100+ polygon vertices per object. In that case, it's easier to allow false alarms through and simply delete them, rather than labeling missed detections from scratch. **L = 2:** So far it was mentioned that L = 3 is normally the best we can do, and at best we may be able to get L = 5. However, sometimes only 2 good detectors may be available. Using this approach, a fraction of the unassociated true labels will be able to make it over the majority vote line (which is 2) through the post validation step.

4.6.4 Results

The results on Dataset 1 can be seen in Table 4.21. Using this method more detections were achieved at the cost of allowing more false alarms through. The setup was as follows: any unassociated detections from Faster-RCNN were reevaluated by YOLO and RetinaNET, and also any unassociated detections from YOLO were revaluated by Faster-RCNN and RetinaNET. The outputs from RetinaNET already contained an enormous number of false alarms so any unassociated detections from RetinaNET were not revaluated. In an actual labeling scenario, any of the methods mentioned in section 4.6.3 under approximations may be used. In F-SAL, access to algorithm performance in real-time is always available, therefore data similar to Table 4.16 is available to decide which approximation strategy may be best. Ultimately, this method allowed for an additional 2909 detections at a cost of adding 12278 false alarms. Whether this is useful or not, again depends on the labeling objective however even with the additional false alarms, the F-SAL result is still more robust than any of the other independent detectors.

Detector	Correct	Missed	Incorrect Classification	False Alarms
Ground Truth	93086	0	0	0
Best Scenario	63258	29828	0	0
Faster-RCNN	54912	38174	2919	57712
YOLOV5	56334	36752	3911	49283
RetinaNET	50468	42618	2789	47020
F-SAL	**60350**	**32736**	**2143**	**44961**

Table 4.21: Results on Dataset 1. Best Scenario is if full diversity was utilized

4.7 Discussion of Results

4.7.1 Image Data

Overall, the results were good for image data and F-SAL provided more robust labels. It was shown that state-of-the-art object detectors in image data have a degree of diversity amongst one another, and this diversity can be utilized to improve robustness. All stages of the pipeline: Localization Fusion, Label Fusion and Uncertainty Evaluation allowed for improvements in labeling. With respect to [43] which is the only semi-automated labeling experiment to this authors knowledge, they used only a single Faster-RCNN detector to conduct labeling. While compared to an independent Faster-RCNN detector, our F-SAL systems performance was better as we obtained more detections as well as fewer false alarms.

4.7.2 Lidar Data

When evaluating the KITTI and Waymo datasets, it was observed that label fusion and uncertainty evaluation did not play any role. This does not mean that label fusion or uncertainty evaluation are not useful for lidar, what occurred was that all the datasets which were available had a limited number of classes. For label fusion and uncertainty evaluation to be of value, the number of classes has to either be more or classes which have similar features are needed. This is verified by the excellent performance obtained with image datasets which met this requirement.

Apart from the aforementioned, the most important question is, why did lidar perform so poorly on the Waymo dataset? There are in fact several reasons for this, and the paper: "Train in Germany, Test in the USA: Making 3D Object Detectors Generalize" [39] illustrates that this is a general phenomenon with lidar detectors. Essentially the authors through several experiments observed that there is a poor generalization capability with lidar detectors when training on one dataset but testing them on a different one.

Moreover, some reasons for this are that the KITTI dataset is very small and only has 3712 training examples. It also only contains scenes from a single German city in perfect weather conditions and only during the daytime. In the case of the Waymo dataset, there are scenes from several cities in the U.S. and in multiple climates and at varying times of the day. Additionally, the KITTI dataset scenes were mostly on flat roads; in the case of the Waymo dataset, there were several curved and hilly roads. Secondly, in terms of configuration, the KITTI dataset had a single sensor that was mounted exactly 1.6m from the ground; while in the case of the Waymo dataset 5 separate LiDAR sensors were used to generate each point cloud. One observation to make is that unlike the large number of image based datasets; the largest lidar dataset currently is the Waymo dataset itself. Fundamentally, the poor performance observed in the Waymo dataset is attributed to very different data formats between the training and testing sets.

In the future if a large dataset equivalent to what ImageNET, PascalVOC and COCO have done for image recognition tasks is made available, F-SAL based systems should not suffer from what occurred on the Waymo dataset. While cross domain lidar lableing may be challenging presently, as the community has more need for the sensor, more generalized datasets should be released, which would allow F-SAL to produce results similar to the other excellent results observed in the other datasets.

4.7.3 Advantages of F-SAL

When labeling with a single object detector, in a scenario where the false alarm rate is too large, it is possible to lower this by varying confidence thresholds. Essentially, these parameters control the extent to which low confidence detections are suppressed. Increasing this value allows for lower false alarms but also leads to some true detections being suppressed. Another strategy is to use a two-stage classifier, in a two-stage classifier a second classifier is used to reduce false alarms. These two classifiers are end-to-end and must be trained together. However, the biggest issue with this approach is the same, several true detections have to be suppressed in order to reduce false alarms.

