Wheat Head Detection from High Resolution RGB Labelled Images Using Convolutional Neural Networks

Xinyu Yao and Selvyn Perez
(https://github.com/xinyuyao22/Global-Wheat-Detection)

Wheat head density estimation is an appealing trait for plant breeders. Current manual counting is tedious and inefficient. In this study three types of CNN architectures were investigated: (i) YOLO, (ii) EfficientDet, and (iii) Faster R-CNN. Models were evaluated on the mean average precision at different intersections over union (IoU) thresholds, ranging from 0.5 to 0.75 with a step size of 0.05. Further, the number of wheat heads detected from the RGB images will be compared with the number of wheat heads labelled. YOLO got the best performance with $AP_{0.5...0.75}$ of 0.75, mAP@0.5 of 0.86, and rMSE of 10%.

Keywords: Wheat head; Object detection; Object counting; Field imaging; Convolutional neural networks

1. Introduction

Wheat head density is associated with crop yield, but is a difficult and tedious trait for breeders to efficiently measure. Further, it is prone to sampling errors when the sampling area is small due to limited human resources. Computer vision approaches provide a potential solution to increase the throughput as well as the spatial representativeness, leading potentially to an improved accuracy. A number of studies based on high spatial resolution imaging systems applied to plant phenotyping under field conditions have received much attention in recent years[1].

While previous studies[2,3,4] have tested wheat head detection methods on individual datasets, in practice these deep learning models are difficult to scale to real-life phenotyping platforms, since they are trained on limited datasets, with expected difficulties when extrapolating to new situations. Most training datasets are limited in terms of genotypes, geographic areas and observational conditions. Wheat head morphology may significantly differ between genotypes with notable variation in head morphology, including size, inclination, color and the presence of awns[5]. The appearance of heads and the background canopy also change significantly from emergence to maturation due to ripening and senescence[6]. Further, planting densities and patterns vary globally across different cropping systems and environments, with possible overlap between heads for the higher head densities[5]. To fill the need for a large and diverse wheat head dataset with consistent labelling, the Global Wheat Head Detection (GWHD) dataset[5] was published, which can be used to benchmark methods proposed in the

computer vision community. The GWHD dataset results from the harmonization of several datasets coming from seven countries and three continents, at different growth stages with a wide range of genotypes.

The advances in computation capacity along with the availability of very large collections of labelled images have fostered enhanced machine learning methods based on convolutional neural networks (CNNs) in the field of computer vision[7]. Existing object detectors are mostly categorized by whether they have a region-of interest proposal step (two-stage)[8,9] or not (one-stage)[10,11]. R-CNN and Faster R-CNN have been demonstrated to be efficient for detection and analysis of wheat head for certain genotypes, geographic areas and observational conditions[2,12]. While two-stage detectors tend to be more flexible and more accurate, one-stage detectors are often considered to be simpler and more efficient by leveraging predefined anchors[13]. Yolo and EfficientDet have achieved state-of-art average precision (AP) on object detection.

The main objective of this study is to evaluate deep learning approaches for high-throughput wheat head counting under field conditions. For this purpose, three types of CNN architectures were investigated: (i) YOLOv5, (ii) EfficientDet, and (iii) Faster R-CNN. Models were evaluated on the mean average precision at different intersections over union (IoU) thresholds, ranging from 0.5 to 0.75 with a step size of 0.05. Further, the number of wheat heads detected from the RGB images will be compared with the number of wheat heads labelled.

2. Literature Review

2.1 Image Augmentation

Data augmentation is a commonly used technique for increasing both the size and the diversity of labeled training sets by leveraging input transformations that preserve corresponding output labels. In computer vision, image augmentations have become a common implicit regularization technique to combat overfitting in deep learning models and are ubiquitously used to improve performance. While most deep learning frameworks implement basic image transformations, the list is typically limited to some variations of flipping, rotating, scaling, and cropping. Moreover, image processing speed varies in existing image augmentation libraries.

Albumentations[14] is an open source Python library for fast and flexible image augmentations. It efficiently implements a rich variety of image transform operations that are optimized for performance, and does so while providing a concise, yet powerful image augmentation interface for different computer vision tasks, including object classification, segmentation, and detection.

2.2 Object Detection Models

A modern detector is usually composed of two parts, backbone which is pre-trained on ImageNet and a head which is used to predict classes and bounding boxes of objects[11]. As to the head part, it is usually categorized into two kinds, i.e., one-stage object detector and two-stage object detector. The most representative two-stage object detector is the R-CNN series, including fast R-CNN , faster R-CNN[23], R-FCN, and Libra R-CNN. As for one-stage object detectors, the most representative models are YOLO, SSD, and RetinaNet. Object detectors developed in recent years often insert some layers between backbone and head, and these layers are usually used to collect feature maps from different stages. We can call it the neck of an object detector (Fig. 1). Usually, a neck is composed of several bottom-up paths and several top down paths. Networks equipped with this mechanism include Feature Pyramid Network (FPN), Path Aggregation Network (PANet) , NAS-FPN, and BiFPN (Fig. 2).

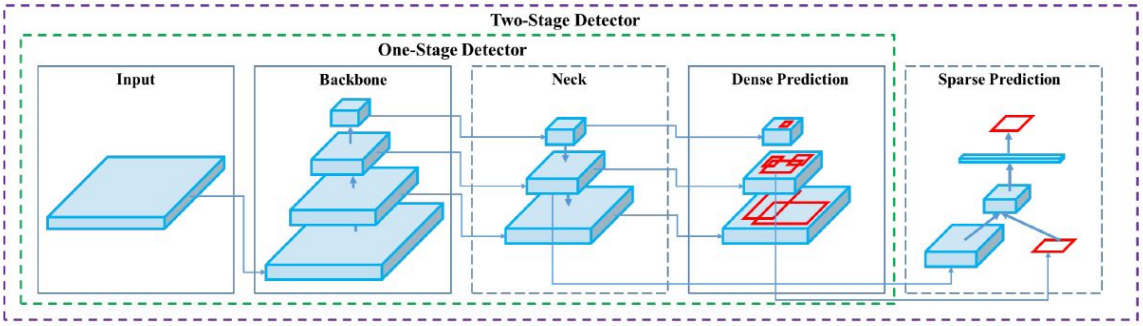

Fig. 1 Object Detector.

