Data Science Workshop 2023

PERSONAL PROTECTIVE EQUIPMENT DETECTION USING YOLOX

June 28, 2023

Mateusz Wójcik, Elżbieta Jowik, Mateusz Kierznowski, Bartłomiej Eljasiak

1 Introduction

The aim of this project was an implementation of real-time detection of PPE (e.g., hard hat, safety vest) compliances of workers. The main goal was to develop and validate a model for detecting and classifying objects through image analysis, utilizing the YOLOX framework.

The effectiveness of the model depends on the quality of the data it is trained on. Hence, the project consisted of two primary stages: gathering training data and training a model capable of generating probability predictions for multiple classes of labels.

The project's scope was thoroughly discussed and agreed upon with the product owner. The remaining part of this section provides a brief summary of the established requirements.

Requirements for the model

1. Object detection using YOLOX framework;
2. Supported object classes: Person, Helmet, Head, sVest;
3. Emphasize on minimizing false positive detections, such as bald people or people wearing hats;
4. Compatibility with YOLOX framework;
5. Support for YOLOX "S" model.

Deliverables

1. Model formats: ONNX, native format for YOLOX (BIN and XML)
2. Training and validation datasets
3. Test results, including:
    - Recall (R), Precision (P), mean Average Precision (mAP) for each object class
    - Evaluation of algorithm performance

2 Existing methods

Nipun et al. proved the existence of multiple approaches that can be utilized for addressing our specific problem. In their study [3], they outlined three distinct methodologies that focus on verifying compliance with personal protective equipment (PPE) requirements.

Approach-1: YOLO-v3-A1 model detects worker, hat, and vest (three object classes) individually. Next, ML classifiers (Neural Network, Decision Tree) classify each worker as W (wearing no hat or vest), WH (wearing only hat), WV (wearing only vest), or WHV (wearing both hat and vest).

Approach-2: YOLO-v3-A2 model localizes workers in the input image and directly classifies each detected worker as W, WH, WV, or WHV.

Approach-3: YOLO-v3-A3 model first detects all workers in the input image and then, a CNN-based classifier model (VGG-16, ResNet-50, Xception) was applied to the cropped worker images to classify the detected worker as W, WH, WV, or WHV.

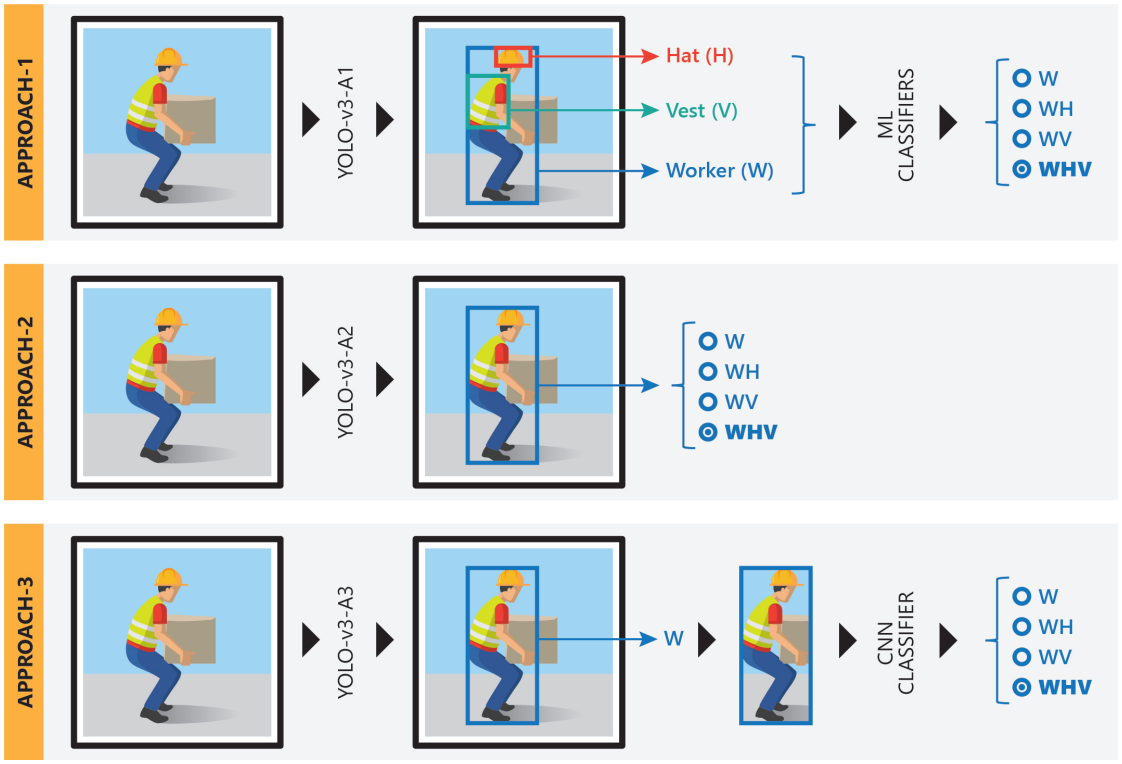

Figure 1: A visual representation highlighting the contrasting discussed approaches. Source: [3]

3 YOLOX

YOLOX [2] is a single-stage object detector. It is a solution for real-time object detection tasks that builds upon the popular You Only Look Once (YOLO) family of models. YOLOX extends YOLOv3 with a DarkNet53 backbone. Specifically, YOLO's head is replaced with a decoupled one. For each level of FPN feature, it adopts a 1×1 conv layer to reduce the feature channel to 256. Then it adds two parallel branches with two 3×3 conv layers each for classification and regression tasks, respectively.

