Thai traffic signs detection with conformal prediction and android deployment

by

Pongsiri Huang

Natthida Jutinandana

Thahseen Mohamed Rafeek

A report submitted in partial fulfillment of the requirements for
the degree of Bachelor of Engineering in
Computer Engineering

Project Advisor:

Dr. Anand Dersignh

Examination Committee:

Dr. Jerapong Rojanarowan, Dr. Wisuwat Plodpradista,
Assoc. Prof. Dr. Jiradech Kongthon, Assoc. Prof. Dr. Vorapoj Patanavijit,
Dr. Amulya Bhattarai, Dr. Ehsan Ali, Mrs. Sneha Paudel

Assumption University
Vincent Mary School of Engineering
Thailand
March 2024

Approved by Project Advisor:

Name: Dr. Anand Dersingh

Signature: 

Date: 7 March 2024

Plagiarism verified by:

Name: Mrs. Sneha Paudel

Signature: Sneha (turnitin: 11%)

Date: 07th March 2024

Contents

Abstract

This project utilizes the YOLOv5 deep learning model, specially trained on a dataset of Thai traffic signs, to detect traffic signs in Thailand. The dataset was compiled to reflect the diversity and specific characteristics of the region's road signs. A unique feature of our approach is the use of conformal prediction that are not widely applied or extensively covered in the literature related to traffic sign detection. These techniques introduced a new layer of sophistication to our project.

Additionally, we ventured into practical deployment by developing a mobile application that leverages our model. Despite adhering to standard design and development practices, integrating the model with conformal prediction techniques represents a significant enhancement.

The integration of conformal prediction is a standout element of our work, showcasing its utility in bolstering the confidence of deep learning models. This is crucial for applications demanding high levels of accuracy and reliability.

1. Introduction

Traffic sign detection focuses on identifying and classifying traffic signs within images. This process is crucial not only for recognizing the presence of traffic signs but also for distinguishing their types, such as stop signs, speed limit signs, or pedestrian crossing signs. The accuracy of these predictions is vital in various applications, including autonomous driving systems and driver assistance technologies. These systems depend on precise detection to correctly interpret and respond to road signs, thereby ensuring safe and informed driving decisions.

Despite the challenges faced building the system, many studies in traffic sign detection have achieved impressive performance, as measured by Mean Average Precision (mAP) across various datasets, there's a growing recognition of the need to assess not just the accuracy but also the certainty of these predictions. In critical applications like autonomous vehicles, the ability of a model to quantify its confidence in its predictions is just as important as its accuracy. This aspect, known as uncertainty quantification, plays a crucial role in enhancing the safety and reliability of automated systems. It provides a measure for these systems to understand the confidence level of their interpretations and respond accordingly.

In this report, we will take you through our journey of data collection, training of the YOLOv5 model, and the application of advanced techniques like conformal prediction. We will delve into the challenges we faced and the successes we achieved in implementing these technologies, leading to the development of a practical mobile application. This application not only showcases the real-world applicability of our model but also underscores its potential to enhance road safety and the efficiency of traffic systems.

1.1 Objective

In this project, we aim to achieve the following:

1. **Collect Thai Traffic Sign Dataset**: To ensure our model is tailored for specific regional requirements, we will collect Thai Traffic Sign Dataset. This dataset will provide essential data for training and testing our model, ensuring performace on Thai roadways, including local conditions, varying backgrounds (including roads and surroundings), diverse lighting scenarios, and the condition of traffic signs.

2. **Training the YOLOv5 Model**: Train the YOLOv5 model on Thai Traffic Sign dataset. We will also train on German Traffic Sign Dataset and Aquarium Dataset for analaysis.

3. **Apply Conformal prediction and Conformal Risk Control on Object detection**: We will study and apply Conformal Prediction techniques to object detection tasks, with a specific focus on the classification aspects. This approach will enable us to assess the reliability and certainty of the classifications made by the YOLOv5 model. However, the method should also generalize to other object detection model since Conformal prediction does not make any assumption about a model. We will also share our implementations on Traffic signs detection with some conformal prediction github

4. **Deployment on Android Devices**: Deploy a trained YOLOv5 model on Android devices.

These objectives are designed to explore object detection model, YOLOv5 in particular, and its practical applications, especially in the context of mobile technology and specific regional requirements. Additionally, the incorporation a Conformal prediction techniques represents a step towards increasing the reliability and accuracy of these models.

1.2 Chapter organization

Chapter 2 provides a brief overview of related work through a literature review. Chapter 3 outlines our methods and Chapter 4 discuss dataset and training of the detection model. Chapter 5 will be on the mobile application. Chapter 6, 7, 8, 9 will be we calibration, conformal prediction, conformal risk control and their implementation on object detction. Chapter 10 will be an extra chapter for author's reflection. Finally, Chapter 11 concludes the work by summarizing the achievements.

2. Literature Review

In this chapter, we delve into the existing body of work surrounding object detection, with a special emphasis on YOLOv5 and its application in traffic sign detection. Object detection, as a field, has seen rapid advancements in recent years, particularly with the introduction of models like YOLO (You Only Look Once), which have revolutionized the way machines perceive and interpret visual data. YOLOv5 stands out for its efficiency and accuracy, making it an ideal choice for real-time applications such as traffic sign detection. The advantages and disadvantages of models would also be cover in this chapter.

The latter part of the chapter shifts focus to the concept of uncertainty quantification in machine learning. This area has become increasingly relevant as the demand for reliable and safe AI systems grows. We will list few literature on different methods and approaches for quantifying prediction uncertainty in deep learning models. Finally, we introduce conformal prediction and conformal risk control as emerging techniques in the realm of uncertainty quantification. Conformal prediction, a method for making reliable and statistically valid predictions, offers a framework for understanding and quantifying the confidence in the predictions made by machine learning models. We will discuss its principles, applications, and how it can be integrated with object detection models, particularly in the context of traffic sign detection, to enhance their reliability and safety.

2.1 Object Detection

Object detection is a task that involves locating and identifying objects within an image. This field has seen a significant impact from deep learning models in recent years, with several notable architectures making substantial strides. Key among these are R-CNN [1], Fast R-CNN [2], Faster R-CNN [3], and RetinaNet [4], Detectron [5], which have been instrumental in advancing the field. Of particular relevance to our study is YOLOv5, which stands out as an efficient and effective solution in object detection.

In typical object detection task, input x is an image and label y consist of bounding box coordinates and corresponding class distribution. The prediction $\hat{y}$ include estimated probability of object presence, bounding box coordinates, and estimated class distributions. For the purposes of this work, our attention is on datasets where the classes are mutually exclusive.

Object detection models are usually evaluated and compared by Mean Average Precision (mAP) of predictions on public dataset.

2.2 YOLOv5

YOLOv5 [6] is a set of object detection models widely used in various tasks including Traffic Signs detection due to its one-stage nature, achiving real-time performance. Furthermore, YOLOv5 benefits from extensive community support and a robust ecosystem for deployment.

Numerous studies have explored various facets of object detection in the context of traffic sign recognition. For instance, the study in [7] tackles challenges such as lighting conditions, small traffic sign detection, and diverse backgrounds. Another research, [8], proposes architectural modifications to enhance the detection of small traffic signs. Morever, Thai traffic signs detection by YOLOv3 had been studied by [9].

However, these specific areas are not the primary focus of our project. We will be employing the standard YOLOv5 model as provided by Ultralytics. For the purposes of this report, it is essential to understand that YOLOv5 operates by segmenting the input image into a grid, within which it predicts the presence of objects, bounding boxes, and estimates class probabilities for each grid cell.

2.2.1 Resources

There is a significant community around YOLOv5, contributing various models and resources to meet diverse needs. In our project, we leverage the following key resources:

Ultralytics YOLOv5 Repository A pivotal resource for accessing various YOLOv5 models is the self-contained repository hosted on GitHub by Ultralytics. This repository, available at Officail YOLOv5 Github: `https://github.com/ultralytics/yolov5`, provides a comprehensive collection of YOLOv5 models, along with extensive documentation and support for implementation. The repository is regularly updated, ensuring access to the latest advancements and versions of YOLOv5.

Deployment on Android An invaluable resource for deploying YOLOv5 on Android devices is the GitHub repository, Torch Android Demo App `https://github.com/pytorch/android-demo-app`. This repository offers comprehensive details on the deployment process. While the core ideas and methods presented are still applicable, some aspects may require adjustments to align with the latest technological developments as of the writing of this paper. We will discuss the adjustments in Deployment section.

2.3 Uncertainty Quantification

Traditional machine learning models are typically designed to predict the most likely value without considering uncertainty. Uncertainty Quantification, however, is a task that focuses on identifying and measuring the levels of uncertainty within these predictions. This process is crucial for understanding the confidence and reliability of the model's outputs, especially in critical applications where decisions are based on these predictions. By quantifying uncertainty, we can better gauge the trustworthiness of the model and make more informed decisions based on its predictions.

Machine learning models, including those in deep learning, are often not *well-calibrated* [10] [11]. This mean that their output distributions may not accurately reflect true probabil-

ities, even when the outputs are estimated probabilities. However, these outputs do provide a heuristic notion of uncertainty, which is indicative but not definitive of the model's confidence.

In realm of deep learning, numerous approaches have been studied such as Bayesian inference approach [12], [13], [14], Monte-Carlo Dropblock [15]. In our study we will focus on recently arising emerging framework of Conformal prediction [16]. This approach is gaining attention for its potential to provide a robust and theoretically grounded way of handling uncertainty in deep learning models. Conformal prediction leverages heuristic notion of uncertainty to form well-calibrated predictions. It essentially transforms these heuristic uncertainties into more reliable and interpretable measures of confidence. We will attempt to apply Conformal prediction on YOLOv5 detection outputs to obtain prediction with probabilistic guarantee.

2.4 Conformal Prediction

Conformal Prediction (CP) is a statistical framework designed for quantifying the uncertainty of predictions in machine learning. Rooted in the works of [16], [17], [18], and [19], CP offers a way to generate prediction sets that come with a coverage guarantee, under the assumption of exchangeability.

In this report, our focus is on Split Conformal Prediction, a variant of CP. This method allows us to define a significance level, α, and apply CP as a post-processing step to obtain a *prediction set* with a *coverage* guarantee, as detailed in section 3.4. The coverage guarantee provided by conformal prediction can be mathematically expressed as:

$$1 - \alpha \leq \mathbb{P}(Y_{n+1} \in C^{\alpha}(X_{n+1})) \leq 1 - \alpha + \frac{1}{n+1}$$

This formula ensures that the true label Y_{n+1} of a new observation X_{n+1} will be contained in the conformal prediction set $C^{\alpha}(X_{n+1}))$ with a probability that respects the specified significance level α.

2.4.1 Conformal prediction on object detection

The methodologies involved in applying conformal prediction to object detection have been detailed in [20]. These techniques are primarily demonstrated in the context of bounding box regression. In contrast, our focus will be on the classification of traffic signs within each bounding box.

2.5 Conformal Risk Control

Conformal Risk Control (CRC) [21] extend the idea of Conformal prediction to a more general risk (or loss) function ℓ such as false negative rate, F1-score, and more, and CRC provide a guarantee:

$$\mathbb{E}\left[\ell\left(C_{\lambda}(X_{n+1}), Y_{n+1}\right)\right] \leq \alpha, \tag{2.1}$$

Where α is user-specified significance level.

2.6 Another approach

We were unaware of Learn then Test [22] until few weeks before the due date. They provide a framework generalize from Conformal Risk Control. The worked example also cover controlling multiple risks in object detection task. We hope to study the topic further in the future.

3. Methodology

In this section, we outline the methodology employed for the development of traffic sign detection and recognition system utilizing YOLOv5 object detection model. It includes project outline, data collection, create an android application, conformal prediction as well as the conformal risk control in object detection.