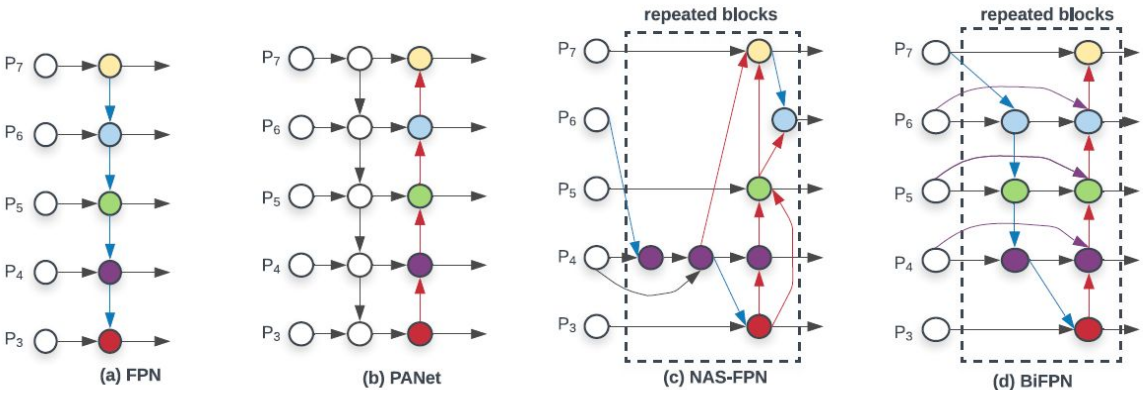

Fig. 2 Feature network design[22] – (a) FPN introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P3 - P7); (b) PANet adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) BiFPN with better accuracy and efficiency trade-offs.

Table 1 Comparison of Object Detection Models

Models	Backbone	Neck	Parameters
YOLOv5	CSPDarknet53	PANet	27.6M
EfficientDet-D5	EfficientNet-B5	BiFPN	34M
Faster-RCNN	ResNet-50	FPN	23M

2.2.1 YOLO

Similar to the architecture of YOLOv4[15] , YOLOv5 consists of CSPDarknet53 as backbone, SPP additional module, PANet path-aggregation neck, and YOLOv3 (anchor based) head.

In addition, YOLO v5 uses:

Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing

Bag of Specials (BoS) for backbone: Leaky ReLU activation, Cross-stage partial connections (CSP), Multiinput weighted residual connections (MiWRC)

Bag of Freebies (BoF) for detector: GIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyperparameters, Random training shapes

Bag of Specials (BoS) for detector: Sigmoid activation, SPP-block, SAM-block, PAN path-aggregation block (Fig. 2 (b)), DIoU-NMS

All YOLO anchor boxes are auto-learned in YOLOv5 when you input your custom data. YOLOv5 runs faster than EfficientDet with comparable performance (Fig. 3).

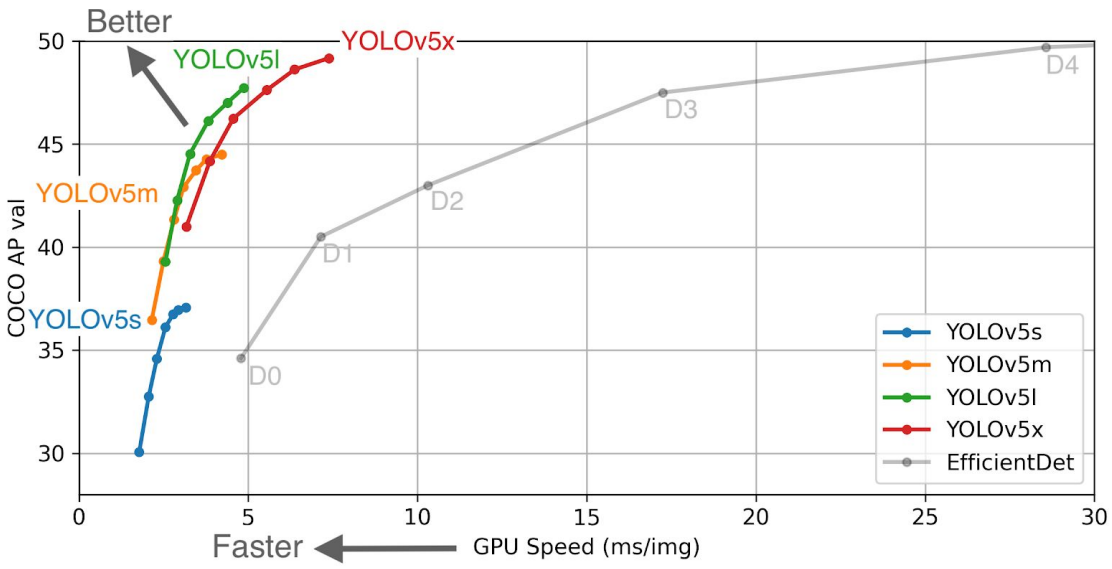

Fig. 3 Comparison of the speed and accuracy of YOLOv5 and EfficientDet (https://github.com/ultralytics/yolov5).

2.2.2 EfficientNet

Scaling up ConvNets is widely used to achieve better accuracy. The most common way is to scale up ConvNets by their depth[16] or width[17]. Another less common, but increasingly popular, method is to scale up models by image resolution[18]. In previous work, it is common to scale only one of the three dimensions – depth, width, and image size. Though it is possible to scale two or three dimensions arbitrarily, arbitrary scaling requires tedious manual tuning and still often yields sub-optimal accuracy and efficiency. Mingxing Tan et al.[19] show that it is critical to balance all dimensions of network width/depth/resolution, and propose a simple yet effective compound scaling method, which uniformly scales network width, depth, called EfficientNet.

2.2.3 EfficientDet

While fusing different input features, most previous works[20,21] simply sum them up without distinction; however, since these different input features are at different resolutions, they usually contribute to the fused output feature unequally. To address this issue, Mingxing Tan et al.[22] propose a simple yet highly effective weighted bi-directional feature pyramid network (BiFPN) (Fig. 2 (d)), which introduces learnable weights to learn the importance of different input features, while repeatedly applying topdown and bottom-up multi-scale feature fusion.

Combining EfficientNet backbones with BiFPN and compound scaling, Mingxing Tan et al.[22] have developed a new family of object detectors, named EfficientDet (Fig. 4), which

consistently achieve better accuracy with much fewer parameters and FLOPs than previous object detectors.

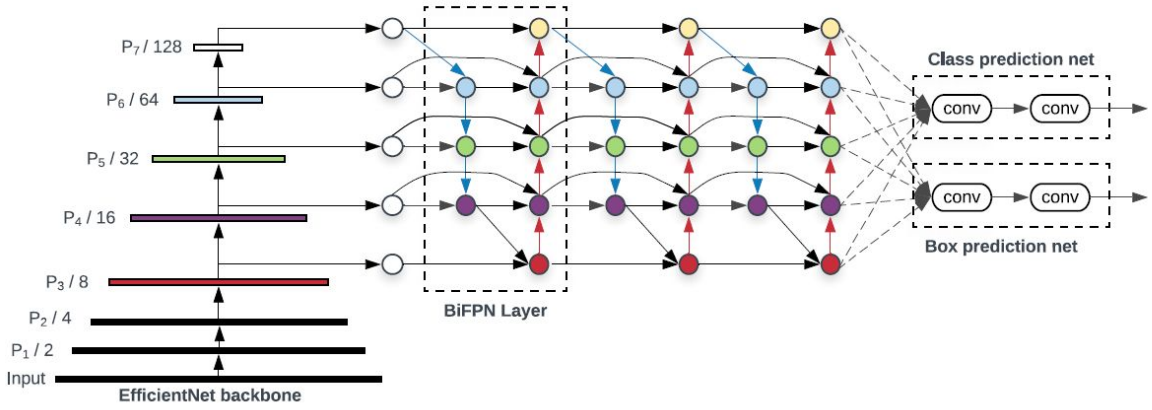

Fig. 4 EfficientDet architecture[22] – It employs EfficientNet as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints.

