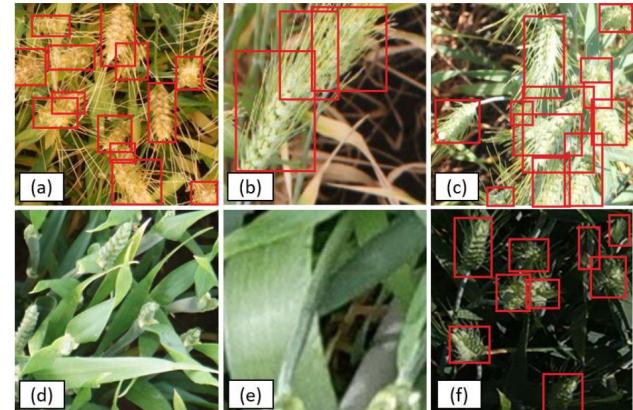

Global Wheat Head Detection

Selvin Perez and Xinyu Yao

Problem statement

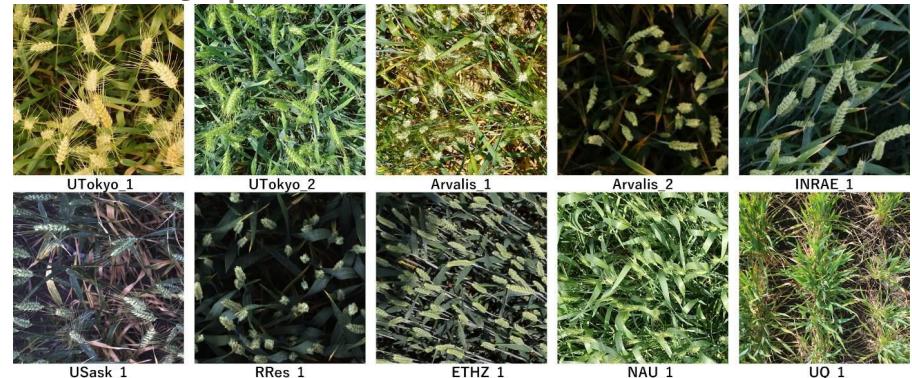
- Object detection: to detect wheat head from images taken in the field
- Developing a model with high accuracy of wheat heads counting and high precision of wheat heads detection in the field to help breeders and growers to predict production



Source: Global Wheat Head Detection (GWHD) dataset

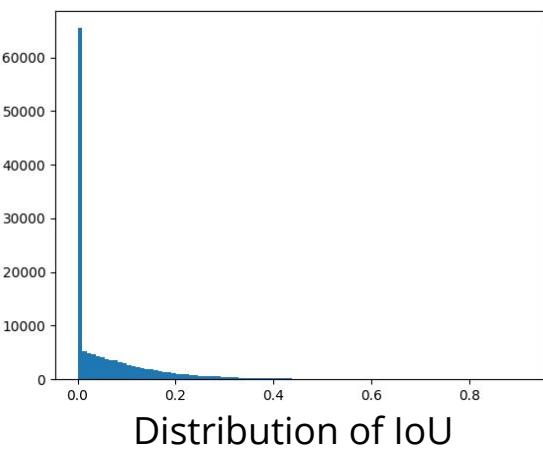
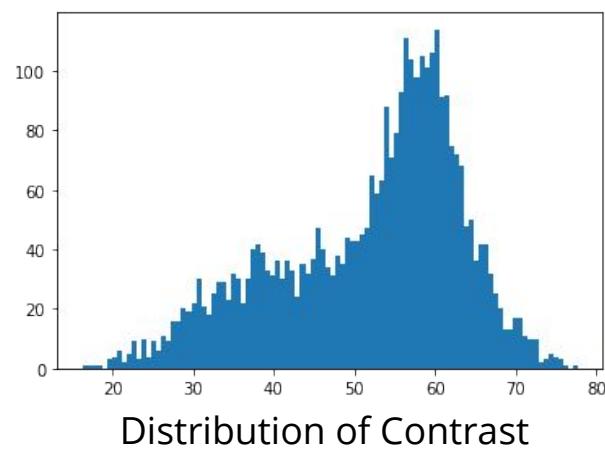
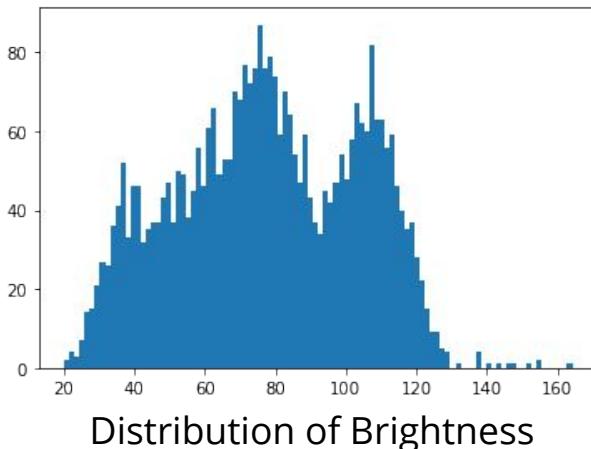
Dataset