3.1 Project Outline

The following are the tasks which is required in order to build the system:

1. **Collect Thai Traffic Sign Dataset**: To ensure our model is tailored for specific regional requirements, we will collect Thai Traffic Sign Dataset. This dataset will provide essential data for training and testing our model, ensuring performace on Thai roadways, including local conditions, varying backgrounds (including roads and surroundings), diverse lighting scenarios, and the condition of traffic signs.

2. **Train YOLOv5 Model**: Train the YOLOv5 model on the Thai Traffic Sign dataset to tailor its detection capabilities to the specific nuances of Thai road signs. Additionally, we plan to extend this training to include the German Traffic Sign Dataset, allowing us to compare and contrast the model's performance across different dataset.

3. **Apply Conformal prediction and Conformal Risk Control on Object detection**: Explore and integrate Conformal Prediction techniques within object detection tasks, concentrating particularly on the classification aspects. This will allow us to gauge the reliability and certainty of classifications made by our YOLOv5 model. Importantly, this method is designed to be model-agnostic, meaning it should be applicable to other object detection models as well, given that Conformal Prediction doesn't rely on model-specific assumptions. Our implementation details and progress can be accessed at Traffic Signs detection with some conformal prediction GitHub.

4. **Deployment on Android Devices**: Deploy a trained YOLOv5 model on Android devices.

These are designed to explore object detection model, YOLOv5 in particular, and its practical applications, especially in the context of mobile technology and specific regional requirements. Additionally, the incorporation a Conformal prediction techniques represents a step towards increasing the reliability and accuracy of these models in real-world scenarios

3.2 Dataset Collection

Dataset is crucial for accurate detection, ensuring reliable model performance in traffic sign detection for mobile applications. Our dataset for Thai traffic sign detection combines self-collected images and data from online sources. self-collected images is to ensure a diverse and representative data set. Then, we used the labelImg tool for data annotation, drawing rectangular boxes around each sign for accuracy and ease of annotation. Big thanks to `https://www.youtube.com/watch?v=BBPnXvUc540` for letting us use traffic signs in the videos.

We split our data into 3 set:

1. Training data: 681 images used to train the YOLOv5 model.
2. Calibration data: 100 images used in conformalization step. Conformal algorithms learn statistics from this set to construct a well-calibrated prediction in a future.
3. Test data: 100 images used to evaluate model performance during development.

Our dataset consists of the following classes:

- Speed Limit 20, 30, 40, 45, 50, 60, 70, 80, 90, 100, 120
- Traffic Lights (yellow)
- Pedestrian Crossing (blue, yellow)
- No U-turn (red)
- U-turn ahead (blue)
- Turn Left Ahead (blue)
- No Left Turns (red)
- Turn Right Ahead (blue)
- No Right Turns (red)
- Go Ahead (blue)
- Give Way (red)
- Stop (red hexagon)
- No Parking
- No Entry
- No Stopping
- Roundabout (blue, yellow, red)
- Keep Right (blue)
- Keep Left Or Right (blue)
- Parking (blue)
- Right Curve (yellow)
- Keep Left (blue)
- T Intersection (yellow)
- Left Curve (yellow)
- No Crossing (red)

3.3 Android Application

Deploying our system in a user-accessible format necessitates the creation of an application. This application comprises three major components: the backbone, the model, the user interface design, and the database, which holds user data. The backbone of our system is the YOLOv5 model. Figma is used for designing user interface prototype. Android Studio serves as the platform to integrate these components, connecting the YOLOv5 model with the user interface and the database.

3.3.1 Deploying YOLOv5 model on Android

For deploying the YOLOv5 model on Android, we adapted the Torch Android Demo App Tutorial, see 2.2.1, making necessary adjustments for compatibility with the newer version

of YOLO. We will elaborate on this process, including the specific adjustments made, in Chapter 5.

3.3.2 User Interface

The user interface (UI) design is the point of interaction between the application and the user's smart device. This includes all elements that the user interacts with, such as the display screen, on-screen keyboard, buttons, menu icons, and other interactive components.

Our application's UI design prioritizes simplicity and low complexity. To aid user understanding, a tutorial is provided for additional clarification. The detailed discussion on this aspect will be covered in Chapter 5.

3.4 Conformal prediction

Conformal prediction (CP) is a framework on quantifying uncertainty. It evaluate the new prediction base on model past predictions and provide a well-calibrated prediction, described by the following theorem:

Theorem 1 (Coverage)**.** *Let input-label pair* $\{(X_i, Y_i)\}_{i=1}^{n+1}$ *be exchangeable. Consider a scenario where* $\{(X_i, Y_i)\}_{i=1}^{n}$ *are used in calibration step then we construct a prediction set* $C^{\alpha}(X_{n+1})$ *on a future input* X_{n+1}*, the coverage is guaranteed:*

$$1 - \alpha \leq \mathbb{P}(Y_{n+1} \in C^{\alpha}(X_{n+1})) \leq 1 - \alpha + \frac{1}{n+1}$$

Where α *is user-defined significance level. The first n samples* $\{(X_i, Y_i)\}_{i=1}^{n}$ *are usually called calibration set.*

Since we are focusing on classification of traffic signs, the guarantee can be read as "the probability of a correct traffic sign to be in prediction set is greater than $1 - \alpha$ (e.g. 0.95) and lower than $1 - \alpha + \frac{1}{n+1}$ (e.g. 0.96) as long as the order which we observe a traffic sign does affect the type of the traffic sign".

Calibration set. Conformal prediction introduce Calibration set: a split of dataset similar to how we split training, validation and test set. Conformal Prediction require a calibration set which should be unseen to model and independent of training and validation set. Calibration set is used to evaluate a model and compute statistic of the model given unseen inputs. Note that theoretically we only require calibration set and test set to be exchangeable, not necessarily need to be independent.

Nonconformity measure. Nonconformity measure is a function $s(y, \hat{y})$ that measure how strange or nonconforming a new example is to the previous examples. When making prediction on a new example, we can compute nonconformity score of the new example and compare to nonconformity scores obtained from previous examples. nonconformity measure can be as simple as absolute distance: $|y - \hat{y}|$ in regression, or estimated probabilities of the correct class $\hat{f}(x)_Y$ in classification.

We will use Adaptive Prediction Set (APS) and Regularized Adaptive Prediction Set (RAPS) as our nonconformity measure described in [18] and [19], respectively. We will attempt to describe them intuitively and Least Ambiguous set-valued Classifier (LAC) method just for intuition in section 7.

3.4.1 Overall Process of Conformal prediction

Given a trained model and calibration set. We can evaluate model on calibration set and obtain statistic about the model given calibration set. Then we use the statistic to construct a set prediction during inference.

1. Preparation: we will need a trained model $\hat{f}$ and a calibration set.
2. *Conformalization*: calculate nonconformity scores $\{s_i, ..., s_n\}$ from $\hat{f}$ on calibration set. Then calculate a quantile at α of the scores $q_\alpha = \left\lceil Quantile\left(s_1, ..., s_n; \frac{\lceil (n+1)(1-\alpha) \rceil}{n}\right)\right\rceil$.
3. *Inference*: calculate nonconformity score of the new prediction and compare it with the previously observed scores.

$$C^\alpha(X) = \{y : condition\ by\ comparing\ scores\}$$

Following these steps, prediction set should obtain a probabilistic coverage of $1-\alpha$. Note that coverage is always guaranteed with no concern over model performance or choice of nonconformity measure. However, if the underlying model is poor or nonconformity measure is not well-suited. The prediction set $C(X_{n+1})$ will be too large to be informative.

3.4.2 Conformal Risk Control

Conformal Risk Control (CRC) [21] extend the idea of Conformal prediction to a more general risk (or loss) function ℓ such as false negative rate, F1-score, and more, and CRC provide a guarantee:

$$\mathbb{E}\left[\ell\left(C_\lambda(X_{n+1}), Y_{n+1}\right)\right] \leq \alpha, \tag{3.1}$$

In the paper, they give an example of CRC on multilabel classification where loss function $\ell\left(C_\lambda(X_{n+1}), Y_{n+1}\right) = 1 - |Y_{n+1} \cap C_\lambda(X_{n+1})| \,/\, |Y_{n+1}|$ and *prediction set* includes classes with high estimated probabilities $C_\lambda = \{k : f(X)_k \geq 1 - \lambda\}$. The example illustrate that as λ grows, they include more classes in the C_λ and loss function ℓ become smaller. In other word, C_λ become more conservative toward risk.

This approach is similar to traditional conformal prediction in that as the prediction set becomes larger, it tends to be less informative. The larger set, while reducing the risk of incorrect exclusions, may include more classes than necessary, leading to a trade-off between risk control and the informativeness of the predictions.

The objective of CRC is straightforward: it aims to estimate the smallest λ that fulfills the condition set out in (3.1), using a finite-sample approximation denoted by $\hat{\lambda}$. The method for computing $\hat{\lambda}$ is formalized as follows:

$$\hat{\lambda} = \inf\left\{\lambda : \frac{n}{n+1}\hat{R}_n(\lambda) + \frac{B}{n+1} \leq \alpha\right\} \tag{3.2}$$

Where n is size of calibration set and $\hat{R}_n(\lambda) = (L_1(\lambda) + ... + L_n(\lambda)/n$ where $L_i : \Lambda \to (-\infty, B]$ is a random function bounded by B. In practice L_i would be loss value evaluated on model on input-label pair $L_i(\lambda) = \ell\left(C(X_i), Y_i\right)$.

3.5 Conformal prediction and Conformal Risk Control in object detection

[20] outline *box-wise* and *image-wise* approaches. We will focus on box-wise approach where we apply conformal prediction on only positive prediction. This method rely on matching predicted and truth bounding box. Then conformal prediction is applied on the class distributions of matched bounding boxes. The algorithm for the *box-wise* approach is summarized as follows:

1. *Preparation:* We will need an object detection model that output bounding box and estimated class distribution $\hat{f}(X)$ and a calibration input-label pairs $\{(x_i, y_i)\}_{i=1}^{n+1}$.

2. *Match bounding boxes:* Passing calibration set into the model obtaining $\{\hat{y}\}_{i=1}^{n}$. Then bounding boxes are matched between each truth and predicted labels a set of obtaining matched bounding boxes M=$\{(y, \hat{y}) : \textit{box matching algorithm}\}$. We will use Intersection Over Union (IoU) threshold into Hungarian matching with IoU cost as a box matching algorithm.

3. *Conformalization:* calculate nonconformity scores $s_i, ..., s_n$ from predicted probabilities and ground-truths $\{(y, \hat{y})\}_{i=1}^{n}$.

4. *Inference:* calculate nonconformity score(s) of each new predicted box and compare it with the previously observed scores.

This *box-wise* approach is straightforward and intuitive, applying Conformal Prediction to each detected box without requiring new theorems or proofs. If object presence is correctly predicted, the classification of each bounding box is guaranteed to achieve $1 - \alpha$ coverage. However, if an object is present but the model fails to predict its presence, the *box-wise* approach is not applied, leading to a false negative. Thus, while this approach aids in accurately classifying detected objects, it does not address the model's potential to miss detecting an object. This underscores the approach's emphasis on classification accuracy of detected objects over the improvement of object detection capabilities.