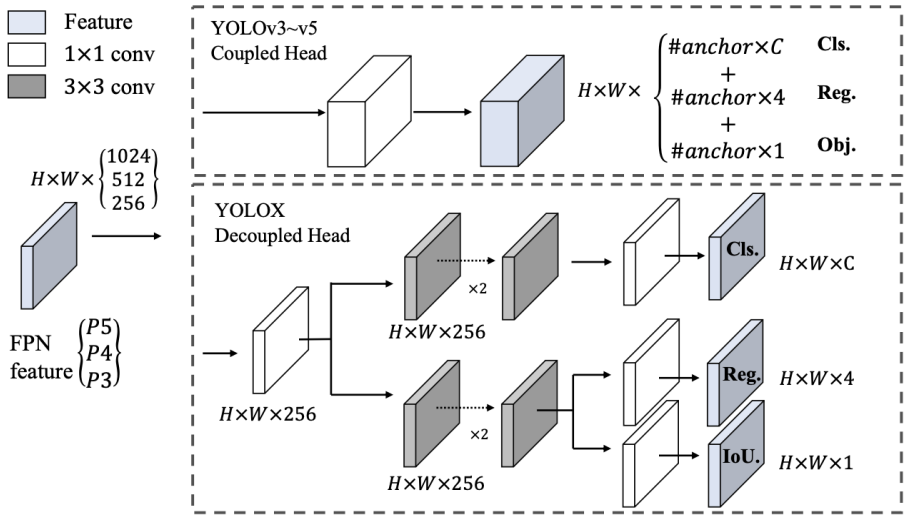

Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, it first adopts a 1×1 conv layer to reduce the feature channel to 256 and then adds two parallel branches with two 3×3 conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch. Source: [2].

4 Chosen datasets

The initial data source we utilized was The Hard Hat dataset [4], which was suggested by the product owner. This dataset focuses on object detection and specifically targets workers in workplace environments where wearing a hard hat is necessary. The annotations in the dataset encompass instances of both "person" and "head" categories, allowing for scenarios where individuals may be present without wearing a hard hat. Consequently, one potential application is to develop a classifier that can distinguish between workers who adhere to safety regulations in the workplace and those who do not.

Due to the small amount of data and poor coverage of classes, we decided to incorporate more datasets representing selected classes from a large number of protective personal equipments datasets. There are four different datasets under consideration:

- PPE-detect-v3 Computer Vision Project (available via link). This dataset contains 1031 images with multiple labels (blue-helmet, glove, no helmet, no vest, orange helmet, person, red-helmet, safety boot, vest, white-helmet, yellow-helmet).

- Construction Site Safety Computer Vision Project (available via link). This dataset offers 717 images with over 20 labels.

- Personal Protective Equipment - Combined Model Computer Vision Project (available via link). The largest dataset from selected containing 44002 images with labels (Fall-Detected, Gloves, Goggles, Hardhat, Ladder, Mask, NO-Gloves, NO-Goggles, NO-Hardhat, NO-Mask, NO-Safety Vest, Person, Safety Cone, Safety Vest).

- ppe8 Computer Vision Project (available via link). The smallest of the considered dataset. However, it covers all necessary labels.

Note that we do not utilize every available image because some images contain only labels that were not selected for the detection task. Thus, the dataset is depicted in 39800 images and divided into train and validation subsets in an 80:20 ratio. Label occurrences are shown in Table 1.

Label	Number of occurences
No helmet	21790
Helmet	90377
Person	41042
Vest	29421

Table 1: Descritpion of initial dataset

5 Encountered issues

The above table provide some inadequacy. The sum of labelled helmets and No helmet does not equal the number of labelled people. This is probably because not all people were correctly labelled among the selected datasets.

While creating a common dataset, we encountered the problem of not fully covering classes. Some datasets did not have a common subset of classes (helmet, no_helmet, vest, people) which resulted in a situation where the model, despite the real presence of some class in the image, did not receive the corresponding label. As a result, the YOLOX architecture received conflicting information from the combined dataset, which consequently translated into very low scores of the original models.

To overcome this problem, we have incorporated some additional tools to provide missing labels. Firstly, we used a base pre-trained YOLOX-x model to produce *person* annotations. The additional bounding boxes for both the training and evaluation datasets proved to significantly improve the metrics of the model. Additionally, people started to be detected in most cases and scenes.

After training the second set of models with new annotations, we noticed that, in

cases where there is no helmet, there is no information about people's heads. After some research, we decided to further improve the dataset by producing new annotations given the pre-trained model published in the GitHub repository by Vincenzo Varriale[6]. The model was trained on the WIDER FACE dataset – a face detection benchmark introduced in [5]. Although the bounding boxes were mostly focused on faces instead of heads, this approach improved the metrics of models as well.