2.2.4 Faster R-CNN

A Faster R-CNN[23] constructs a single, unified model composed of RPN (region proposal network) and Fast R-CNN with shared convolutional feature layers. Previous algorithms (R-CNN & Fast R-CNN) use selective search to find out the region proposals. Selective search performs poorly in comparison to Faster R-CNN. Instead of using a selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.

Faster-RCNN Workflow

1. Image Input - Images are passed through a pre-trained CNN up until an intermediate layer (Transfer Learning)
2. Feature Extractor - A Region Proposal Network (RPN) uses the features that the CNN computed to find up to a predefined number of regions (bounding boxes), which may contain objects
3. Region of Interest (RoI) Pooling - Extract previous features which would correspond to the relevant objects into a new tensor
4. R-CNN module - Classifies the content in the bounding box (or discard it, using “background” as a label) and adjusts the bounding box coordinates (so it better fits the object)

2.3 Weighted Boxes Fusion (WBF)

WBF[24] algorithm can be described in following steps:

1. List B contains all the predicted boxes from all N models, and is sorted in a decreasing order of confidence scores C.

2. Empty list L and F (Fusion) are created.

3. Boxes in list B are iterated from the start and box whose Intersection over Union (IoU) with current box is above a certain threshold (default=0.55) is added to list L.

4. Boxes in list L are fused into one box according to the following fusion formulas and added to list F. Coordinates of the fused box is a weighted sum of coordinates of each box, where weights are confidence for boxes:

$$C = \frac{C_1+C_2+...+C_T}{N} \quad (1)$$

$$X1 = \frac{C_1*X1_1+C_2*X1_2+...+C_T*X1_T}{C_1+C_2+...+C_T} \quad (2)$$

$$X2 = \frac{C_1*X2+C_2*X2_2+...+C_T*X2_T}{C_1+C_2+...+C_T} \quad (3)$$

$$Y1 = \frac{C_1*Y1_1+C_2*Y1_2+...+C_T*Y1_T}{C_1+C_2+...+C_T} \quad (4)$$

$$Y2 = \frac{C_1*Y2_1+C_2*Y2_2+...+C_T*Y2_T}{C_1+C_2+...+C_T} \quad (5)$$

3. Dataset and Exploratory Analysis

3.1 Dataset

All images share a common format of 1024×1024 px with a resolution of 0.1-0.3mm per pixel. The training dataset corresponds to 3422 images from Europe and North America representing 73% of the whole GWHD dataset images. To evaluate model performance, including robustness against unseen images, the test data set includes all the images from Australia, Japan, and China, representing 1276 images.

The field in China, where the images are collected, has higher sowing density (seeds*m^{-2}), therefore a data augmentation method: mixup is considered to train models with images of higher wheat head density. However, the fields in Japan and Australia, where the images are collected, have relatively lower sowing density (seeds*m^{-2}), therefore CoarseDropout is considered to train models with images of lower wheat head density. This dataset is diverse in terms of developmental stages when the images were collected, from flowering to ripening, which resulted in the color of wheat heads and canopy ranging from green to yellow. Therefore, randomly changing hue, saturation, and

value (HSV) of training images was considered to train models with images of various colors of wheat heads and canopy. Although all images were acquired at nadir-viewing direction, half the field of view along the image diagonal varies from 10° to 46°. Some geometric distortions may be observed for few sub-datasets due to the different lens characteristics of the cameras used, excerpts of the acquired images have different. Images collected from Japan and Switzerland are particularly affected by this issue. Therefore, PiecewiseAffine, HorizontalFlip, VerticalFlip, and RandomRotate90 were considered to train models with images of different perspectives.

3.2 Exploratory Analysis

(A) Distributions of brightness and contrast of training images

RGB images are read using cv2 and then converted to grayscale, where the mean and standard deviation of all the pixels is calculated as the brightness and contrast of the image. The distributions of brightness and contrast are relatively broad (Fig. 5), so randomly changing brightness and contrast of input images were considered for training image augmentation.

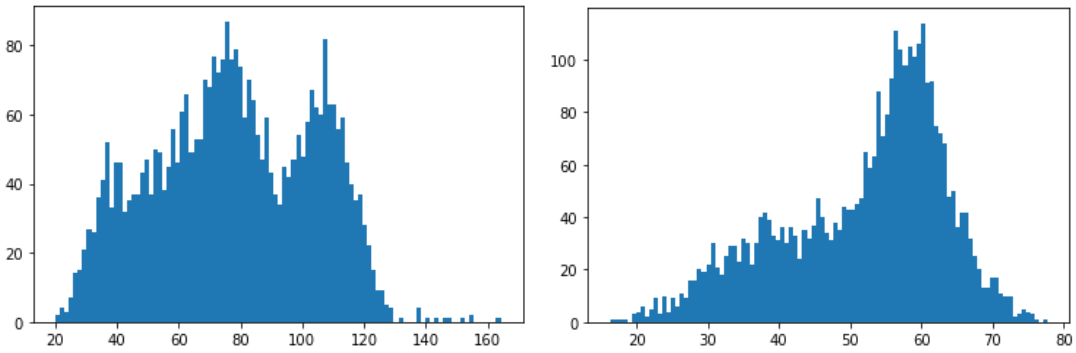

Fig. 5 Distribution of brightness (left) and contrast (right) of training images.

(B) Distribution of IoU of labelled bounding boxes

There are a total of 138831 labelled bounding boxes, of which over half have overlap with at least one bounding box (Fig. 6). This high occurrence of overlapping and occluded objects is unique to the GWHD Dataset and makes detection more challenging.