Conformal Risk Control to control false negative rate

In this section, we adopt a straightforward strategy to manage the risk of omitting pixels, operationalized through a confidence threshold. We introduce a risk function designed to quantify the likelihood of missing pixels as follows:

$$L_i^{FNR}(\lambda) = 1 - \frac{|Y_{n+1} \cap C\lambda(Xn+1)|}{|Y_{n+1}|} \tag{3.3}$$

To construct a prediction set, we filter bounding boxes based on their confidence level:

$$C\lambda(Xi) = y : f(X_i)_y \geq 1 - \lambda \tag{3.4}$$

This approach is underpinned by a guarantee that:

$$\mathbb{E}\left[L_{n+1}^{FNR}(\lambda)\right] \leq \alpha, \tag{3.5}$$

The intuitive approach we've adopted involves selecting a confidence threshold, $\hat{\lambda}$, to ensure that, on average, the predicted bounding boxes do not miss more than an α fraction of pixels. This method aims to minimize the risk of overlooking relevant information in the detection process.

It's important to recognize that the false negative risk defined in (8.1) might not be the most suitable risk function for every scenario, largely because of its scale invariance. Such invariance implies that failing to detect a small traffic sign significantly elevates the risk value, potentially skewing the risk assessment. Despite this, we chose this function for its straightforwardness and ease of use.

Furthermore, our method's reliance on a confidence threshold to mitigate the risk of missing pixels has its limitations. Specifically, the efficacy of this approach is inherently linked to the model's capability to accurately predict bounding boxes. A high confidence threshold cannot compensate for poorly predicted bounding boxes, underscoring a fundamental limitation in merely adjusting the confidence threshold.

More sophisticated applications of Conformal Risk Control (CRC) extend beyond simple threshold adjustments. For instance, as explored in [20], CRC can be employed to calculate the margin of error in bounding box predictions, offering a more nuanced and effective way to address prediction inaccuracies. This advanced application of CRC underscores the potential for more refined approaches to enhancing model reliability and accuracy in object detection tasks.

Overview during Inference

During the inference phase, the conformalized outputs are simply additional post-processing step. In box-wise conformal prediction, each predicted bounding box is treated as a separate classification task. The conformal risk control parameter, $\hat{\lambda}$ is used to filter out bounding boxes with low confidence scores.

Fig. 3.1: This flowchart provides an overview of the inference process with conformal prediction. On the right, we have the standard output from YOLO's prediction, while on the left, the output reflects box-wise conformal prediction. The conformal risk control is primarily utilized for determining the confidence threshold in this process.

4. Thai Traffic signs Dataset and Training of YOLOv5

This section dive in the training set of Thai traffic sign dataset. Then, we describe the process of finetuning pretrained YOLOv5 object detection model from official github repository. This is probably the most practical way to achieve high performance object detector with very little difficulty.

4.0.1 Training set analysis

Before training the model it is good to investigate the dataset. We will simply show few samples from dataset and frequency of class.

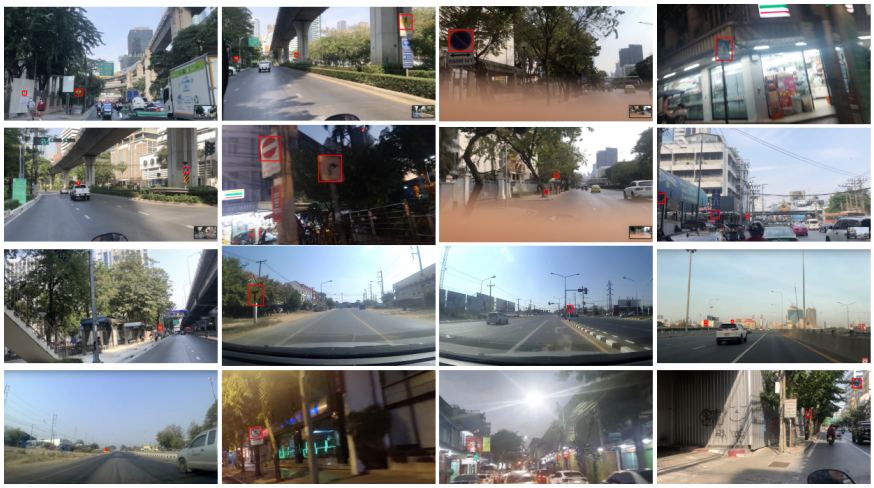

Fig. 4.1: 16 randomly sampled images with bounding boxes from training set

Figure 4.1 displays a collection of images randomly selected from our training set, each annotated with bounding boxes, no class label. This samples show the diversity in viewpoints and lighting conditions captured in our dataset. It is evident from the assortment that the dataset predominantly features daytime images. Furthermore, an examination of these samples prompts an important consideration regarding the relevance of traffic signs at various distances; specifically, it highlights the challenge of determining the significance of traffic signs that appear too distant to be meaningful.

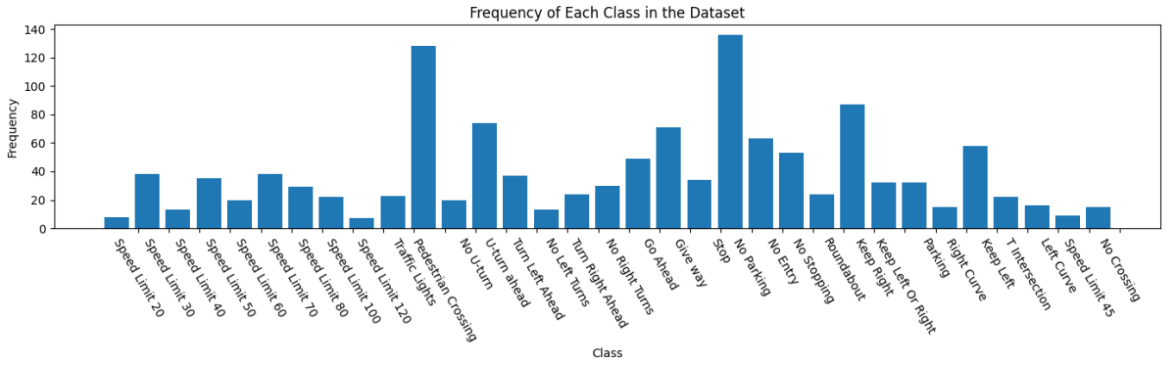

Fig. 4.2: This plot show how frequency each class appear in our That dataset.

Figure 4.2 reveals significant class imbalances in our dataset, which isn't inherently problematic. This could reflect the actual distribution of traffic signs in our region. For example, 'No Parking' signs are prevalent in Bangkok, whereas 'Speed Limit 20' and 'Speed Limit 45' signs are relatively rare. Additionally, the absence of 'Speed Limit 120' signs can be attributed to the lack of highway footage in our data collection. However, for the purpose of generalization and building a model that accurately represents real-world conditions, it is crucial to ensure that our dataset encompasses a more diverse and representative sample of traffic signs.

4.1 Training YOLOv5 detector

Training yolov5 detector is almost straight forward thanks to invaluable resources from Ultralytics and the communities, https://github.com/ultralytics/yolov5, for providing pretrain weights and reduce our work to one line of function call (if don't count setting parameters):

Code 4.1.1: Command to train yolov5m

```
python yolov5/train.py --weights yolov5m.pt ...parameters (e.g. --data
↪  traffic_sign_dataset.yaml --freeze 10)...
```

We use default parameters and pretty standard augmentations. In fact, we just use default augmentation from the config file except we do not fliplr, since flipping image could confuse left and right traffic signs. The most important thing to note here is that we freeze the backbone.

4.2 Deployment of YOLOv5 on Android Devices

We successfully deployed YOLOv5 object detection model on Android platforms. However, the integration of conformal prediction presented challenges, primarily due to the numerical discrepancies across different hardware configurations. Consequently, this section is positioned prior to the discussion on conformal prediction.

This section will detail the process and methodologies employed in deploying the YOLOv5 object detection model on Android devices.

5. Implementation and Tutorial of our Android app

5.1 Deploy YOLOv5 on android application

This section discuss how we deploy YOLOv5 onto mobile application.

We simply follow the tutorial `https://github.com/pytorch/android-demo-app/tree/master/ObjectDetection`, but with newer version of YOLOv5 (As in 26-February-2023). Hence, we will only discuss what we do differently from them.

1. Clone the YOLOv5 and android-demo-app repositories and follow their respective procedure.

   ```
   git clone https://github.com/ultralytics/yolov5
   git clone https://github.com/pytorch/android-demo-app.git
   ```

2. We use this specific commit. Then just install requirements.txt:

   ```
   cd yolov5
   git reset --hard 9cdbd1de6b64193b444365982427d5f6d48d6a97
   pip install -r requirements.txt
   ```

3. Find the export_torchscript function under yolov5/export.py folder and replace it as follows:

   ```python
   @try_export
   def export_torchscript(model, im, file, optimize, prefix=colorstr("TorchScript:")):
       # YOLOv5 TorchScript model export
       LOGGER.info(f"\n{prefix} starting export with torch {torch.__version__}...")
       f = file.with_suffix(".torchscript.pt")
       fl = file.with_suffix(".torchscript.ptl")
       ts = torch.jit.trace(model, im, strict=False)
       d = {"shape": im.shape, "stride": int(max(model.stride)), "names": model.names}
       extra_files = {"config.txt": json.dumps(d)}  # torch._C.ExtraFilesMap()
       if optimize:  # https://pytorch.org/tutorials/recipes/mobile_interpreter.html
           optimize_for_mobile(ts)._save_for_lite_interpreter(
           str(f), _extra_files=extra_files
           )
       else:
           ts.save(str(f), _extra_files=extra_files)
           ts._save_for_lite_interpreter(str(fl))
       return f, None
   ```

4. Export the trained YOLOv5

```
python export.py --weights yolov5s.pt --include torchscript
```

5. In Android Studio, use the new version of lite interpreters in your build.gradle file:

```
implementation 'org.pytorch:pytorch_android_lite:1.13.0'
implementation 'org.pytorch:pytorch_android_torchvision_lite:1.13.0'
```

5.2 Tutorial on Mobile app

This section will acts as the guideline for users, clarifying their confusion in terms of UI design of the application.

As shown in Figure 5.1, after login and sign up it will direct users to the welcome page, and then the main page which shows all the options for users to select. The options are, real-time, camera, upload, and tutorial.

- The real-time option allows users to detect Thai traffic in real-time even while driving or walking. The users can put their phone in front of the Thai traffic sign images, it will detect and show the name of the sign automatically.
- Camera option application will open a camera. User can take a photo of the sign and forward it into the detection model.
- Upload option is where users can upload photos from their phone's gallery to the detection model.
- Tutorial lead user to tutorial page.

Figures 5.2, 5.3, 5.4, and 5.5 are some of the UI designs of our mobile app.