Finally, while analysing the predictions of the model, we noticed that safety vests were rarely detected in some scenarios. To improve the performance of the model in this class, we followed a similar approach. We found a pre-trained model published in [1] that detected even more PPE classes and used it to produce predictions for a safety vest class. This process provided us with the final dataset.

An example of adding new labels to a dataset is shown in figures 3 and 4.

Figure 3: Vanilla images

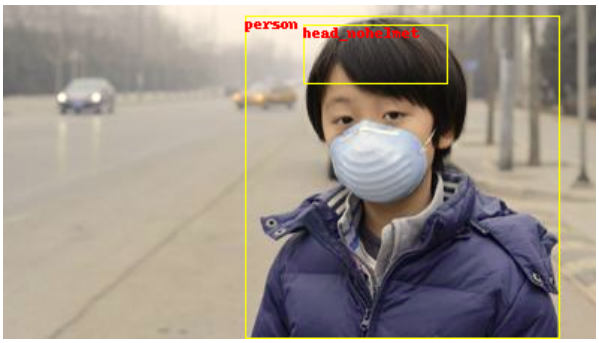

Figure 4: Image with *person* label added

After a detailed investigation, we enhanced the dataset by doubling the number of labels. Table 2 represents the final dataset with additional labels. The difference between the sum of the helmet and no helmet against person class is still evident. However, we were able to reduce the problem to a size that does not adversely affect the developed models.

Label	Number of occurences
No helmet	57754
Helmet	90377
Person	132548
Vest	41161

Table 2: Descritpion of final dataset

6 Evaluation

Before describing the metrics used during training, we first introduce some basic concepts on detection metrics. Intersection over Union, known as Jaccard Index, is used to measure the overlap between the ground truth and the predicted bounding box. It is defined as

$$\text{IoU} = \frac{|G \cap P|}{|G \cup P|},$$

where G is the ground-truth bounding box, and P is the predicted bounding box. Typically, an IoU higher than 0.5 is considered as a good prediction. In our analysis, we considered values between 0.25 and 0.95 as the existence of a correct detection is more important than the precise localisation in our task.

The IoU metric is used to classify predictions as True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). Based on these values, both Precision and Recall metrics are evaluated.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Next, the Precision-Recall pairs are plotted for each class based on the confidence threshold of predictions. The Average Precision (AP) is calculated as the area under the Precision-Recall curve. Then, the mean Average Precision (mAP) is evaluated by taking the mean over AP over all classes and/or considering IoU thresholds often given in the name of the metric. Similarly, the mean Average Recall can be calculated.

7 Experiments

7.1 Initial experiments

One of the challenges of the project was navigating through a vast space of possible model architectures, configurations and hyperparameters. `YOLOx` library, which was a starting point for our research, had publicised several pre-trained models of similar yet different architectures. They differed mainly in the number of parameters, but this would also impact the performance of the inference (see Table 3).

Model	$\textbf{mAPval}_{0.5:0.95}$	$\textbf{mAPtest}_{0.5:0.95}$	Speed V100 (ms)	Params (M)
YOLOX-s	40.5	40.5	9.8	9.0
YOLOX-m	46.9	47.2	12.3	25.3
YOLOX-l	49.7	50.1	14.5	54.2
YOLOX-x	51.1	51.5	17.3	99.1
YOLOX-D53	47.7	48.0	11.1	63.7

Table 3: Pretrained model architectures available in `YOLOx` library

Whereas almost all architectures were tested during the project, due to hardware limita-

tions, it was decided that experiments were to be performed on the smaller architectures initially. After the results of the initial work were gathered, a gradual shift towards larger models was made. In the end, to our surprise, the **YOLOX-m** model outperformed both **YOLOX-l** and **YOLOX-x** in our final solution.

During initial experiments observed performance did not reach our expectations. After taking a closer look at the individual parts of a ML pipeline the major cause of lackluster results was identified as missing annotations in the dataset. This was described in detail in Section 5 and the impact of the improvements can be observed in 5 when we compare the two very similar runs which are represented as green and brown lines spanning to above 60 epochs.