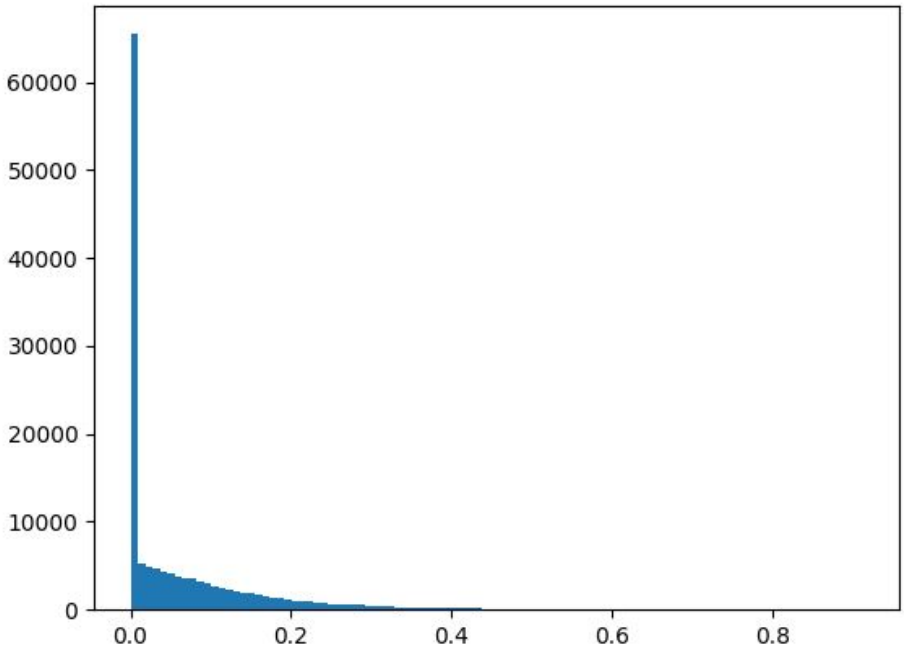

Fig. 6 Distribution of IoU of labelled bounding boxes

4. Method

This research framework consists of five consecutive steps: (i) splitting training and validation sample sets, (ii) image augmentation, (iii) training detection models, and (iv) models evaluation.

The specific algorithm (workflow) is as follows:

Step 1: Images in the training dataset were splitted into 5 folders, 4 folders were used for training models and 1 folder was used for model evaluation;

Step 2: HorizontalFlip, VerticalFlip, RandomRotate90, RandomBrightnessContrast, HueSaturationValue, Blur, PiecewiseAffine, CoarseDropout, and MixUp were performed while loading images to model training;

Step 3: YOLO, EfficientDet and Faster R-CNN were implemented using Pytorch and trained on NVIDIA Tesla P100;

Step 4: Models were evaluated on the mean average precision at different IoU threshold, ranging from 0.5 to 0.75 with a step size of 0.05, as well as root mean squared error (RMSE) and the relative RMSE (rRMSE).

4.1 Image Augmentation

Though the GWHD dataset results from the harmonization of several datasets coming from seven countries and three continents, at different growth stages with a wide range of genotypes, there are still limitations in terms of genotypes, geographic areas and observational conditions compared to the worldwide situation. Therefore, heavy augmentations from albumentations library (https://albumentations.ai) were performed while loading images to model training.

4.1.1 HorizontalFlip, VerticalFlip, and RandomRotate90

HorizontalFlip, VerticalFlip, and RandomRotate90 (Fig. 7) were performed on 50% of training images in each epoch to train models with images of different perspectives. Although all images were acquired at nadir-viewing direction, half the field of view along the image diagonal varies from 10° to 46°. Some geometric distortions may be observed for few sub-datasets due to the different lens characteristics of the cameras used, excerpts of the acquired images have different. Images collected from Japan and Switzerland are particularly affected by this issue. Therefore, HorizontalFlip, VerticalFlip, and RandomRotate90 were performed.

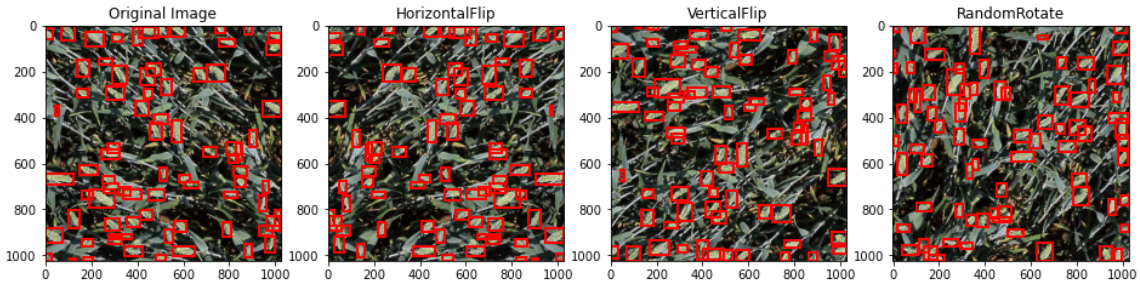

Fig. 7 Image augmentation of HorizontalFlip, VerticalFlip, and RandomRotate

4.1.2 RandomBrightnessContrast

RandomBrightnessContrast (brightness_limit=0.4, contrast_limit=0.85) (Fig. 8) was performed on 90% of training images in each epoch. The distributions of brightness and contrast are relatively broad (Fig. 5), so randomly changing brightness and contrast of input images were considered for training image augmentation.

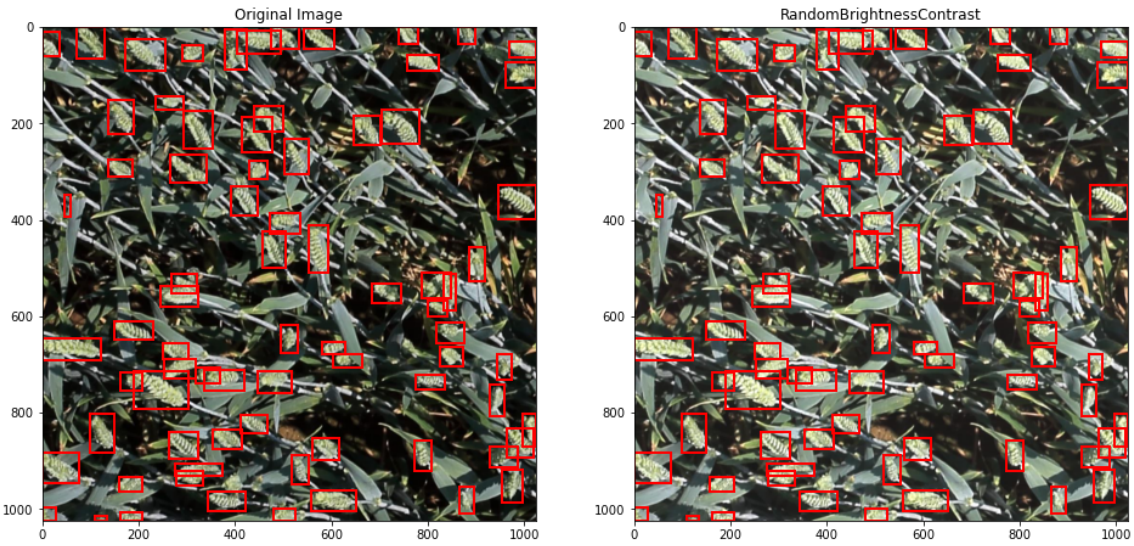

Fig. 8 Image augmentation of RandomBrightnessContrast

4.1.3 HueSaturationValue

HueSaturationValue (hue_shift_limit=0.4, sat_shift_limit=0.4, val_shift_limit=0.4) (Fig. 9) was performed on 90% of training images in each epoch. This dataset is diverse in terms of developmental stages when the images were collected, from flowering to ripening, which resulted in the color of wheat heads and canopy ranging from green to yellow. Therefore, randomly changing hue, saturation, and value (HSV) of training images was considered to train models with images of various colors of wheat heads and canopy.