- Contains 4,700 high-resolution RGB images and well labelled wheat heads collected from 7 countries around the world at different growth stages with a wide range of genotypes.
- Common format of 1024 * 1024 with resolution of 0.1-0.3 mm per pixel
- Training on 73% of images from Europe and North America
- Testing on 27% of images from Australia, Japan, and China



Exploratory Data Analysis

- The distribution of brightness and contrast are relatively broad
- There are a total of 138831 labelled bounding boxes, of which over 50% have overlap with at least one bounding box

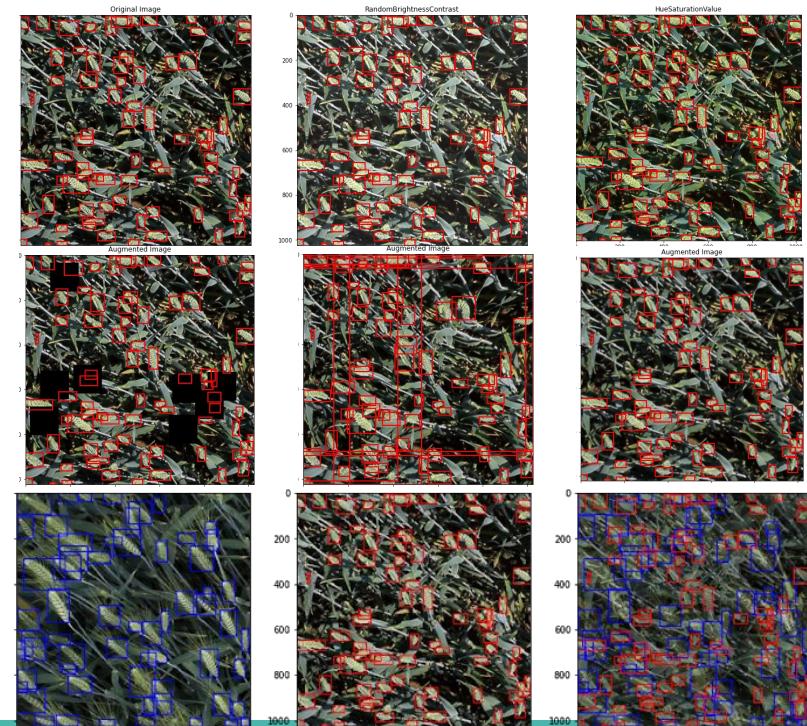


Workflow

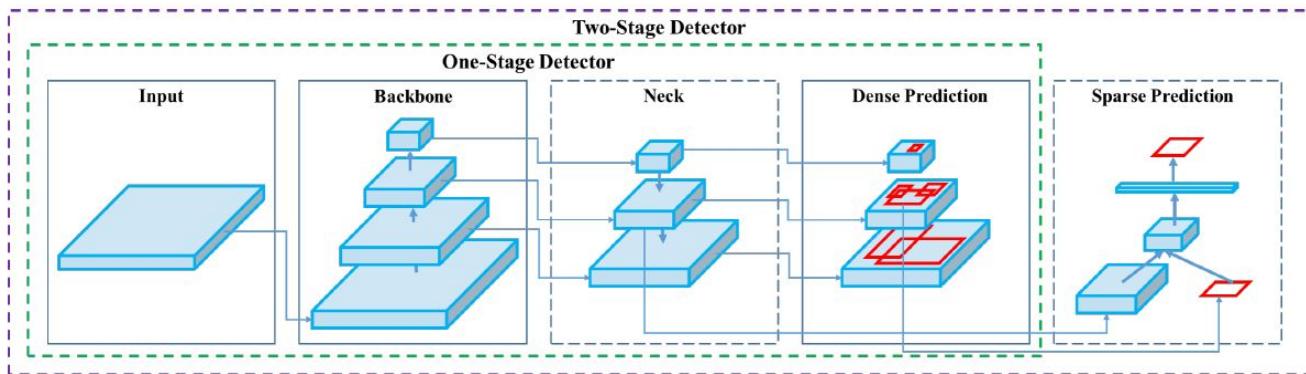
- Step 1: Images were splitted into 5 folders, of which 4 folders were used for training models and 1 folder was used for model evaluation;
- Step 2: Heavy image augmentation was performed while loading images to model training;
- Step 3: YOLO, EfficientDet and Faster R-CNN were implemented using Pytorch and trained on NVIDIA Tesla P100;
- Step 4: Models were evaluated on the mean average precision (mAP), as well as root mean squared error (RMSE) and the relative RMSE (rRMSE).