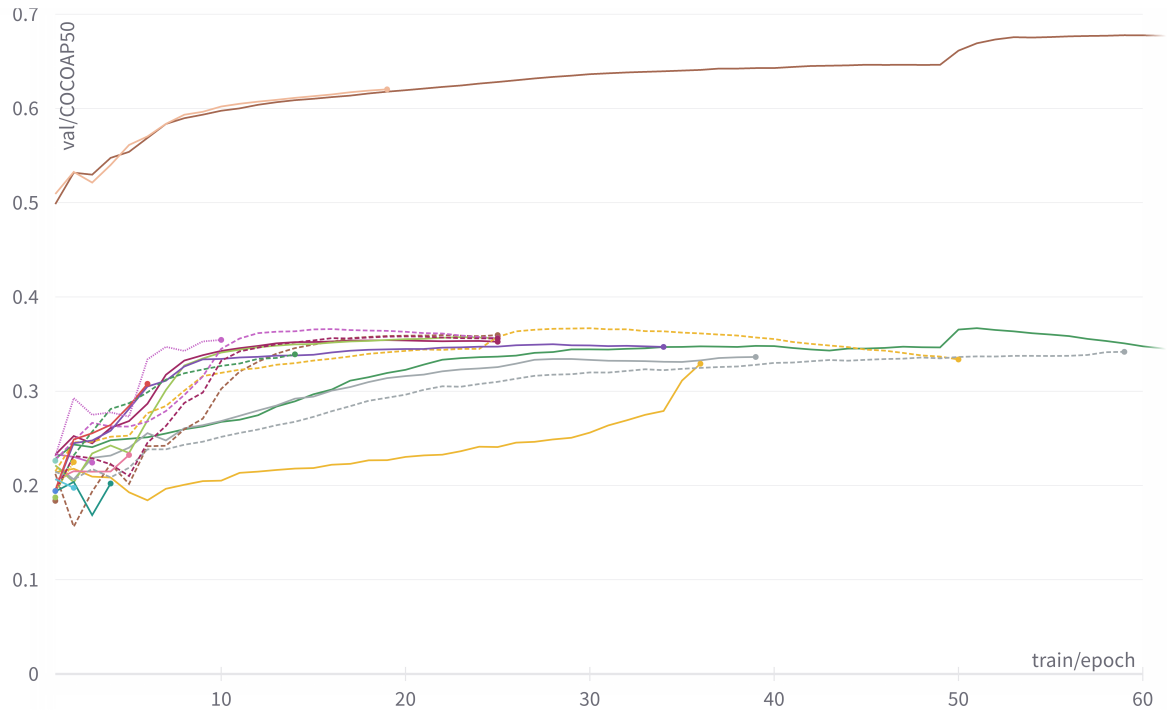

Figure 5: Results of multiple initial experiments (measured in mAP50)

7.2 Augmentation

One of the project's assumptions was that the created model would later be deployed in our product owner ecosystem. This was one of the reasons why model resilience and the

ability to adapt to new data were of the most importance. This naturally implies that numerous steps were taken to ensure that models were not overfitting on training data. Alongside L1 loss for models weights this also included incorporating data augmentation techniques, which also significantly improved models' performance in a few final epochs. This can be clearly observed in Figures 5 and 7. The impact of augmentation can be easily noticed, but surprisingly, there was no significant difference between the validation scores when augmentation of an increasing level was applied. It seemed that model was not influenced by the strength of the augmentation techniques, even though, in one case, images were rotated just by a few degrees and zoomed in by one-tenth and, in another case, fully flipped and zoomed in up twice. This behaviour could be explained by the high initial resilience of the pre-trained model or a mistake in the exploration of augmentation impact. This was not researched further as the current augmentation brought the desirable and satisfactory improvement to the model.

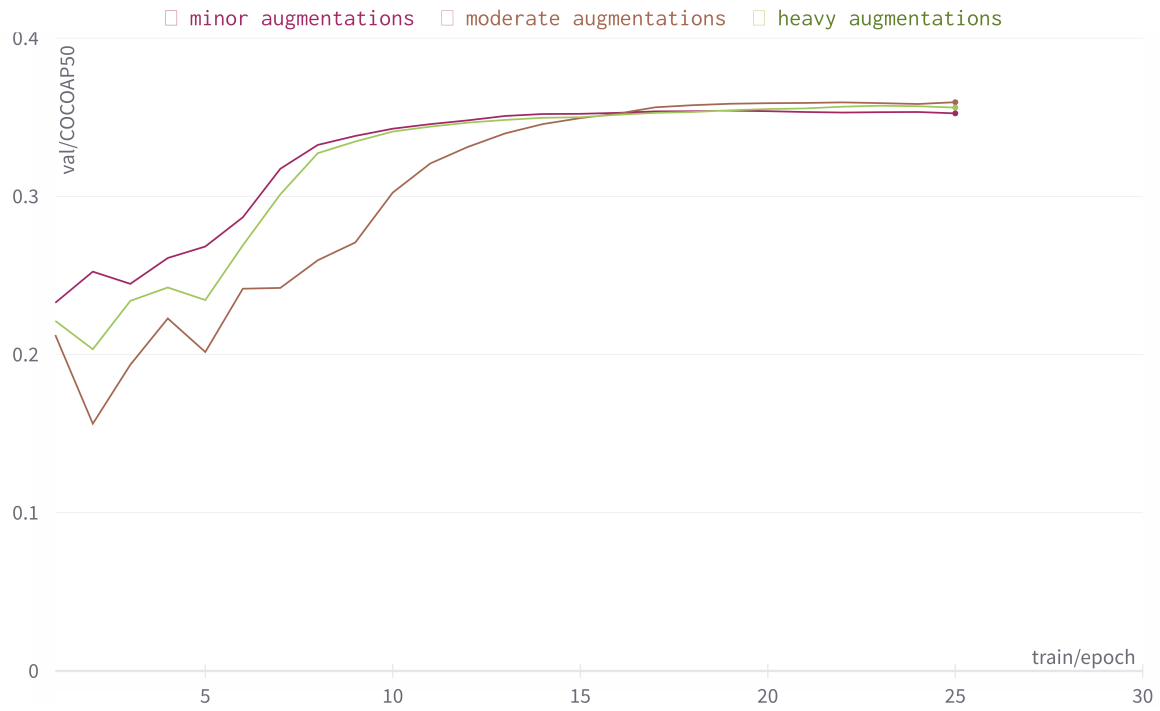

Figure 6: Testing different augmentations

7.3 Final models

After the initial tests a few observations have been made and one significant flaw was spotted. Our first iteration for dataset did lack annotations, especially for classes person, helmet. Final form of dataset, although still not perfect, significantly improved results, this was mentioned in Sections 5 and 7.1. To our surprise the model of medium size, achieved the highest scores on validation dataset. Comparison to 'large' model can be seen in 7. When it comes to hyperparameters, initial experiments found that SDGOptimizer with exponential decay for learning rate is the best choice. With batch size, we were limited by hardware to 16 for medium model, and our initial (maximal) learning rate was 0.01 per batch, this means that per image it was divided by the batch size. Decay decreased the initial batch learning rate down to 0.0005.