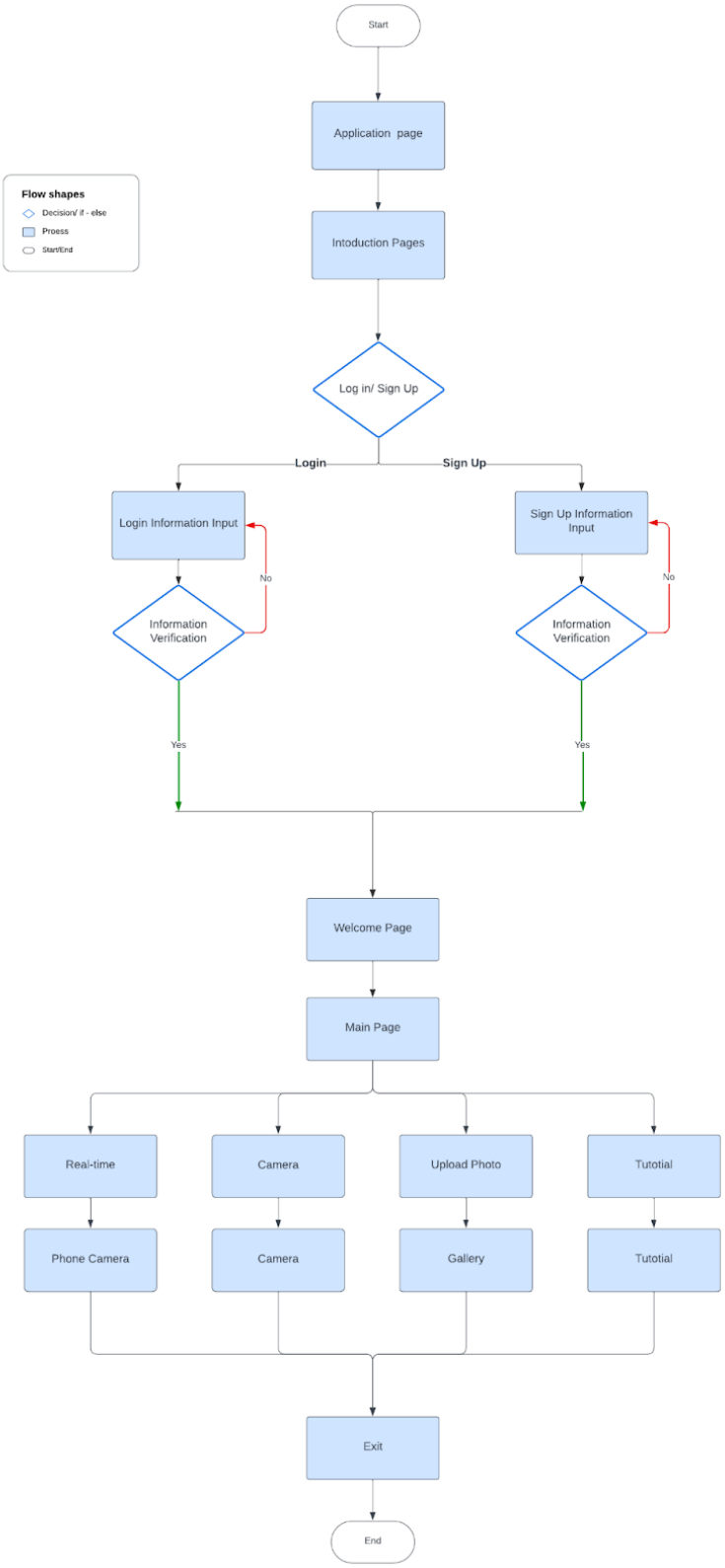

Fig. 5.1: Activity diagram of the mobile application start on the top.

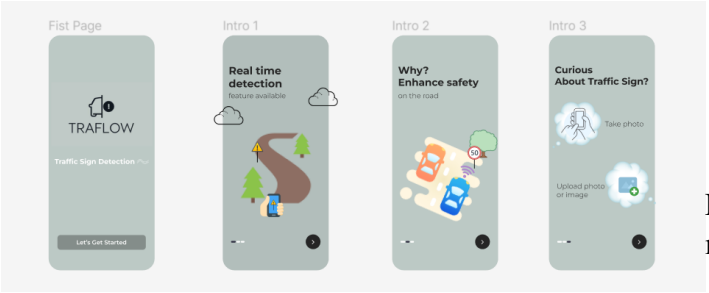

Fig. 5.2: First page when opening our mobile app.

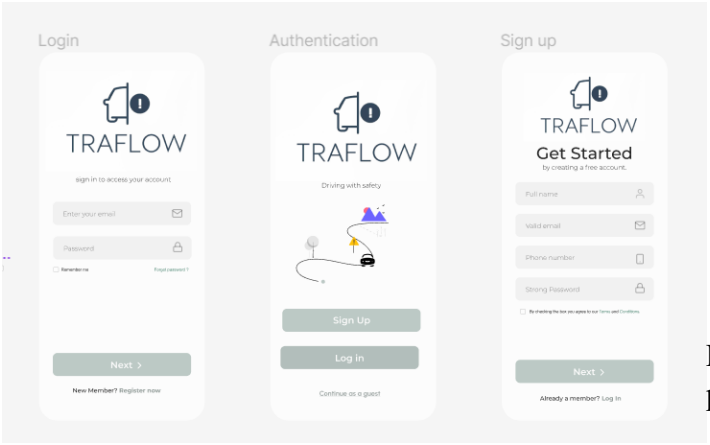

Fig. 5.3: Login, sign up, authentication pages.

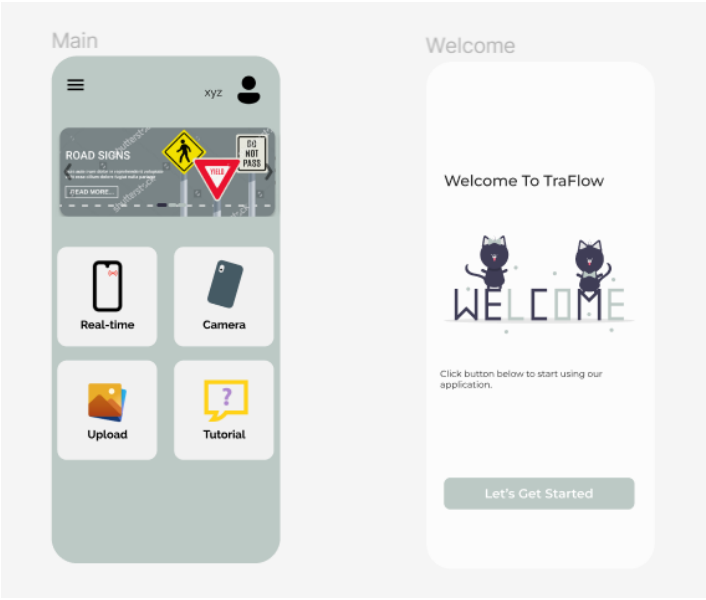

Fig. 5.4: Main and welcome page.

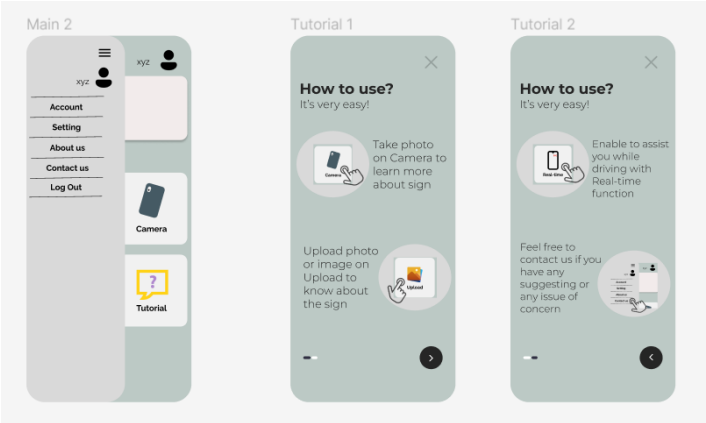

Fig. 5.5: Tutorial pages.

6. Is MNIST classifier well-calibrated?

This section is here to verify that deep learning model are generally not well-calibrated despite having high accuracy.

We do a simple experiment: we trained two neural networks on the MNIST dataset to see how well-calibrated they were. Our smaller network showed pretty good calibration with an accuracy of 0.98, ECE of 0.0065 and an SCE of 0.0357. The larger network wasn't far off either, with an accuracy of 0.98, ECE of 0.0101 and an SCE of 0.0404. When we looked at the reliability diagrams in figure 6.1 and 6.2, they confirmed that both networks did a decent job at calibration, although the larger one seemed a bit overconfident.

However, we should remember that MNIST isn considered simple and straightforward dataset, and models usually score high on it. So, while our MNIST classifiers seem to be on the right track, this isn't always the case with deep learning models in general. Other studies often find that when you throw more complex data at these models, their calibration isn't quite as sharp.

It's also worth noting that even though both of our models hit the same accuracy level, right down to the second decimal, they showed different calibration scores. This is a heads-up that even with identical accuracy, models can behave quite differently when it comes to calibration. In practice, this means a model's accuracy isn't the only thing to watch out for—its calibration could still be off, which could lead to trouble in more complex or critical applications.

A Supplementary Experiment on GTSRB: In addition to MNIST, we trained a classifier for the GTSRB (German Traffic Sign Recognition Benchmark) dataset. The reliability diagram, illustrated in figure 6.3, reveals that, despite a high test set accuracy of 0.98, the GTSRB classifier lacks proper calibration.

Further exploration involved analyzing classification distributions from a YOLOv5 detector trained on traffic sign detection. Given its comparatively lower accuracy, it might be posited that the model's calibration is inadequate due to insufficient training. Therefore, we have chosen not to include these experimental results.

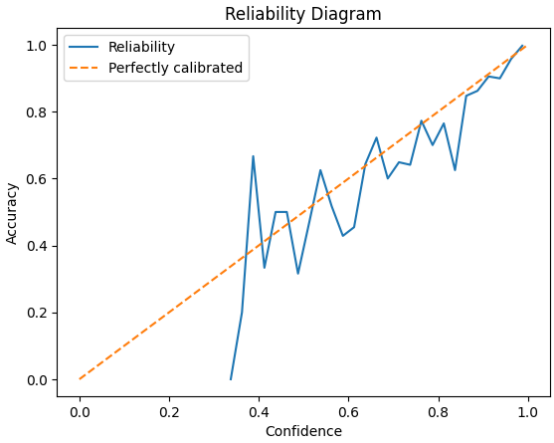

Fig. 6.1: Reliability diagram for small network. Showing that it is quite well-calibrated.

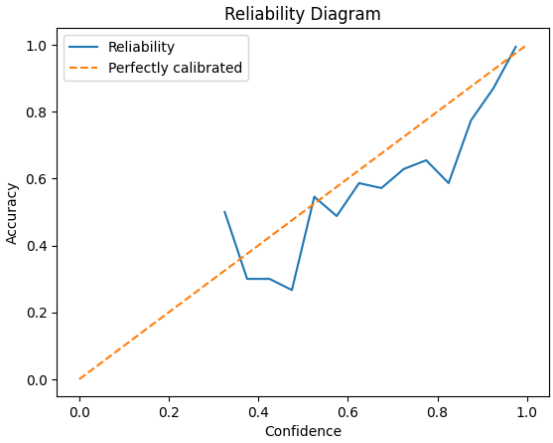

Fig. 6.2: Reliability diagram for larger network. Showing that they are quite well-calibrated trending toward overconfidence

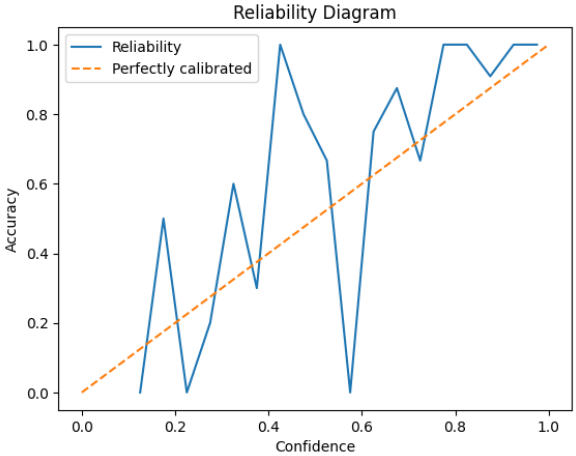

Fig. 6.3: Reliability diagram of GTSRB classifier

7. Implementing Conformal prediction for classification problem

As mentioned in section 3.5, box-wise approach treat each predicted bounding box as a separated classification problem. Hence, in this section, we start by implementing Conformal prediction for classification. We will also go through some examples of how each nonconformity scores work.

It is important to note that conformal prediction guarantee a coverage regardless of the choice of nonconformity measure. However, selecting a good nonconformity measure will result in a smaller prediction set, which can be thought of as more informative.