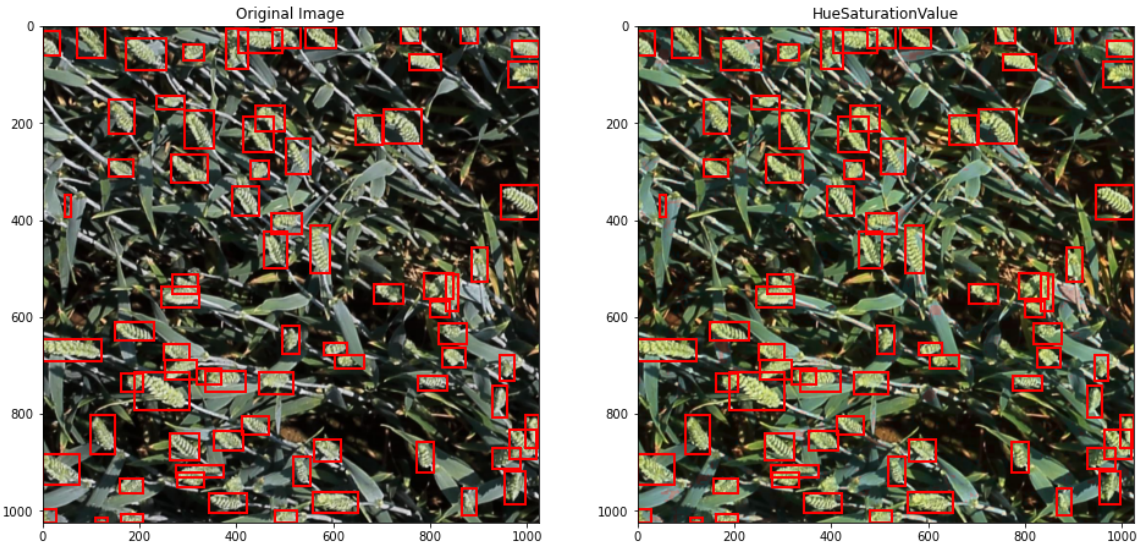

Fig. 9 Image augmentation of HueSaturationValue

4.1.4 CoarseDropout

CoarseDropout (max_holes=8, max_height=8, max_width=8, min_holes=None, min_height=None, min_width=None, fill_value=1, always_apply=False) (Fig. 10) was performed on 50% of training images in each epoch. The fields in Japan and Australia, where the images are collected, have relatively lower sowing density (seeds*m^{-2}), therefore CoarseDropout is considered to train models with images of lower wheat head density.

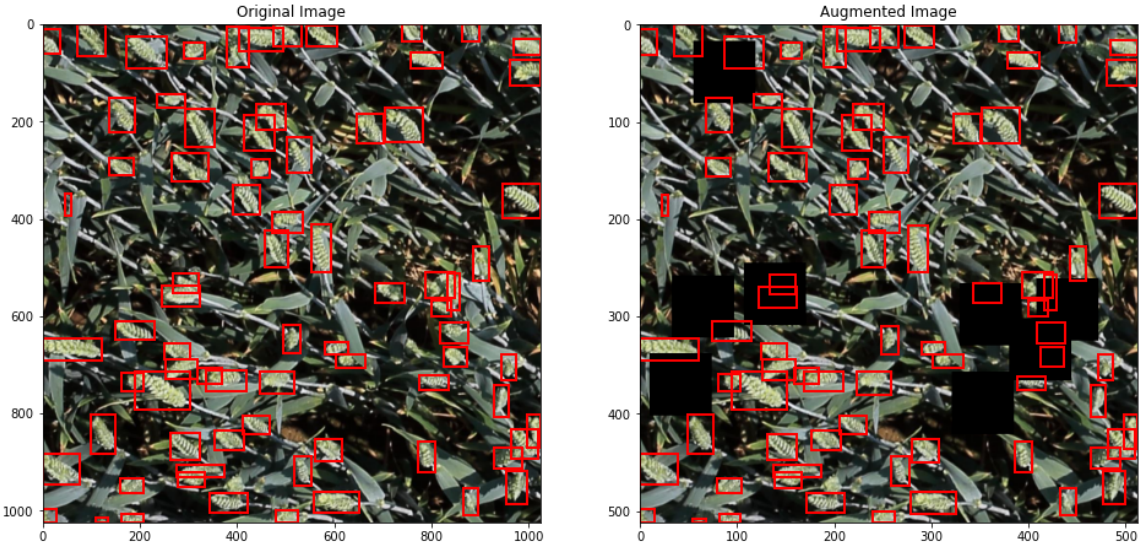

Fig. 10 Image augmentation of CoarseDropout

4.1.5 Mixup

Mixup combines pairs of images and their labelled bounding boxes and feeds to the model training[25]. The original image and a randomly selected image from the training dataset are added pixel-wise to generate the mixup image (Fig. 11), where labelled bounding boxes are also augmented. The field in China, where the images are collected, has higher sowing density (seeds*m^{-2}), therefore mixup is considered to train models with images of higher wheat head density.

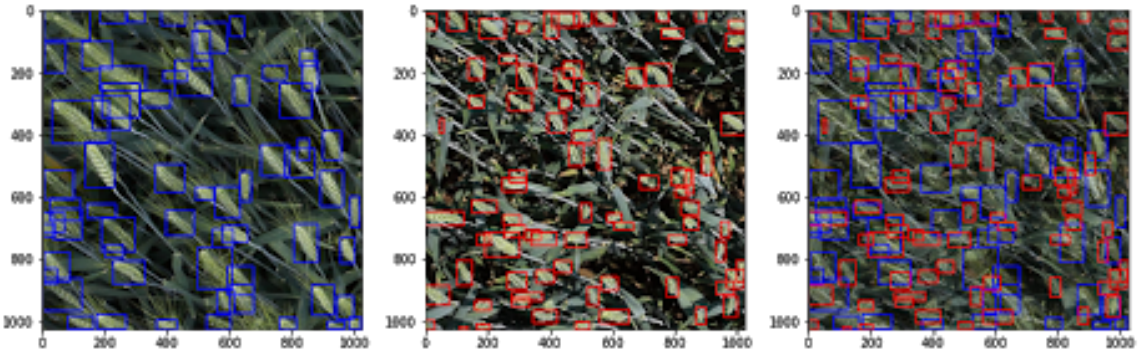

Fig. 11 Image augmentation of Mixup, using random selected image (left) and original image (middle) to generate the augmented image (right).