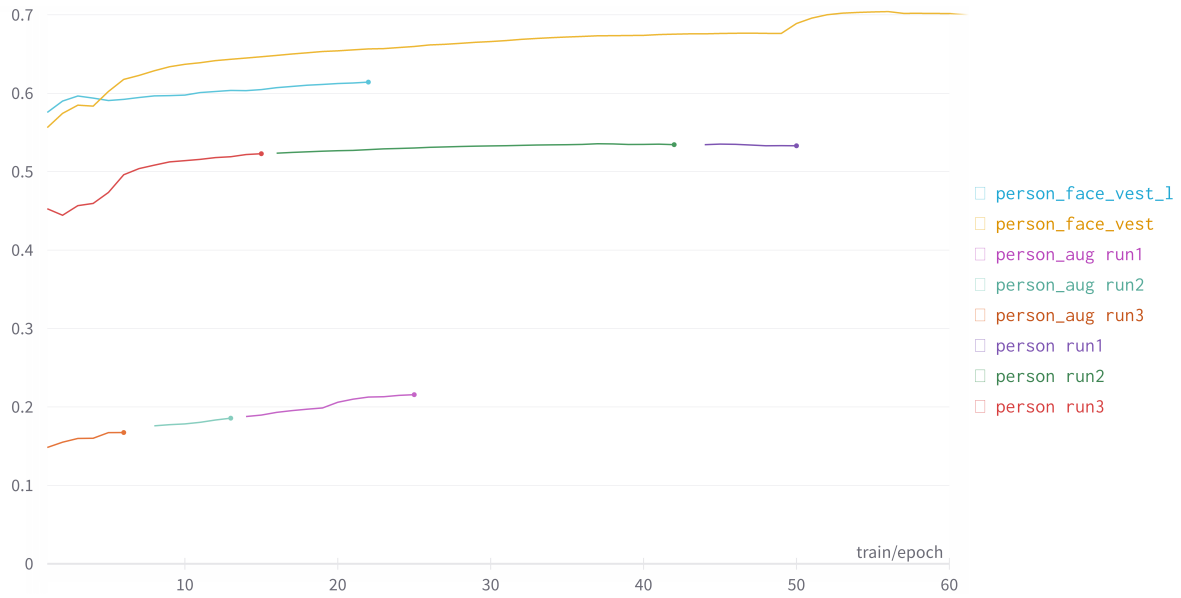

Figure 7: Final model experiments. Y axis represents AP50 measure calculated over validation dataset.

8 Results

The results will be split into four different scenarios based on custom annotations added to the dataset.

1. dataset without any custom annotations
2. *person* class annotations added using base pre-trained YOLOx model
3. *person* and *head_nohelmet* classes added to the dataset
4. *person*, *head_nohelmet*, and *vest* classes added to the dataset

The final validation metrics calculated during training are presented in Table 4. The metric AP50:95:0.5 is averaged over different IoU thresholds from 0.5 to 0.95 with the step of 0.005.

Scenario	AP50	AP50:95:0.5
1	0.3669	0.2295
2	0.5356	0.3456
3	0.6777	0.4437
4	0.7042	0.4621

Table 4: Validation metrics evaluated during training process inspired by Microsoft COCO evaluation

As already mentioned, the second metric is quite strict and focused on the precise localisation of the objects. There is a significant improvement in the values of the metrics between scenarios despite generating more annotations and making detection a more difficult task. After downloading model weights from our logging framework, `wandb.ai`,

for all 4 scenarios, we further evaluated the model and analysed both precision and recall for different IoU thresholds and classes. The results of other metrics are given in Table 5.

Metric	Scenario 1	Scenario 2	Scenario 3	Scenario 4
AP25:75	0.441	0.588	**0.651**	0.643
AP25	0.515	0.656	**0.726**	0.707
AP50	0.465	0.610	**0.683**	0.670
AP75	0.286	0.448	0.474	**0.490**
AR25:75	0.543	0.660	0.783	**0.798**
AR25	0.635	0.749	0.868	**0.879**
AR50	0.561	0.678	0.812	**0.825**
AR75	0.387	0.514	0.601	**0.629**

Table 5: Chosen metrics for all four scenarios. Average Precision (AP) and Average Recall (AR) were calculated on different IoU thresholds, where, for example, AP75 means Average Precision for 0.75 IoU threshold.

It can be noted that the model in the fourth scenario appeared to be more sensitive but less precise for lower values of IoU thresholds. We also analysed the mAP and mAR metrics for different sizes of annotations in images. The results are shown in Table 6. We noticed that there is a significant difference in both precision and recall between the small classes (mostly *helmet* and *nohelmet*) classes versus medium and large classes (*person* and *vest*) in favour of the latter classes.