7.0.1 Least Ambiguous set-valued Classifier

Least Ambiguous set-valued Classifier (LAC) is a simple nonconformity measure that only consider confidence of the most likely label, defined as

$$s_i(y_i, \hat{y}_i) = \hat{y}_{i,k} \tag{7.1}$$

where $\hat{y}_{i,k} = \hat{f}(X_i)_k$ is an estimated probabilities of the correct label. Then the prediction set of new input is form by

$$C^{\alpha}(X) = \{y : \hat{y}_k > q_{1-\alpha}\} \tag{7.2}$$

Intuitively, the set $\{s_i\}_{i=1}^{n}$ can be interpreted as indicating "how confident the model is in the correct label," with a larger $s = \hat{y}_k$ implying greater confidence. The term $\hat{q}_{1-\alpha}$ can be interpreted as the quantile of confidence at $1-\alpha$. If the confidence is greater than $\hat{q}_{0.05}$, it suggests that the model is more confident in its prediction than in 95% of its past predictions for the correct class. Therefore, for $\hat{y}_k$ greater than $\hat{q}_{0.05}$, it should be included in the prediction set. And the following codeblock 7.0.1 is a simplified Python implementation of LAC method.

Code 7.0.1: Implementation of Least Ambiguous set-valued Classifier (LAC)

```
# Get data. X is input sand y is targets.
X_cal, y_cal = calibration_dataset.get_Xy()
X_test, _ = test_dataset.get_Xy()
#--- LAC method ---
# gather nonconformity scores.
pred_cal = classifier.forward(X_cal) # predictions on calibration set.
nonconformity_scores = predictions[range(len(targets)), y_cal]

# determine quantile of nonconformity scores.
qhat = np.quantile(nonconformity_scores, q=ALPHA) # find quantile at alpha

# inference
pred_test = classifier.forward(X_test)
prediction_set = pred_test > qhat # coverage achieved!
```

Next, Consider two situations for a binary classifier with 1) perfect accuracy of 100% but underconfidence and 2) poor accuracy of 0% but overconfidence. We will show that LAC method is able to achieve coverage of $1-\alpha$ as promised in (1) in both cases. Figure 7.1 illustrates the distribution of estimated probabilities and the nonconformity quantile for each scenario.

First, with the classifier of 100% accuracy: Since the classifier is always accurate, $\hat{q}_{1-\alpha}$ of the nonconformity measure will always exceed the estimated probabilities of the incorrect class. Consequently, the prediction set will never include the incorrect class. For the correct class, the prediction set will include it $1-\alpha$ of the time.

In the second scenario, with a classifier of 0% accuracy: The correct class will still be included $1-\alpha$ of the time, since $\hat{q}_{1-\alpha}$ is with respect to the correct class. However, $\hat{q}_{1-\alpha}$ will always be less than that of the incorrect class, thus they are always included in the prediction set. This situation, where both classes are included in a binary classification, does not provide much informative value.

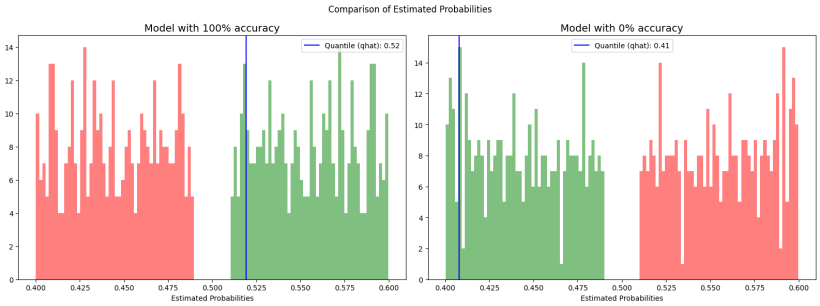

Fig. 7.1: The figure presents two histogram plots illustrating the output distributions of two hypothetical binary classifiers. The x-axis represents the estimated probabilities of the model, and the y-axis shows the binned count of each confidence value. The green histogram represents the distribution of confidence in the correct class, as referenced from known labels, while the red histogram represents the confidence in the incorrect class. The vertical blue line indicates the quantile of the nonconformity measure. These plots compare models with extreme accuracies: one with 100% accuracy and the other with 0% accuracy, visually demonstrating the described concepts.

Adaptive Prediction Set

Unfortunately, this section is not written due to time limitation. However, the implementation and figures can still be found in github: https://github.com/PongsiriH/Traffic-Signs-detection-with-some-conformal-prediction

Regularized Adaptive Prediction Set

Unfortunately, this section is not written due to time limitation. However, the implementation and figures can still be found in github: https://github.com/PongsiriH/Traffic-Signs-detection-with-some-conformal-prediction

8. Implementing Conformal prediction and Conformal risk control for object detection.

This chapter describe the core implementation of Conformal prediction and Conformal Risk Control for object detection. Section 8.1 discuss box-wise approach of conformal prediction to achieve classification coverage, and section 8.2 discuss using conformal risk control to achieve bounding box coverage. Then section 8.3 merge the two implementation into one pipeline.

8.1 Conformal prediction to guarantee classification coverage of each bounding boxes

This section show the implementation of box-wise method conformal prediction on object detection. We assume that each method of conformal prediction for classification described in previous chapter 7 are put into these functions:

Code 8.1.1: Pseudo-Python code of conformal prediction functions

```
def gather_nonconformity_scores(labels, predictions, method):
    if method not in ['lac', 'aps', 'raps']:
        raise error
    else:
        # nonconformity measure as describe in chapter 6.
    ...

def quantile(nonconformity_scores, alpha):
    ...

def form_prediction_set(inputs, alpha, method):
    if method not in ['lac', 'aps', 'raps']:
        raise error
    ...
    else:
        # construct prediction set as describe in chapter 6.
```

To begin, we gather nonconformity scores of each bounding box that get matched with ground-truth box. Then quantile $\hat{q}$ is determined and used during inference. Note that box matching algorithm is only needed for gathering nonconformity scores.

Code 8.1.2: Gathering nonconformity scores from bounding boxes with hungarian matching.

```
ALPHA = 0.05 # user-specifies.
METHOD = 'aps'

# -- box-wise approach.
# 1) gather nonconformity scores from calibration dataset.
nonconformity_scores = []
for image, label in calibration_dataset: # for each image-label pairs.
    # 1.1) forward yolo model. (can be any model)
    pred = yolo.forward(image)
    pred = confidence_thresholding(pred, conf_thresh)
    pred = nms(pred, nms_thresh)

    # 1.2) match ground-truth box with prediction box.
    iou_matrix = iou(label['xyxy'], pred['xyxy'])
    i, j = matching_iou_hungarian(iou_matrix, iou_thresh)
    gts_class = label['class_label'][i]
    pred_dist = pred['class_dist'][j]

    # 1.3) gather nonconformity scores of each box.
    ncs = gather_nonconformity_scores(gts_class, pred_dist, METHOD)
    nonconformity_scores.extend(ncs)

# 2) calculate quantile of nonconformity_scores
if METHOD == 'lac': ALPHA = 1 - ALPHA # due to the way we implement lac method.
qhat = quantile(nonconformity_scores, ALPHA)

# 3) inference
image, _ = test_dataset.sample(1)
pred = yolo.forward(image)
pred = confidence_thresholding(pred, conf_thresh)
pred = nms(pred, nms_thresh)
pred['prediction_set'] = form_prediction_set(pred['class_dist'], ALPHA, METHOD)
```

8.2 Conformal Risk Control to find the "right" confidence threshold

This section will describe how we utilize Conformal Risk Control (CRC) to determine the confidence threshold for filtering YOLOv5 detection's output, aiming to control the false negative bounding boxes. The first step in this process is to define a risk function. For our purposes, we will simply use the risk of missing pixels of the ground-truth, or the false negative rate. Although this may not be the best risk function for this task, we will stick with it due to simplicity. The false negative rate (FNR) is defined as

$$L_i^{FNR}(\lambda) = 1 - \frac{|Y_{n+1} \cap C_\lambda(X_{n+1})|}{|Y_{n+1}|} \tag{8.1}$$

Then we determine $\hat{\lambda}$ by

$$\hat{\lambda} = \inf\left\{\lambda : \frac{n}{n+1}\hat{R}_n(\lambda) + \frac{B}{n+1} \leq \alpha\right\} \tag{8.2}$$

During inference a prediction set is constructed by filtering out bounding boxes with low confidence:

$$C_\lambda(X_i) = \{y : f(X_i)_y \geq 1 - \lambda\} \tag{8.3}$$

And (3.1) guarantee that

$$\mathbb{E}\left[L_{n+1}^{FNR}(\lambda)\right] \leq \alpha, \tag{8.4}$$

In other words, we select a confidence threshold $\hat{\lambda}$ that aims not to miss more than α pixels on average. This allow for more intuitive way of setting a confidence threshold rather than just experimenting with different values.

FNR normalize the loss for each target before adding them up, which essentially makes our evaluation size-agnostic. This ensures that our measurements aren't unfairly influenced by the size of the objects we're trying to detect. However, this approach has a flip side; if the detection system fails to recognize a small object, like a distant traffic sign, the risk can spike significantly, leading to overly conservative $\hat{\lambda}$. Another limitation of using FNR to set confidence thresholds. Its behavior isn't solely influenced by the threshold. For example, correct object presenceness with inaccurate bounding boxes leads to high FNR, but this isn't a confidence threshold issue. A better risk function could be employed to improve the predictions.

Next, we will guide you through Python pseudo-code of what we just described. First, we define the false negative rate risk:

Code 8.2.1: define false negative rate risk

```
def false_negative_risk(yolo, dataset, conf_thresh, nms_thresh):
    # False negative in this case the pixels missed by predictions.
    scores = 0
    num_samples = 0
    for image, label in dataset: # for each image-label pairs.
        # 1) forward and post-process prediction.
        pred = yolo.forward(image)
        pred = confidence_thresholding(pred, conf_thresh)
        pred = nms(pred, nms_thresh)

        # 2) calculate false negative risk.
        pred_masks = create_binary_mask(pred['xyxy'])
        label_masks = create_binary_mask(label['xyxy'])
        for label_mask in label_masks:
            overlapping_pixels = area(pred_masks * label_mask)
            count_true_pixels = area(label_mask)
            scores += overlapping_pixels / count_true_pixels
            num_samples += 1
    return 1 - scores / num_samples
```

Then, we solve for $\hat{\lambda}$ using brenqt

Code 8.2.2: determine lambda_hat using brentq

```
from scipy.optimize import brentq

def fn_lambda_hat(lamb):
    # assume that nms_thresh and alpha are defined globally.
    conf_thresh = 1 - lamb
    n = len(calibration_dataset) # size of calibration set.
    B = 1 # upper bound for false_negative_risk.
    fnr = false_negative_risk(yolo, calibration_dataset, conf_thresh, nms_thresh)
    return fnr - ( (n+1)/n*alpha - B/(n+1) )

lamda_hat = brenqt(fn_lambda_hat, 0, 1)
```

We should use the $\hat{\lambda}$ during inference and risk will be controlled.

Code 8.2.3: forwarding with the optimal lambda$_hat$

```
conf_threhs = 1 - lamda_hat
pred = yolo.forward(unseen_image)
pred = confidence_thresholding(pred, conf_thresh)
pred = nms(pred, nms_thresh)
```

8.3 Conformalized Pipeline

Assuming we already have the nonconformity quantile $\hat{q}_{coverage}$ and $\hat{\lambda}_{conf}$ from section 8.1 and 8.2, respectively. We can easily integrate the two together.