4.1.6 PiecewiseAffine

Apply affine transformations that differ between local neighbourhoods. This augmenter places a regular grid of points on an image and randomly moves the neighbourhood of these points around via affine transformations, which leads to local distortions (Fig. 12). Some geometric distortions have been observed in images collected from Japan and Switzerland due to the field of view along the image diagonal varying from 10° to 46°. Therefore, PiecewiseAffine was considered to train models with images containing geometric distortions.

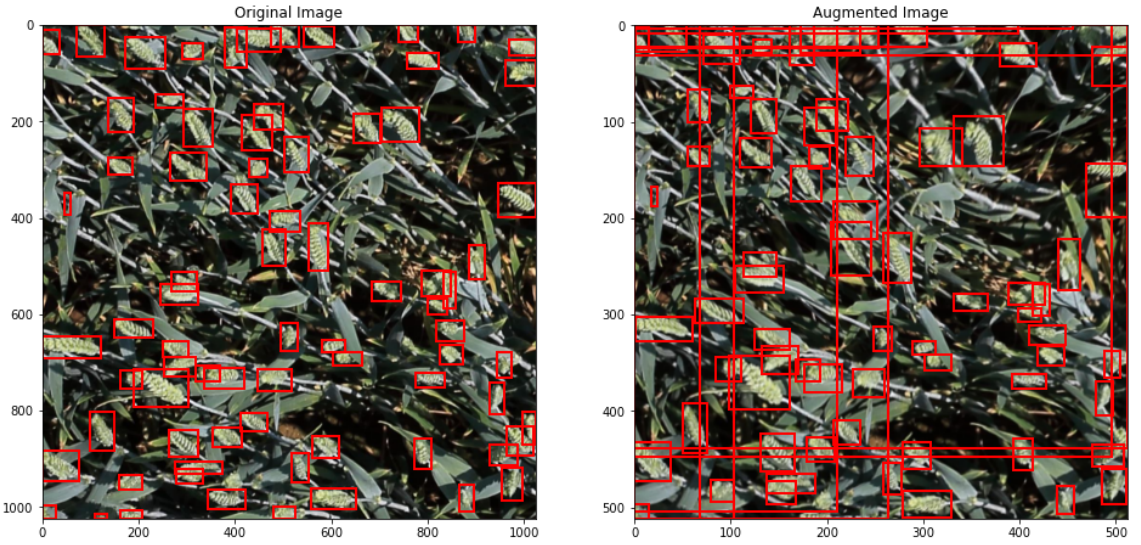

Fig. 12 Image augmentation of PiecewiseAffine

4.1.7 Blur

As images are taken from the field under natural conditions, motion or wind could cause possible blurring, therefore, Blur is performed on training image dataset (Fig. 13) to increase the robustness of the models.

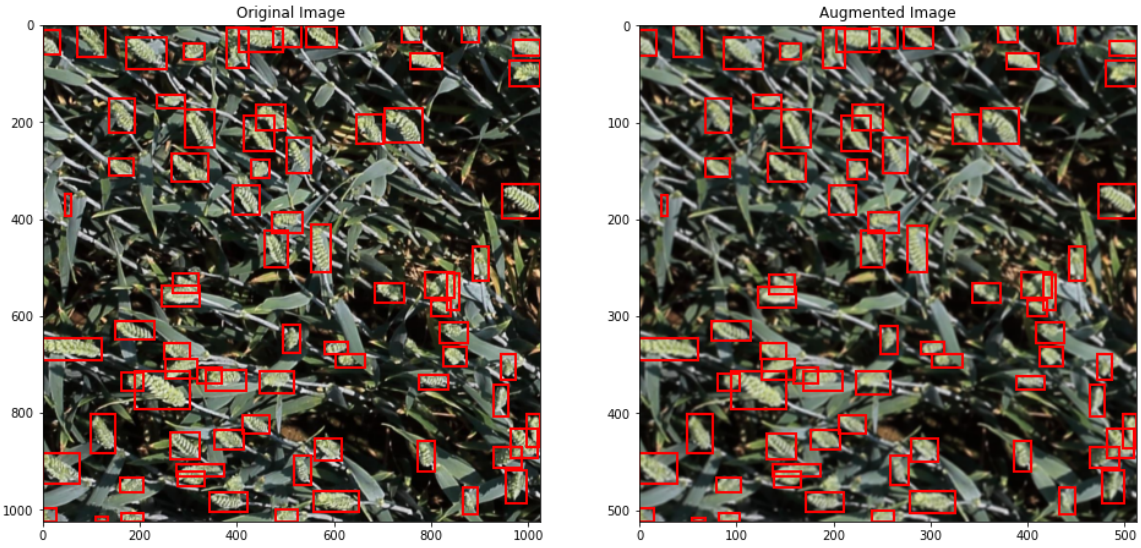

Fig. 13 Image augmentation of Blur

4.2 Training YOLO

The open-source repository created by Ultralytics was cloned from (https://github.com/ultralytics/yolov5), and requirements.txt dependencies are installed, including Python>=3.8 and PyTorch>=1.6. Dataset.yaml was then created, which defines: a path to a directory of training images (or path to a *.txt file with a list of training images); the same for our validation images; the number of classes; a list of class names. In addition to image augmentation mentioned above, YOLOv5 training pipeline employs some other augmentations, such as CutMix, Mosaic, etc. YOLOv5x was trained on images sizes of 1024*1024, batch size of 2, with APEX for 100 epochs. Learning rate was set as 0.01, with momentum of 0.937, and weight_decay of 0.0005. The model with the highest AP on validation dataset was saved and evaluated on test dataset.

4.3 Training EfficientDet-D5

EfficientDet-D5 was trained on images sizes of both 512*512 and 1024*1024. Random search was performed for hyperparameter tuning. Learning rate was randomly selected from loguniform (0.001, 0.00001), learning rate optimizer of Adam, AdamW, and SGD was randomly selected, and learning rate scheduler of OneCycleLR, ReduceLROnPlateau, and CosineAnnelingLR was randomly selected. The best

combination of hyperparameters is learning rate of 0.0002, with optimizer of AdamW, and optimizer of ReduceLROnPlateau. Model was trained for 40 epochs, and the model with the highest AP on validation dataset was saved and evaluated on the test dataset.

4.4 Training Faster R-CNN

Faster R-CNN was trained on image sizes of both 512*512 and 1024*1024.

Stochastic Gradient Descent (SGD) vs Adam

SGD is an optimization algorithm that is used to update network weights iteratively based on training data. Adam is an optimization algorithm that can be used instead of the classic.

The Adam optimizer performed better than the SGD optimizer during a smaller period of time.