Metric	Scenario 1	Scenario 2	Scenario 3	Scenario 4
mAP25:75:small	0.377	0.410	0.483	**0.522**
mAP25:75:medium	0.523	0.653	**0.693**	0.682
mAP25:75:large	0.514	0.695	**0.744**	0.721
mAR25:75:small	0.452	0.487	0.643	**0.662**
mAR25:75:medium	0.619	0.728	0.811	**0.828**
mAR25:75:large	0.642	0.788	0.887	**0.898**

Table 6: Mean Average Precision and mean Average Recall for different sizes of bounding boxes provided as an evaluation in the YOLOx framework.

The final analysis is mostly focused on Average Precision and Average Recall per class. The resulting evaluation is presented in Table 7 and Table 8.

Class	Scenario 1	Scenario 2	Scenario 3	Scenario 4
no helmet	0.2780	0.2791	0.6008	**0.6814**
helmet	0.6162	0.6298	0.6087	**0.6338**
person	0.2760	0.8572	0.8382	**0.8649**
vest	**0.5937**	0.5880	0.5556	0.3915

Table 7: Average Precision (AP) per class for all scenarios

Class	Scenario 1	Scenario 2	Scenario 3	Scenario 4
no helmet	0.3251	0.3221	0.8495	**0.8699**
helmet	0.7121	0.7180	0.7061	**0.7192**
person	0.4268	0.8972	0.8858	**0.8979**
vest	**0.7070**	0.7043	0.6916	0.7057

Table 8: Average Recall (AR) per class for all scenarios

Although the average precision for *vest* class decreased in the fourth scenario because of the increased number of annotations, there was a significant qualitative improvement noticed after model inference. More safety vests appeared in the predictions and were correctly detected. For other classes, the improvement in metrics after adding annotations of a corresponding class.

Some example predictions from all scenarios are presented in the Appendix section.

9 Summary and Future Improvements

To summarise, we believe that the obtained models are able to detect personal protective equipment in an acceptable way. The predictions' performance depends on the particular class and the number of example annotations provided in training data. Additionally, the fine-tuning process of YOLOx models is possible for large enough datasets, and their authors provide a great framework with useful tools for training, evaluation, and real-time inference. However, the framework appears to be sometimes tricky to modify.

We have also proven that generating new annotations using pre-trained models may be a useful tool to improve the dataset and the performance of models. Also, the dataset lacks negative cases that may confuse the model during training. Nonetheless, we see great

room for improvement in the annotation generation process by correcting the annotations and removing duplicates from them.

The main problem with the final model is that it sometimes duplicates predictions of *helmet* and *head* classes due to the fact that the added annotations were mostly focused on faces instead of heads, and removing duplicates in the process was difficult due to the lack of intersection between the bounding boxes. The improvement would require some more manual work, which is still less costly than manual annotation. We also believe that the *vest* class is predicted better than the metrics may suggest. Some hyperparameter tuning and augmentations may resolve the mentioned issues together with some manual annotation corrections.

Finally, YOLOx provides really fast inference (≈ 40 FPS on GPU), which may enable using models in real-time for production.

10 Appendices

Appendix 1.

Reference to repository:

Appendix 2.

Setting up the project

This section goes through setting up environment for training models, this requires some distribution of Linux OS, fortunately it can be also achieved with the `WSL` in Windows. First step is to clone our repository into preferred location. This repository consists of two projects `2023L-data-science-workshop/` and also **2023L-data-science-workshop/libs/YOLOX-data-science-workshop** which is a fork from **YOLOx** repository with our modification, it will serve as a environment for training models.

Setting up python environment

During project we used primarily Python 3.9, although versions up to 3.11 should also work. At this step it is assumed that the OS that is being used has some acceptable version of Python3 installed. To set up environment for trainig models next lines have to be executed, with the most important one being the last as it install the modified YOLOx library.

```
python -v venv env
source ./env/bin/activate
pip install -r requirements.txt
pip install -e .
```

Keep in mind that **PyTorch** (which is the deep learning library used underneath) requires special drivers being installed on machine being used. After performing the

above steps we heavily encourage anyone to test if the environment is set up properly by executing following two lines.

```
import torch
torch.cuda.is_available()
```

The expected output is `True`. If the returned value is False, then appropriate missing drivers have to be installed. Alternatively one can used container prepared by NVIDIA, available here.

Preparing dataset

After that we have to make sure that we have data placed in the right location, it being sub directory of **2023L-data-science-workshop/libs/YOLOX-data-science-workshop/datasets/** in our case it was sub directory **dsw**, this will be important in setting up experiments for model training. The structure of the dataset folder should be as following.

```
datasets/  # All datasets should be placed here
    dataset_name/
        annotations/
            - train_annotations.coco.json
            - val_annotations.coco.json
        train/   # Training images
        val/     # Validation images
```

Preparing training instructions

There is one more element that has to be prepared for the training and it is the experiment file. To set the parameters and hyperparameters of pipeline used in training YOLOx expects a .py file with class **Exp** that inherits from **BaseExp**. We prepared numerous such files, all of which can be found in **YOLOX-data-science-workshop/exps/dsw/**.