Code 8.3.1: conformalized pipeline

```
ALPHA_COVERAGE = 0.05 # alpha for box-wise coverage
ALPHA_FNR = 0.2 # alpha for controling false negative rate

conf_threhs = 1 - lamda_hat
pred = yolo.forward(unseen_image)
pred = confidence_thresholding(pred, conf_thresh)
pred = nms(pred, nms_thresh)
pred['prediction_set'] = form_prediction_set(pred['class_dist'], ALPHA, METHOD)
```

9. Results and Analysis of YOLOv5's detections and conformalized predictions

This section will evaluate and discuss results of YOLOv5's detections and conformalized predictions on 1) German Traffic Signs Dataset 2) Thai Traffic Signs dataset. We evaluate the base YOLOv5 by mean average precision (mAP), then we show that coverage is guaranteed box-wise and risk is controlled pixel-wise as promised in chapter 8. Then we analyze the conformalized predictions. Thanks to YOLO communities, evaluating mean average precision for YOLO is a one-liner:

Code 9.0.1: Ultralytics Benchmarking YOLO model

```
python yolov5/benchmarks.py --weights "path/to/your/weights/best.pt" --data
↪  "path/to/your/dataset.yaml"
```

9.1 Detection and conformalized Detection on German Traffic Signs Dataset

Training set of GTSDB has 600 images. Since, GTSDB is dataset for competition, it does not provide ground-truth for test set. We divide the training set into 3 splits: 300 training, 100 calibration, 200 testing. We use significance level $\alpha_c = 0.90$ for classification and false negative rate risk level $\alpha_{fnr} = 0.10, 0.40, 0.80$.

First, we aim to present a comparison between the ground-truth, YOLO's base prediction, and the conformal prediction set. Figure 9.1 illustrates a scenario where the model exhibits high confidence in its traffic sign classification, resulting in the prediction set containing only a single label. Conversely, Figure 9.2 demonstrates a case where, despite the base detection accurately identifying the "go straight" sign, the model lacks confidence in its classification. Consequently, the prediction set includes two labels. This uncertainty may arise from partial obstruction of the traffic sign by a tree branch.

It's important to note that the inclusion of "similar" classes in the prediction set is not a guaranteed outcome of conformal prediction. This occurs when the model ranks a "similar" class as the second, third, or subsequent most confident label. As seen in Figure 9.2, the second label in the prediction set is "go right," which, like "go straight," features a blue and white arrow. This suggests that the model has learned to associate the blue and white color

scheme with the "go right" sign, or it indicates that "go straight" and "go right" signs are closely related in the feature space defined by our model's weights.

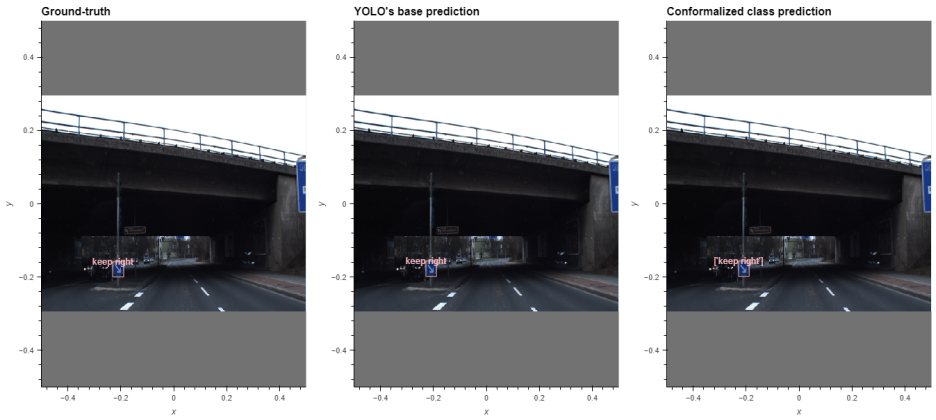

Fig. 9.1: High confidence prediction in traffic sign classification by the model, showcasing a single label in the prediction set, indicative of precise model certainty.

Now, Let discuss evaluation of the data presented in Table 9.1. The metric *Precision@0.7* represents the proportion of true positive predictions, where a true positive is characterized by a predicted bounding box with an IoU greater than 0.7 in relation to the actual bounding box, coupled with accurate classification. In a similar vein, *Recall@0.7* is defined with the same criteria for identifying true positives. The mAP (mean Average Precision) is a standard metric used to evaluate the overall precision of predictions across different classes and IoU thresholds.

The key metric of interest here is *boxCoverage@0.7*, which exclusively considers classification coverage, independent of other aspects like object presenceness. It counts only those classifications where the predicted bounding boxes have a high IoU with the ground-truth boxes. Unlike other metrics, it does not penalize for missing objects. For instance, if the model correcly predict the presenceness of just one out of 200 traffic signs and correctly classify the traffic sign, it would still achieve a perfect box-coverage score. Therefore, to gain a comprehensive understanding of the model's performance, this score should be interpreted in conjunction with metrics like mean Average Precision (mAP) and recall, which provide a broader evaluation of the model's detection and classification capabilities.

Moreover, the coverage of standard APS conformal prediction is guarantee on average, *marginal coverage*, not *conditional coverage*. As shown in table 9.1, setting $\alpha_c = 0.10$ resulting in average coverage of 0.906 but some classes have particularly high or low coverage. We can extend close to conditional coverage given a larger dataset with alternative.

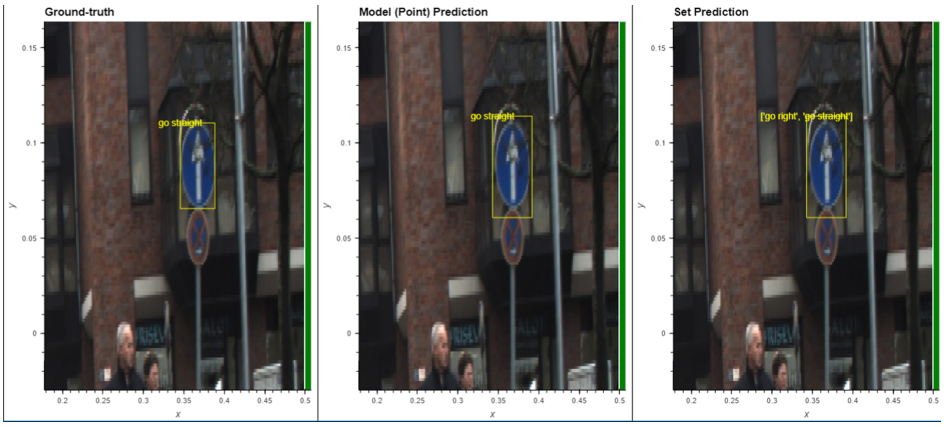

Fig. 9.2: An instance of low confidence prediction where the model includes multiple labels in the prediction set. Here, the "go straight" sign is identified, but due to potential obstructions like a tree branch, the model also suggests a "go right" sign, reflecting uncertainty in classification.

Class	Precision@0.7	Recall@0.7	mAP50	mAP50-95	boxCoverage@0.7
across all classes	0.786	0.756	0.837	0.613	0.906
no traffic both ways	0.578	0.5	0.496	0.298	0.750
go right	0.0273	0.0273	0.295	0.155	0.667
go left	0.864	0.5	0.548	0.249	0.750
go straight	0.417	0.25	0.557	0.409	0.800
no overtaking	0.851	1.0	0.995	0.722	0.962
slippery road	0.835	0.75	0.804	0.519	1.000000
cycles crossing	1.0	0.0	0.275	0.217	0.500000
speed limit 30	0.672	0.79	0.884	0.657	0.947368
priority at next intersection	0.863	0.909	0.948	0.63	0.928
restriction ends	0.841	1.0	0.995	0.796	0.667

Table 9.1: Evaluation of the YOLOv5m model on test set on average and the classes that our model struggle with, detailing precision, recall, mean Average Precision at various IoU thresholds, and bounding box coverage with high IoU greater than 0.7.

Next, let see figure 9.1.1 if conformal risk control can control the risk of missing pixels. Short answer, yes. But if we take a look at $\hat{\lambda}$ it is obvious that it become too conservative with respect to FNR when we set desired risk α to be too low. This could be due to the fact that percentage of pixels missed is not the best risk to model confidence threshold.

Code 9.1.1: Is the false negative risk controlled?

```
fnr_train = false_negative_risk(training_set, lambda_hat, IOU_NMS_THRESH)
fnr_cal = false_negative_risk(calibration_set, lambda_hat, IOU_NMS_THRESH)
fnr_test = false_negative_risk(testing_set, lambda_hat, IOU_NMS_THRESH)

# --- OUTPUTS/RESULTS. False negative rate for each alpha.
# When setting alpha = 0.1
# lambda_hat = 0.9994728565522926766462319392303470522165298461914062 5
Percentage of missing pixels in training set   :  8.071979469907644 %
Percentage of missing pixels in calibration set:  9.109235422380388 %
Percentage of missing pixels in test set       :  7.481371098392897 %

# When setting alpha = 0.4
# lambda_hat = 0.1188964545717878795727528995485045015811920166015625
Percentage of missing pixels in training set   :  33.02996989588737 %
Percentage of missing pixels in calibration set:  39.608337512005434 %
Percentage of missing pixels in test set       :  34.655756306004 %

# When setting alpha = 0.8
# lambda_hat = 0.044189482926316192568094720627414062619209289550781 25
Percentage of missing pixels in training set   :  84.16984125397148 %
Percentage of missing pixels in calibration set:  80.16423989621832 %
Percentage of missing pixels in test set       :  84.21822315150051 %
```

Remember that we are just determining confidence threshold, thus lowering α_{fnr} will increase false positive rate as shown in figure 9.3. More sophisticated measure of risk could be used to improve the performance, similarly to how selecting a better nonconformity measure affect prediction width in conformal prediction.

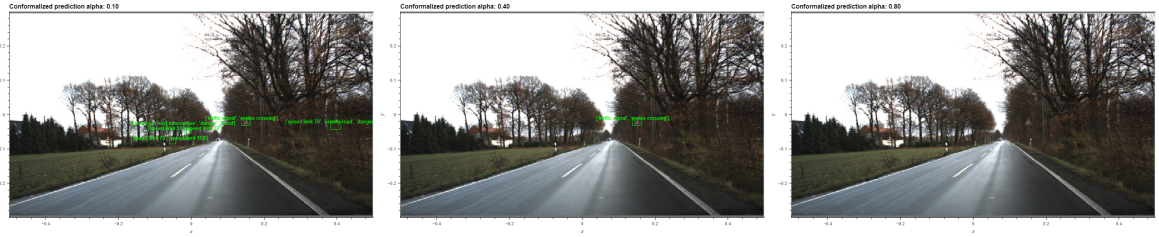

Fig. 9.3: Comparison of model predictions at varying α_{fnr} levels. The sequence from left to right illustrates increasing values: the leftmost figure at α_{fnr}=0.10, the central figure at α_{fnr}=0.40, and the rightmost figure at α_{fnr}=0.80, demonstrating the model's response to different risk thresholds.

One satifying note is that in figure 9.4 the case where there are false positive prediction, prediction set is always larger than 1, indicating that the model is uncertain on its classification.

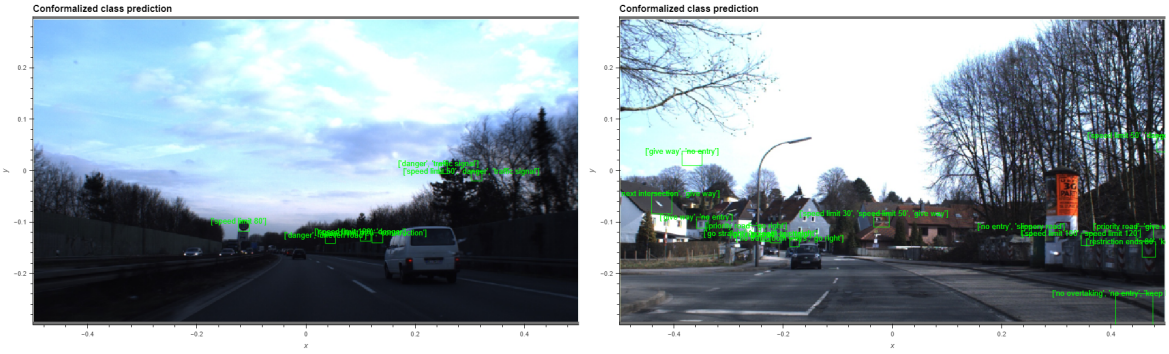

Fig. 9.4: Uncertain classification in false positive boxes.