The following parameters were used for SGD within the PyTorch function: learning rate=0.0001, momentum=0.0005, weight decay=0.0005, nesterov=True

The following parameters were used for Adam within the PyTorch function: learning rate=0.0001, betas=(0.9, 0.999), epsilon=1e-08, weight_decay=0.0005, amsgrad=False

	SGD		Adam	
Epoch	Loss	Precision	Loss	Precision
0	.6842	.6703	.7415	.6110
1	.6819	.6710	.6933	.6314
2	.6804	.6711	.6863	.6303
3	.6783	.6721	.6559	.6343
4	.6768	.6721	.6513	.6589
5	.6731	.6726	.6563	.6358
6	.6752	.6719	.6475	.7188
7	.6739	.6714	.6417	.7061
8	.6735	.6710	.6359	.6905
9	.6700	.6720	.6326	.6913

4.5 Criteria to Evaluation Algorithm

Models are evaluated on the mean average precision at different IoU thresholds. The IoU of a set of predicted bounding boxes and ground truth bounding boxes is calculated as:

$$\text{IoU}(A,B)=(A\cap B)/(A\cup B).$$

The metric sweeps over a range of IoU thresholds, at each point calculating an average precision value. The threshold values range from 0.5 to 0.75 with a step size of 0.05. In other words, at a threshold of 0.5, a predicted object is considered a "hit" if its intersection over union with a ground truth object is greater than 0.5.

At each threshold value t, a precision value is calculated based on the number of true positives (TP), false negatives (FN), and false positives (FP) resulting from comparing the predicted object to all ground truth objects:

$$\frac{TP(t)}{TP(t)+FP(t)+FN(t)}.$$

$$\frac{1}{|thresholds|}\sum_{t}\frac{TP(t)}{TP(t)+FP(t)+FN(t)}.$$

Bounding boxes will be evaluated in order of their confidence levels in the above process. This means that bounding boxes with higher confidence will be checked first for matches against solutions, which determines what boxes are considered true and false positives. Lastly, the score returned by the competition metric is the mean taken over the individual average precisions of each image in the test dataset.

The head counting performances were also quantified using root mean squared error (RMSE) and the relative RMSE (rRMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{n}(t_k - c_k)^2}$$

$$\text{rRMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{n}\left(\frac{t_k - c_k}{t_k}\right)^2}$$

Where N denotes the number of test images, *tk* and *ck* are respectively the reference and estimated counts for image k.

5. Results

5.1 Results of YOLO

On the validation dataset, the model got $AP_{50...75}$ of 0.75, mAP@0.5 of 0.86, MSE of 4.33, and rMSE of 10.3%. On the test dataset, the model got $AP_{50...75}$ of 0.70. The model performed well on detecting wheat heads (Fig. 14). The head count estimated with model YOLO was in relatively good agreement with the head count labelled with R^2 of 0.96 (Fig. 15).

Fig. 14. An example of output image using YOLO.

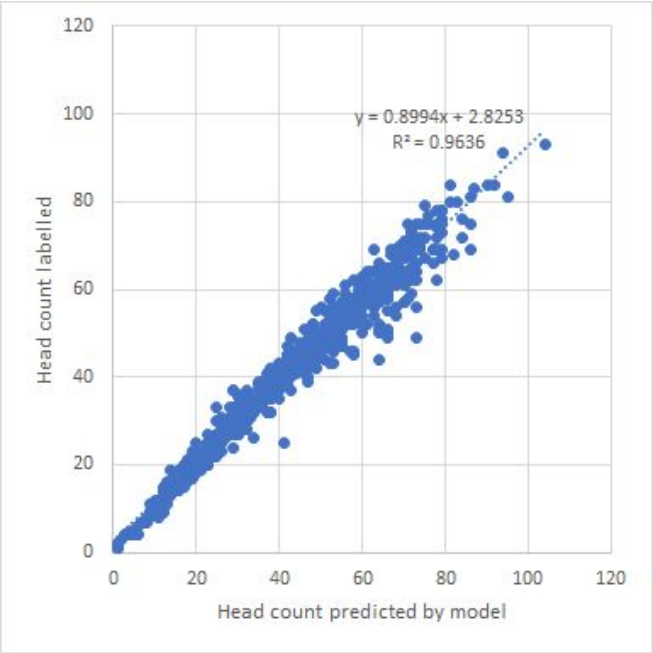

Fig. 15. Comparison between the number of head count in each image visually labelling and that detected using YOLO.

5.2 Results of EfficientDet

EfficientDet trained on images sizes of 512*512 had better performance than trained on images sizes of 1024*1024. On the validation dataset, the model got $AP_{0.5...0.75}$ of 0.72, mAP@0.5 of 0.81, MSE of 3.44, and rMSE of 10.6%. On the test dataset, the model got $AP_{0.5...0.75}$ of 0.63. The model performed well on detecting wheat heads (Fig. 16). The head count estimated with model EfficientDet was in relatively good agreement with the head count labelled with R^2 of 0.97 (Fig. 17).

Fig. 16 An example of output image of EfficientDet

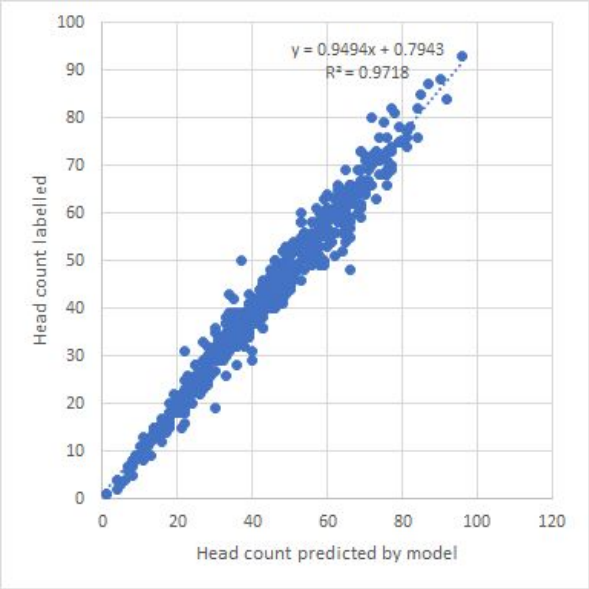

Fig. 17 Comparison between the number of head count in each image visually labelling and that detected using EfficientDet.