The exp file controls various parameters e.g. number of epochs, maximal learning rate, the use and strength of augmentations. Four parameters are crucial and their proper set up is necessary to ensure the successfully initialization of training, they are:

- `depth` and `width` - those parameters have to be changed depending on the size of the models being train, more details can be found here
- `data_dir` - path to dataset used, in our case `"datasets/dsw"`
- `train_ann` - name of the file inside **dataset_name/annotations/** containing training annotations, in our case `"train_annotations.coco.json"`
- `val_ann` - name of the file inside **dataset_name/annotations/** containing validation annotations, in our case `"val_annotations.coco.json"`

Training

If the repository, dataset and environment set up was done according to our guide, the following command is able to start the training loop. Training is initialized by the use of following command.

```
python tools/train.py
    -f exps/dsw/yolox_m_dsw.py \            # chosen experiment
    -d 1 \                                  # number of GPUs available
    -b 16 \                                 # batch size
    --fp16 \                                # float precision
    -o \                                    # occupy GPU for training
    -c ../../models/yolox_m.pth \           # path to chosen model
    --logger wandb \           # (Optional) Weights & Biases configuration
             wandb-project test-dsw \       # project name
             wandb-entity 20231-dsw-orange \
             wandb-name test_run \          # adjust the run name
             wandb-save_dir logs \
```

```
wandb-num_eval_images 5 \
wandb-log_checkpoints True
```

If all goes according to expectations, after a lot of environment initial information being printed out in console, one should also observe training logs similar to those in Figure 8, they will additionally also start appearing in logger if it was specified.

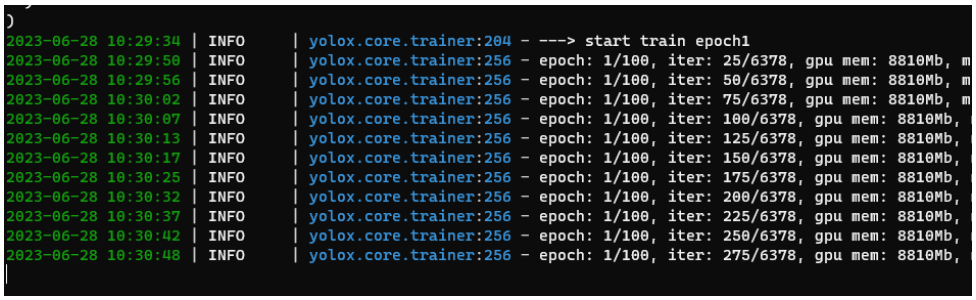

Figure 8: Example of model training logs

Appendix 3.

In this section, we present some example inferences obtained during each scenario and stage of the project. The examples were selected manually to show both the advantages and vulnerabilities of the models.

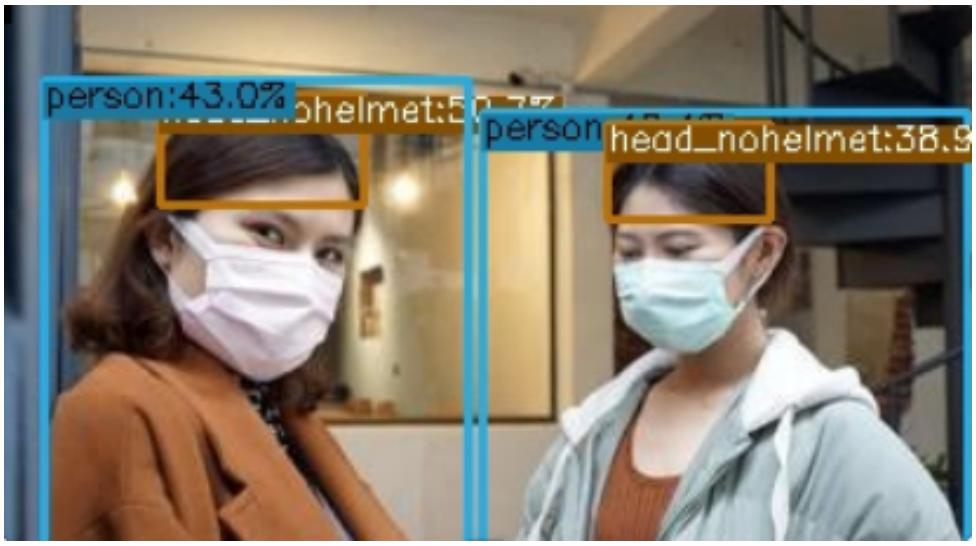

Figure 9: Scenario 1: Initially, the head annotations were only present in the domain of people with masks. The model appeared to correctly detect heads without helmets for such cases.

Figure 10: Scenario 1: In the construction site domain, the initial dataset lacked vest and person annotations. Only the helmet class was correctly detected in most of such cases.

Figure 11: Scenario 1: In some cases, vests were detected correctly. However, the model did not work well for all scenarios.

Figure 12: Scenario 2: After adding person classes generated by other YOLOx models, there was a significant improvement in the correct detection of people with high confidence. The predictions lacked head annotations without helmets.

Figure 13: Scenario 2: Also, the vest class was not detected in most of the cases.