9.1.1 Detection and conformalized detection on Thai Dataset

We trained YOLOv5 on our custom Thai traffic sign dataset and assessed its performance on the test set. The model achieved an mAP@50 of 0.54 and an mAP@50-95 of 0.352. Subsequently, we evaluated the model with conformalized detections.

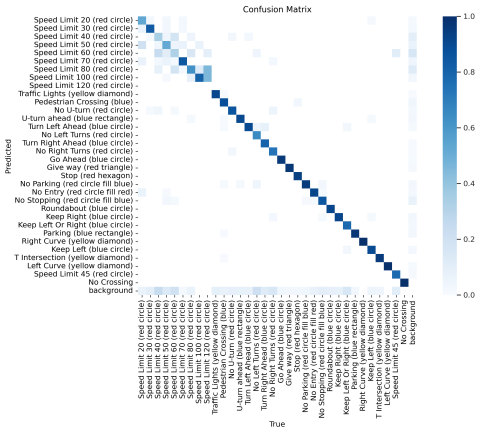

Fig. 9.5: Confusion matrix of YOLOv5 detector trained on our Thai traffic signs dataset.

Figure 9.5 presents the confusion matrix for the YOLOv5 detector after training on our Thai traffic signs dataset. The matrix reveals a tendency of the model to confuse between different speed limit signs, indicating areas where the model's classification performance can be improved. Additionally, it appears that the model occasionally misidentifies background elements as traffic signs. A closer examination reveals that many of these supposed "background" misclassifications are actually traffic signs not represented in our dataset's classes. This insight suggests an opportunity to enhance the model's accuracy and generalization by expanding our dataset to include these overlooked sign categories.

Figure 9.6 show that when setting $\alpha_c = 0.10$, classification sets become too conservative, with average prediction width in test set is of size 12. This could mean two things: 1) Classification aspect of the model is not great 2) Calibration dataset is too small.

Furthur investigation, in figure 9.6, we can see that $\hat{q}_{0.10}$ is quite far away from $\hat{q}_{0.20}$ but only have few samples between them. This could be caused by *noises* in a small calibration

Fig. 9.6: Illustration of conformal prediction in the Thai dataset with $\alpha_c = 0.10$. Class labels are presented numerically.

dataset. If we drawn more data from the same distribution, maybe $\hat{q}_{0.10}$ will get shift to the left resulting in smaller prediction set.

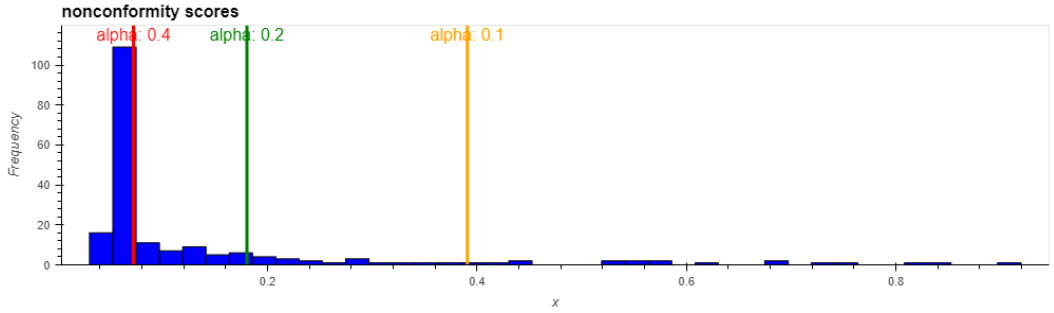

Fig. 9.7: Distribution of nonconformity scores using the APS method on the calibration set, displayed alongside quantiles for different α_c values (0.10, 0.40, 0.80). The graph highlights the distinct gap between α_c values of 0.10 and 0.20 by only few samples.

When we try APS method with $\alpha_c = 0.20$, figure 9.8, we still get a large prediction set, average width of 5.5. This is where regularization come in, we apply RAPS method with $k_{reg} = 5$ and $\lambda_{reg} = 0.01$ and result in average prediction width of 1.8512820512820514 and coverage of 0.8205128205128205, having rand=True. This result is not most satifying; prediction set is still too large, but this is what we currently get for our dataset. We will work use RAPS method with $\alpha_c = 0.20$ in this section of Thai dataset.

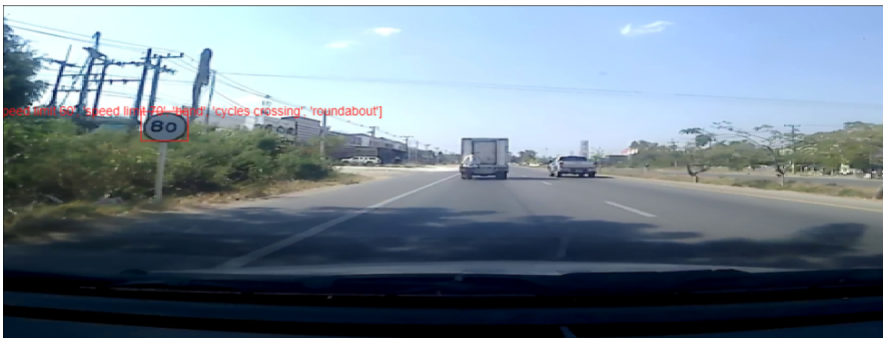

Fig. 9.8: APS method with $\alpha_c = 0.20$ still too conservative in prediction set.

Plots with RAPS method in figure 9.9 and 9.10. We get much smaller prediction set on average, but still too large to be informative in practice.

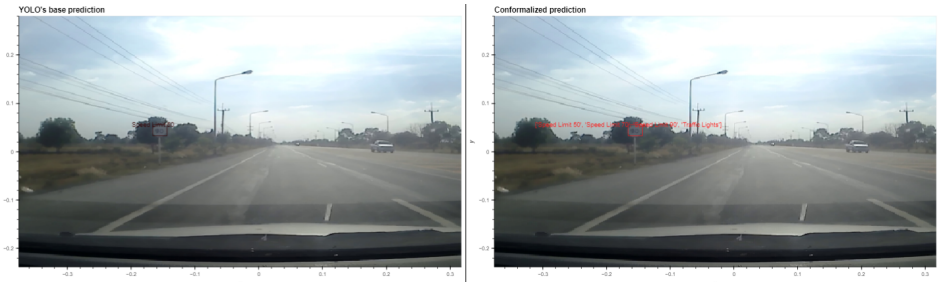

Fig. 9.9: Left image show the case where YOLO's point-prediction is incorrect and right image show where prediction set include the correct label.

Fig. 9.10: Left image show the case where YOLO's point-prediction is incorrect and right image show where prediction set does not include the correct label.

In Codeblock 9.1.2, the model consistently overlooks more pixels in the test set than in the calibration set. This observation suggests two possibilities: either the calibration set does not adequately represent the test set, or the training set shares more similarities with the calibration set than with the test set. Expanding the dataset with additional images could potentially resolve these issues. Nevertheless, a deeper exploration is necessary to identify the underlying cause of this discrepancy. **Realization Note:** The annotator of the test set's bounding boxes tended to draw larger boxes compared to those drawn by the annotator for the training and calibration sets. This bias is probably the cause of the discrepancy.

Code 9.1.2: FNR is not controlled in Thai dataset?

```
fnr_cal = false_negative_risk(calibration_set, lambda_hat, IOU_NMS_THRESH)
fnr_test = false_negative_risk(testing_set, lambda_hat, IOU_NMS_THRESH)

# --- OUTPUTS/RESULTS. False negative rate for each alpha.
# When setting alpha = 0.1
# lambda_hat: 0.99995496869273914342102216323837637901306152343 75
Percentage of missing pixels in calibration set:  9.501530366116 %
Percentage of missing pixels in test set       :  15.318187412704006 %
# When setting alpha = 0.4
# lambda_hat: 0.171142607926239032067883272247854620218276977539 0625
Percentage of missing pixels in calibration set:  39.78662338988958 %
Percentage of missing pixels in test set       :  55.589865522066226 %

# When setting alpha = 0.8
# lambda_hat: 0.062255829573261659004579371412546606734395027160644531 25
Percentage of missing pixels in calibration set:  79.64937933718265 %
Percentage of missing pixels in test set       :  87.22669016621852 %
```

10. Implementation of YOLO from Scratch

This chapter is an additional segment, primarily for my own reflection and not the main focus of this project/report. However, it was a valuable experience that I'd like to share. Most of the implementation discussed in this chapter can be found in JaxMao. I have developed a particular fondness for JAX [23], a high-performance numerical computing library known for its effectiveness in numerical computations.

JAX offers a plethora of tutorials for neural network construction, such as the JAX Advanced Tutorial: Training a Simple Neural Network, which provides a foundational guide for building neural networks. Initially focusing on sequential fully connected layers, this tutorial was the springboard for extending the network to include convolutional layers, batch normalization, and other elements crucial for implementing YOLOv1 and YOLOv3. The architecture and loss functions for these models were heavily inspired by Aladdin Persson's implementations (I recommend searching his YouTube channel for more insights), with just a few tweaks on our end.

Due to computational constraints, our model had to be scaled down, limiting it to a handful of convolution layers and restricting its performance to a small training dataset. Below are the plots of the model outputs on the training set. Figure 10.1 displays results from our YOLOv1 implementation trained on the Road Sign Dataset [24]. As seen, the left result is quite promising, albeit with overlapping bounding boxes for the same object, while the right result demonstrates its effectiveness on smaller, distant objects.

Figures 10.3 and 10.4 show outcomes from YOLOv3, trained on the COCO128 dataset [25]. Despite the limited training due to computational constraints, the results are commendable, considering the brief training duration and lack of hyperparameter optimization.

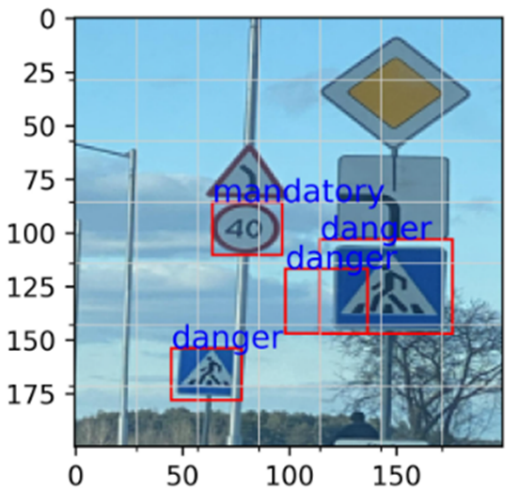

Fig. 10.1: Left: YOLOv1 result with overlapping bounding boxes.

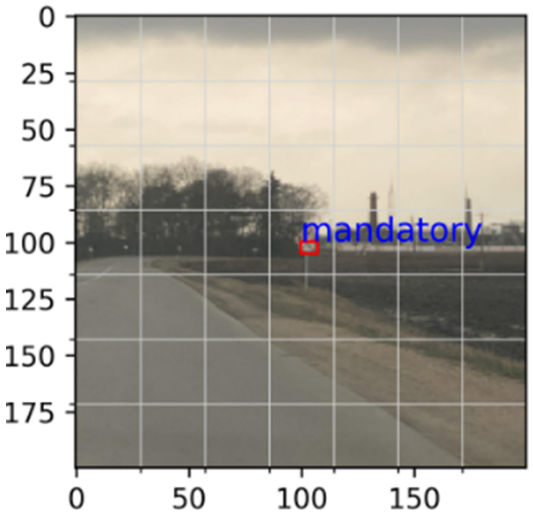

Fig. 10.2: Right: YOLOv1 detection of distant objects.