5.3 Results of Faster R-CNN

Faster R-CNN results after a condensed period of time (10 epochs) with the following parameters:

optimizer = torch.optim.SGD(params, lr=0.001, momentum=0.005,
weight_decay=0.0005, nesterov=True)
Validation Precision: .672453
---
optimizer = torch.optim.SGD(params, lr=0.0001, momentum=0.95,
weight_decay=0.0005, nesterov=True)
Validation Precision: .672577
---
optimizer = torch.optim.SGD(params, lr=0.0001, momentum=0.0005,
weight_decay=0.0005, nesterov=True)
Validation Precision: .672613
---
optimizer = torch.optim.Adam(param, lr=0.0001, betas=(0.9, 0.999), eps=1e-08,
weight_decay=0.0005, amsgrad=False)
Validation Precision: .718781
---

optimizer = torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.0005, amsgrad=False)
Validation Precision: .327536

Experimenting on a longer time (50 epochs) for both SGD and Adam resulted in the following:

optimizer = torch.optim.SGD(params, lr=0.0001, momentum=0.005, weight_decay=0.0005, nesterov=True)
Validation Precision: .748047
---
optimizer = torch.optim.Adam(param, lr=0.0001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.0005, amsgrad=False)
Validation Precision: .761707

6. Discussion

6.1 Image acquisition recommendations

Near nadir viewing directions that limit the overlap between heads are recommended by David, E. et al., especially in the case of high-density head population[1]. However, image plots from an oblique perspective as opposed to the more common nadir perspective were recommended by Hasan, M.M. et al.. In an oblique view a significant number of spike features such as texture, color, shape etc. can be discerned easily. These features can be more readily extracted for the purposes of various plant phenotyping applications such as spike counting, spike shape measurement, spike texture, disease detection, grain yield estimation etc.

We would recommend training CNN models on images dataset with both nadir and oblique views, so the models could have robust performance in different situations and unseen fields and images.

6.2 Models trained on different sizes of images

YOLO and Faster R-CNN trained on images sizes of 1024*1024 got better performance compared to being trained on images sizes of 512*512, which agrees with the suggestions from Christian Eggert et al, that Faster-RCNN has difficulties with small objects. EfficientDet got better performance when trained on images sizes of 512*512.

6.3 Faster R-CNN - SGD and Adam Optimizer Comparisons

The results after 10 epochs find that lowering the learning rate of the SGD optimizer did not alter the validation precision in a significant way. Changing the momentum of the SGD as well did not alter the results in a significant way. However, using the different optimizer, Adam, is where the differences started to be realized. Using a learning rate of .0001, the precision using the Adam optimizer is significantly higher than the SGD optimizer.

Even though the alteration of the learning rate for SGD yielded similar precision results, this was not the same for altering the learning rate of the Adam optimizer. Increasing the learning rate by a thousandth resulted in a significant drop in performance.

Even though performance on Adam was still better after 50 epochs, which took a little over 2 hours, the SGD seemed to start to catch up to Adam. The learning rates continued to be the same. The idea behind this experiment was to keep the similar parameters where possible and to observe the final precision.

A possible conclusion to the reason the SGD seems to perform similar to the Adam optimizer overtime is due to the fact that the nesterov parameter is set to True. Nesterov Momentum (also called Nesterov Accelerated Gradient/NAG) are slight variations of normal gradient descent that can speed up training and improve convergence. This hypothesis was tested by setting nesterov to False and a precision of .731011 was observed.

7. Conclusion

In this study three types of CNN architectures were investigated: (i) YOLO, (ii) EfficientDet, and (iii) Faster R-CNN. Models were evaluated on the mean average precision at different intersections over union (IoU) thresholds, ranging from 0.5 to 0.75 with a step size of 0.05. Further, the number of wheat heads detected from the RGB images will be compared with the number of wheat heads labelled. YOLO got the best performance with AP of 0.75, mAP@0.5 of 0.86, and rMSE of 10%.

Reference:

1. Li, L., Zhang, Q. and Huang, D., 2014. A review of imaging techniques for plant phenotyping. *Sensors*, *14*(11), pp.20078-20111.
2. Madec, S., Jin, X., Lu, H., De Solan, B., Liu, S., Duyme, F., Heritier, E. and Baret, F., 2019. Ear density estimation from high resolution RGB imagery using deep learning technique. Agricultural and forest meteorology, 264, pp.225-234.

3. Milioto, A., Lottes, P. and Stachniss, C., 2018, May. Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs. In *2018 IEEE international conference on robotics and automation (ICRA)* (pp. 2229-2235). IEEE.
4. Ubbens, J., Cieslak, M., Prusinkiewicz, P. and Stavness, I., 2018. The use of plant models in deep learning: an application to leaf counting in rosette plants. *Plant methods*, *14*(1), p.6.
5. David, E., Madec, S., Sadeghi-Tehran, P., Aasen, H., Zheng, B., Liu, S., Kirchgessner, N., Ishikawa, G., Nagasawa, K., Badhon, M.A. and Pozniak, C., 2020. Global Wheat Head Detection (GWHD) dataset: a large and diverse dataset of high resolution RGB labelled images to develop and benchmark wheat head detection methods. *arXiv preprint arXiv:2005.02162*.
6. Anderegg, J., Yu, K., Aasen, H., Walter, A., Liebisch, F. and Hund, A., 2020. Spectral vegetation indices to track senescence dynamics in diverse wheat germplasm. *Frontiers in Plant Science*, *10*, p.1749.
7. LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. nature, 521(7553), pp.436-444.
8. Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).
9. Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91-99).
10. Tan, M., Pang, R. and Le, Q.V., 2019. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070*.
11. Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.
12. Hasan, M.M., Chopin, J.P., Laga, H. and Miklavcic, S.J., 2018. Detection and analysis of wheat spikes using convolutional neural networks. *Plant Methods*, *14*(1), p.100.
13. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. and Murphy, K., 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7310-7311).
14. Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M. and Kalinin, A.A., 2020. Albumentations: fast and flexible image augmentations. *Information*, *11*(2), p.125.
15. Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv preprint arXiv:2004.10934*.
16. He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. CVPR, pp. 770–778, 2016.
17. Zagoruyko, S. and Komodakis, N. Wide residual networks.BMVC, 2016.
18. Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1808.07233, 2018.

19. Tan, M. and Le, Q.V., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.
20. Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. CVPR, 2019.
21. Qijie Zhao, Tao Sheng, YongtaoWang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. AAAI, 2019.
22. Tan, M., Pang, R. and Le, Q.V., 2019. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070*.
23. Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91-99).
24. Solovyev, R. and Wang, W., 2019. Weighted Boxes Fusion: ensembling boxes for object detection models. *arXiv preprint arXiv:1910.13302*.
25. Zhang, H., Cisse, M., Dauphin, Y.N. and Lopez-Paz, D., 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.
26. Eggert, C., Brehm, S., Winschel, A., Zecha, D. and Lienhart, R., 2017, July. A closer look: Small object detection in faster R-CNN. In *2017 IEEE international conference on multimedia and expo (ICME)* (pp. 421-426). IEEE.
27. Sutskever, I., Martens J., Dahl G., Hinton G. - On the importance of initialization and momentum in deep learning http://www.cs.toronto.edu/~hinton/absps/momentum.pdf
28. DeVries, T., Taylor, G. - Improved Regularization of Convolutional Neural Networks with Cutout