Figure 14: Scenario 3: The main problem with the second set of added annotations was that it was focused mostly on faces, and sometimes both helmet and head class was detected for the same person. Nonetheless, some simple preprocessing techniques may improve the result of such predictions.

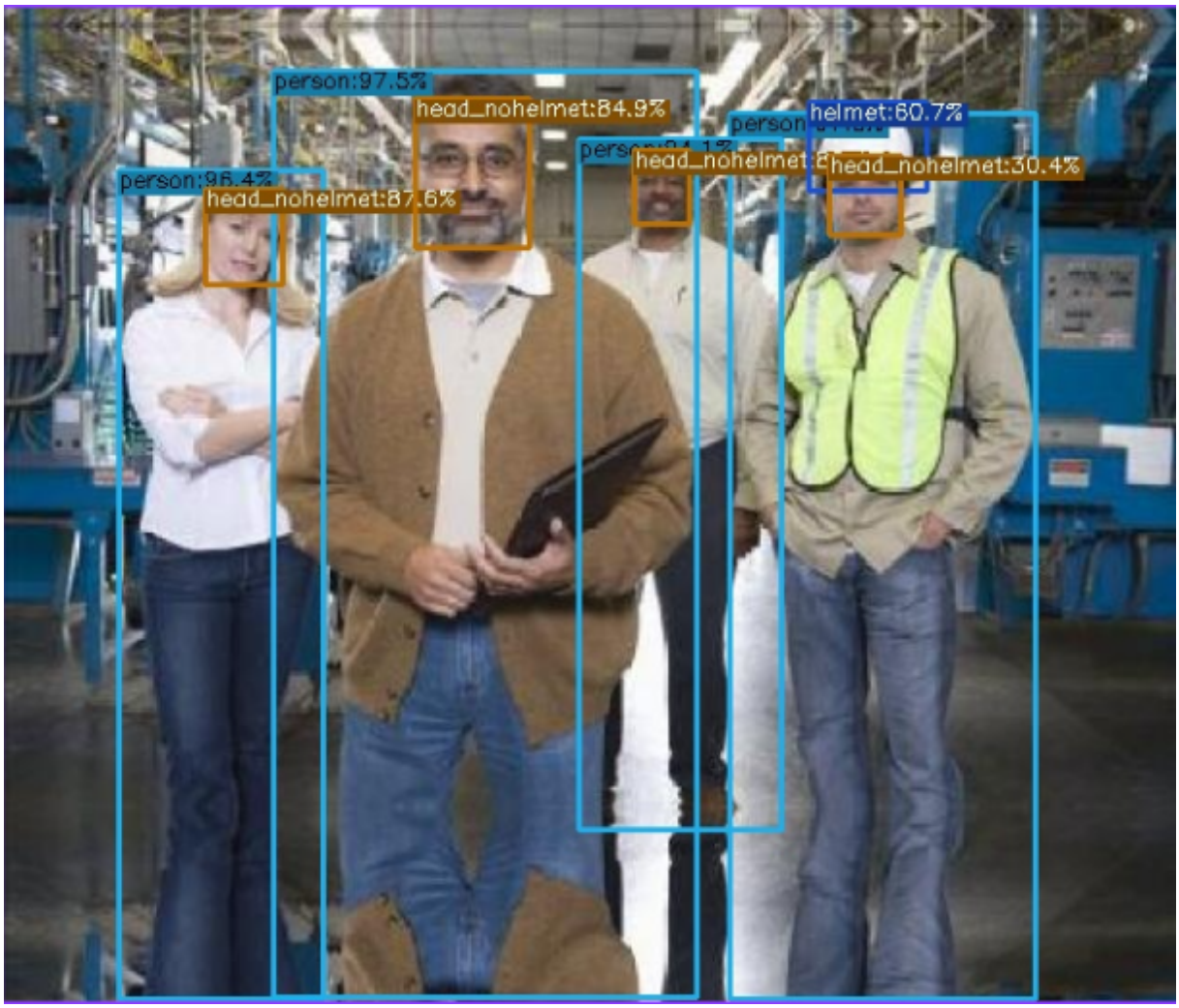

Figure 15: Similar 3: Another example of high-confidence people detection. Predictions still lacked safety vest class.

Figure 16: Scenario 4: Significant qualitative improvement in safety vest detection with high confidence values. However, it can be noted that some predictions are still missing from the image.

Figure 17: Scenario 4: Another example of the already mentioned problem of a duplicated helmet and head class.

Figure 18: Scenario 4: Detection of vest class in different scales

References

[1] M. Ferdous, S. M. M. Ahsan, *PeerJ Computer Science* **2022**, *8*, 24.

[2] Z. Ge, S. Liu, F. Wang, Z. Li, J. Sun, YOLOX: Exceeding YOLO Series in 2021, **2021**.

[3] N. D. Nath, A. H. Behzadan, S. G. Paal, *Automation in Construction* **2020**, *112*, 103085.

[4] Roboflow - Hard Hat Workers Object Detection Dataset, `https://public.roboflow.com/object-detection/hard-hat-workers`, Accessed: June 21, 2023.

[5] S. Yang, P. Luo, C. C. Loy, X. Tang in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), **2016**.

[6] Yolox Face Detection, `https://github.com/VarCode-ai/yolox-face-detection`.