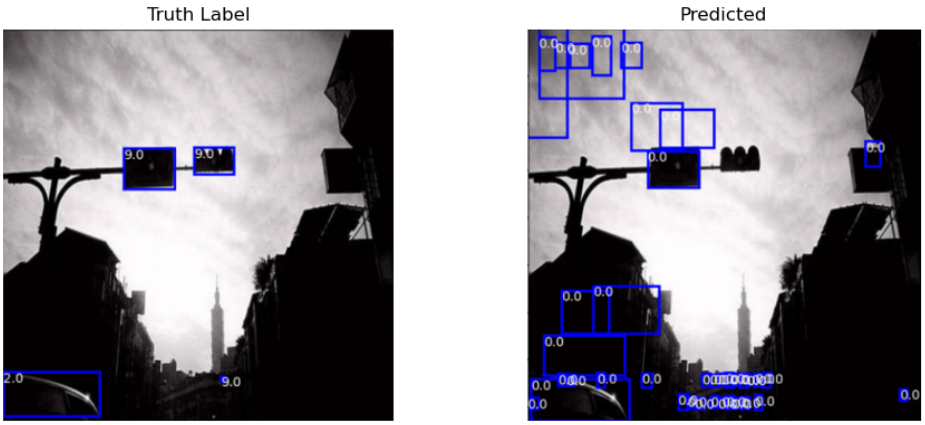

Fig. 10.3: YOLOv3 detecting a traffic light with some false positives.

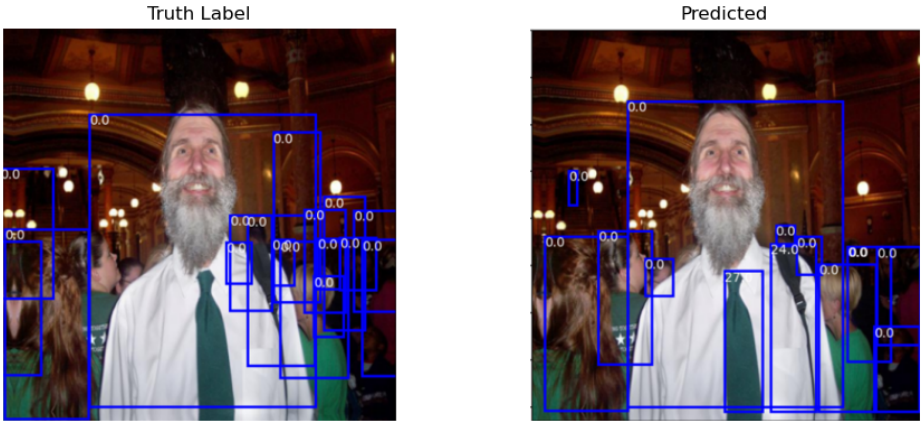

Fig. 10.4: YOLOv3 identifying larger objects (humans) with several false positives.

This chapter, although supplementary to the main project, showcases the challenges and learning from implementing YOLO models from scratch, illustrating the intricacies of deep learning model development.

11. Conclusion

Wrapping up, our project was an exciting adventure into the world of machine learning, with a special focus on detecting Thai Traffic Signs. We kicked off with putting together and polishing the Thai Traffic Signs dataset, which was key for getting our YOLOv5 model up and running. But it was more than just gathering data and training models; we really got our hands dirty with the nitty-gritty of deep learning, picking up tons of useful insights and skills along the way.

We tackled some pretty cool stuff like calibration, conformal prediction, and conformal risk control, getting a better grip on how to tweak deep learning models for more solid and reliable results. This was super important for making sure our model didn't just hit the mark on accuracy, but was also trustworthy – something you really want when you're bringing machine learning into the real world.

Throughout this whole ride, we didn't just focus on algorithms for detecting traffic signs; we also learned a ton about different aspects of machine learning and app building. This project really showed us how diverse and dynamic machine learning projects can be, mixing together technical know-how with creative thinking and designing for real people.

Looking ahead, we're pretty stoked about where we can take this project next. There's room to grow the dataset, try out different deep learning tricks, and pack more cool features into the app. This project was challenging, for sure, but also incredibly fulfilling. It's laid a great foundation for us to keep exploring and innovating in machine learning.

11.0.1 Future works

This section outlines potential enhancements and areas of exploration that we believe could significantly benefit the project but have yet to be undertaken. Our focus for future development encompasses three main domains: mobile application, conformal prediction, and conformal risk control methodologies. It's worth noting that there are additional opportunities for improvement not covered in this discussion

Mobile Application

- **Traffic Sign Detection via Vehicle-mounted Cameras:** While we've deployed our traffic sign detector on smartphones, an exciting direction would be its application through vehicle-mounted cameras for real-time detection and alerts.
- **Feedback-driven Detection Accuracy:** Implementing a feedback loop would allow

for the continuous refinement of detection accuracy by updating the model with new traffic sign information. Predicted bounding boxes could be reviewed and corrected as necessary, serving as additional data to further fine-tune the model. This process not only enhances model accuracy but also adapts the system to recognize new or previously underrepresented traffic signs more effectively.

- **Integration with Mapping Services API:** By integrating with mapping services APIs, the app can merge live traffic updates and road conditions with traffic sign data. This synergy could enable dynamic routing adjustments, enriching the navigational experience with comprehensive environmental insights.

Conformal prediction

- **Class-specific Conformal Prediction:** While our current implementation of conformal prediction ensures overall coverage, tailoring it to specific subcategories could approach conditional coverage more closely. Expanding the dataset and segmenting it into subcategories might refine coverage for particular groups.

- **Multilabel Classification:** Future research could explore the multilabel classification scenarios, where classes are not mutually exclusive. This extension would cater to more complex detection tasks, broadening the applicability of conformal risk control.

Conformal risk control

- **Subcategorization in CRC:** Applying CRC to subcategories can lead to more refined confidence assessments. Due to the discrepancies in model confidence for objects of varying sizes, data segmentation into subcategories promises to provide more specific conditional guarantees. For example, smaller-sized objects, which models frequently overlook due to typically lower confidence levels, could benefit significantly from this approach. Tailoring CRC to consider such subcategories would address these variations, ensuring more reliable detections across the board.

- **Expanding CRC Framework:** The approach we've taken with Conformal Risk Control (CRC) serves essentially as an introductory exploration. It's a straightforward method where we control the risk of missing pixels by using confidence threshold. However, there exists more complex and comprehensive applications of CRC. Future initiatives could aim to broaden the scope by incorporating a wider variety of control variables and risk functions. By delving deeper into the capabilities of CRC, we can enhance the model's predictive accuracy and reliability, especially in scenarios where precision is crucial.

- **Risk Function Refinement:** Our current approach to CRC focuses on the risk of missing pixels which may not be highly dependent on confidence threshold. Investigating alternative risk functions, such as the absence of high IoU bounding box matches, could provide more meaningful control mechanisms.

- **Learn then Test:** Learn then Test approach provide a way to control multiple risks but we have not tried it.

12. Bibliography

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[2] R. Girshick, *Fast r-cnn*, 2015. arXiv: 1504.08083 [cs.CV].

[3] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV].

[4] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, 2018. arXiv: 1708.02002 [cs.CV].

[5] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, *Detectron*, https://github.com/facebookresearch/detectron, 2018.

[6] G. Jocher, *Ultralytics yolov5*, version 7.0, 2020. DOI: 10.5281/zenodo.3908559. [Online]. Available: https://github.com/ultralytics/yolov5.

[7] W. Bai, J. Zhao, C. Dai, *et al.*, "Two novel models for traffic sign detection based on yolov5s," *Axioms*, vol. 12, no. 2, 2023, ISSN: 2075-1680. DOI: 10.3390/axioms12020160. [Online]. Available: https://www.mdpi.com/2075-1680/12/2/160.

[8] J. Shen, Z. Zhang, J. Luo, and X. Zhang, "Yolov5-ts: Detecting traffic signs in real-time," *Frontiers in Physics*, vol. 11, 2023, ISSN: 2296-424X. DOI: 10.3389/fphy.2023.1297828. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fphy.2023.1297828.

[9] P. Thipsanthia, R. Chamchong, and P. Songram, "Road sign detection and recognition of thai traffic based on yolov3," in Oct. 2019, pp. 271–279, ISBN: 978-3-030-33708-7. DOI: 10.1007/978-3-030-33709-4_25.

[10] J. Nixon, M. Dusenberry, G. Jerfel, *et al.*, *Measuring calibration in deep learning*, 2020. arXiv: 1904.01685 [cs.LG].

[11] L. Tao, Y. Zhu, H. Guo, M. Dong, and C. Xu, *A benchmark study on calibration*, 2023. arXiv: 2308.11838 [cs.LG].

[12] C. Bishop, "Bayesian neural networks," English, *Journal of the Brazilian Computer Society*, vol. 4, no. 1, pp. 61–68, Jul. 1997, ISSN: 0104-6500. DOI: 10.1590/S0104-65001997000200006.

[13] J. M. Hernández-Lobato and R. P. Adams, *Probabilistic backpropagation for scalable learning of bayesian neural networks*, 2015. arXiv: 1502.05336 [stat.ML].

[14] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, *Weight uncertainty in neural networks*, 2015. arXiv: 1505.05424 [stat.ML].

[15] S. H. Yelleni, D. Kumari, S. P.K., and K. M. C., "Monte carlo dropblock for modeling uncertainty in object detection," *Pattern Recognition*, vol. 146, p. 110 003, 2024, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2023.110003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S003132032300701X.

[16] V. Vovk, A. Gammerman, and G. Shafer, "Algorithmic learning in a random world," in Jan. 2005. DOI: 10.1007/b106715.

[17] G. Shafer and V. Vovk, *A tutorial on conformal prediction*, 2007. arXiv: 0706.3188 [cs.LG].

[18] A. N. Angelopoulos and S. Bates, *A gentle introduction to conformal prediction and distribution-free uncertainty quantification*, 2022. arXiv: 2107.07511 [cs.LG].

[19] A. Angelopoulos, S. Bates, J. Malik, and M. I. Jordan, *Uncertainty sets for image classifiers using conformal prediction*, 2022. arXiv: 2009.14193 [cs.CV].

[20] L. Andéol, T. Fel, F. D. Grancey, and L. Mossina, *Confident object detection via conformal prediction and conformal risk control: An application to railway signaling*, 2023. arXiv: 2304.06052 [cs.LG].

[21] A. N. Angelopoulos, S. Bates, A. Fisch, L. Lei, and T. Schuster, *Conformal risk control*, 2023. arXiv: 2208.02814 [stat.ME].

[22] A. N. Angelopoulos, S. Bates, E. J. Candès, M. I. Jordan, and L. Lei, "Learn then test: Calibrating predictive algorithms to achieve risk control," *CoRR*, vol. abs/2110.01052, 2021. arXiv: 2110.01052. [Online]. Available: https://arxiv.org/abs/2110.01052.

[23] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: http://github.com/google/jax.

[24] *Road signs dataset*. [Online]. Available: https://makeml.app/datasets/road-signs.

[25] T. Roboflow, *Coco 128 dataset*, https://universe.roboflow.com/team-roboflow/coco-128, Open Source Dataset, visited on 2024-02-22, Sep. 2021. [Online]. Available: https://universe.roboflow.com/team-roboflow/coco-128.