The biggest advantage of F-SAL over these methods is that a lower false alarm rate can be achieved without sacrificing any detections. This is possible because localization fusion requires a majority vote in order to determine a true detection. When L = 3, it is much more difficult for independent detectors to agree on false alarms, therefore true detections can be maximized and false alarms can be suppressed. Also, F-SAL is an end-to-end framework, which has label fusion with feedback as well as uncertainty evaluation, and these stages also contribute to a performance over these methods.

Chapter 5

Conclusion

So far in the literature, the only technique to conduct SAL has been to use standalone object detectors. In this thesis, a brand-new framework for semi-automated labeling was presented. It is based on the fundamental idea that diversity between object detectors seems to exist; this has been empirically verified in this thesis for object detectors that have different region proposal methods. More specifically, F-SAL is a framework that consists of localization fusion, label fusion with feedback and uncertainty evaluation. Based upon human corrections, the system is able to label objects more accurately. Through developing the theoretical pinnings behind F-SAL, it was ultimately verified that F-SAL can produce better results for labeling than individual object detectors. Therefore, if available, labeling should be done with respect to not a single but multiple object detectors.

While the focus on this thesis was on using object detectors, it is also possible to simply take the label fusion and uncertainty evaluation algorithms mentioned here and use them with independent classifiers on a labeling tool. Meaning, a labeler may have a scenario where they may select a region-of-interest in a software program, and then a bank of classifiers can be used to classify the object in that region.

However, one critical area for improvement with F-SAL is possible. Presently feedback is only in the label fusion step, an obvious improvement would be to add feedback to the localization fusion block. However, this is a challenging problem that is not as simple as label fusion. This is based upon the idea that a region proposal method should be learned on a real-time dataset as a human conducts labeling, and based upon corrections the region proposal method of the network should adjust. This would require a specific neural network architecture to be designed for the problem of labeling specifically.

Bibliography

[1] D. Acuna, H. Ling, A. Kar, and S. Fidler, "Efficient interactive annotation of segmentation datasets with polygon-rnn++," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 859–868.

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[3] D. De Gregorio, A. Tonioni, G. Palli, and L. Di Stefano, "Semiautomatic labeling for deep learning in robotics," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 2, pp. 611–620, 2019.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 2009, pp. 248–255.

[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 2012, pp. 3354–3361.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] Y. S. Huang and C. Y. Suen, "The behavior-knowledge space method for combination of multiple classifiers," in *IEEE computer society conference on computer vision and pattern recognition.* Institute of Electrical Engineers Inc (IEEE), 1993, pp. 347–347.

[8] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[10] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[11] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms.* John Wiley & Sons, 2014.

[12] L. I. Kuncheva, J. C. Bezdek, and R. P. Duin, "Decision templates for multiple classifier fusion: an experimental comparison," *Pattern recognition*, vol. 34, no. 2, pp. 299–314, 2001.

[13] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 12 697–12 705.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[16] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[18] D. J. Miller and L. Yan, "Critic-driven ensemble classification," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2833–2844, 1999.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[23] M. D. Richard and R. P. Lippmann, "Neural network classifiers estimate bayesian a posteriori probabilities," *Neural computation*, vol. 3, no. 4, pp. 461–483, 1991.

[24] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[25] F. Scarselli and A. C. Tsoi, "Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results," *Neural networks*, vol. 11, no. 1, pp. 15–37, 1998.

[26] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," *arXiv preprint arXiv:1708.00489*, 2017.

[27] B. Settles, "Active learning literature survey," 2009.

[28] A. J. Sharkey, *Combining artificial neural nets: ensemble and modular multi-net systems*. Springer Science & Business Media, 2012.

[29] Y. Shen, H. Yun, Z. C. Lipton, Y. Kronrod, and A. Anandkumar, "Deep active learning for named entity recognition," *arXiv preprint arXiv:1707.05928*, 2017.

[30] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 529–10 538.

[31] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.

[32] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network," *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[34] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *Journal of machine learning research*, vol. 3, no. Dec, pp. 583–617, 2002.

[35] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[37] A. Tang, R. Tam, A. Cadrin-Chênevert, W. Guest, J. Chong, J. Barfett, L. Chepelev, R. Cairns, J. R. Mitchell, M. D. Cicero *et al.*, "Canadian association of radiologists white paper on artificial intelligence in radiology," *Canadian Association of Radiologists Journal*, vol. 69, no. 2, pp. 120–135, 2018.

[38] H. Wang, H. Shan, and A. Banerjee, "Bayesian cluster ensembles," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 4, no. 1, pp. 54–70, 2011.

[39] Y. Wang, X. Chen, Y. You, L. E. Li, B. Hariharan, M. Campbell, K. Q. Weinberger, and W.-L. Chao, "Train in germany, test in the usa: Making 3d object detectors generalize," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 713–11 723.

[40] A. Wissner-Gross, "Datasets over algorithms," *Edge. com. Retrieved*, vol. 8, 2016.

[41] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[42] D. Yoo and I. S. Kweon, "Learning loss for active learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 93–102.

[43] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, vol. 2, no. 5, p. 6, 2018.

[44] D. Zhou, J. Fang, X. Song, C. Guan, J. Yin, Y. Dai, and R. Yang, "Iou loss for

2d/3d object detection," in *2019 International Conference on 3D Vision (3DV)*. IEEE, 2019, pp. 85–94.