Image Augmentation

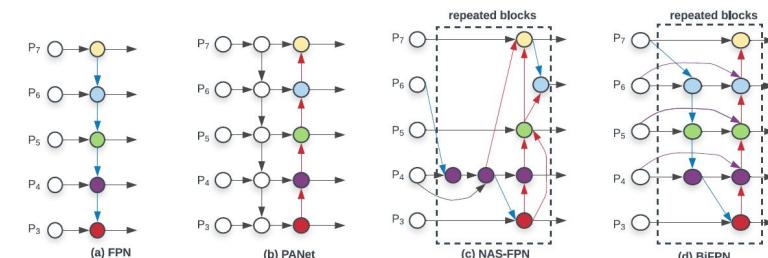
- HorizontalFlip, VerticalFlip and RandomRotate90
- RandomBrightnessContrast
- HueSaturationValue
- CoarseDropout
- PiecewiseAffine
- Blur
- MixUp



One-stage Object Detection: YOLO and EfficientDet



| Model | Backbone | Neck | Parameters |
|-----------------|-----------------|-------|------------|
| YOLO | CSPDarknet53 | PANet | 27.6M |
| EfficientDet-D5 | EfficientNet-B5 | BiFPN | 34M |

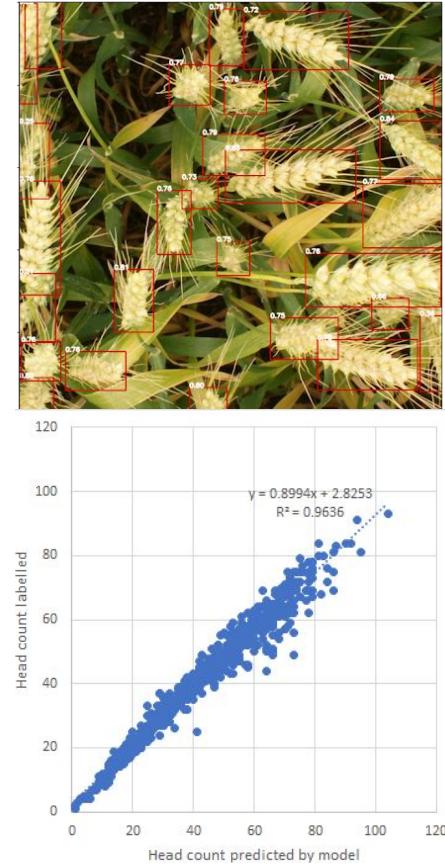


Training YOLO

- The open-source repository created by Ultralytics was cloned from (<https://github.com/ultralytics/yolov5>)
- YOLOv5 training pipeline has already employed some other augmentations, such as CutMix, Mosaic, etc.
- YOLOv5x was trained on images sizes of 1024*1024, batch size of 2, for 100 epochs.
- Learning rate was set as 0.01, with momentum of 0.937, and weight_decay of 0.0005.

Evaluation 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 head count estimated with model YOLO was in relatively good agreement with the head count labelled with R² of 0.96

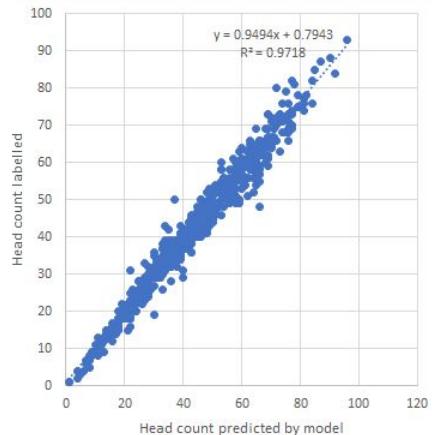


Training EfficientDet

- EfficientDet-D5 was trained on images sizes of both 512*512 and 1024*1024.
- Random search for hyperparameter tuning
 - lr: loguniform (0.001, 0.00001), **0.0002**
 - lr_optimizer: Adam, **AdamW**, SGD
 - lr_scheduler: OneCycleLR, **ReduceLROnPlateau**, CosineAnnealingLR
- Model was trained for 40 epochs

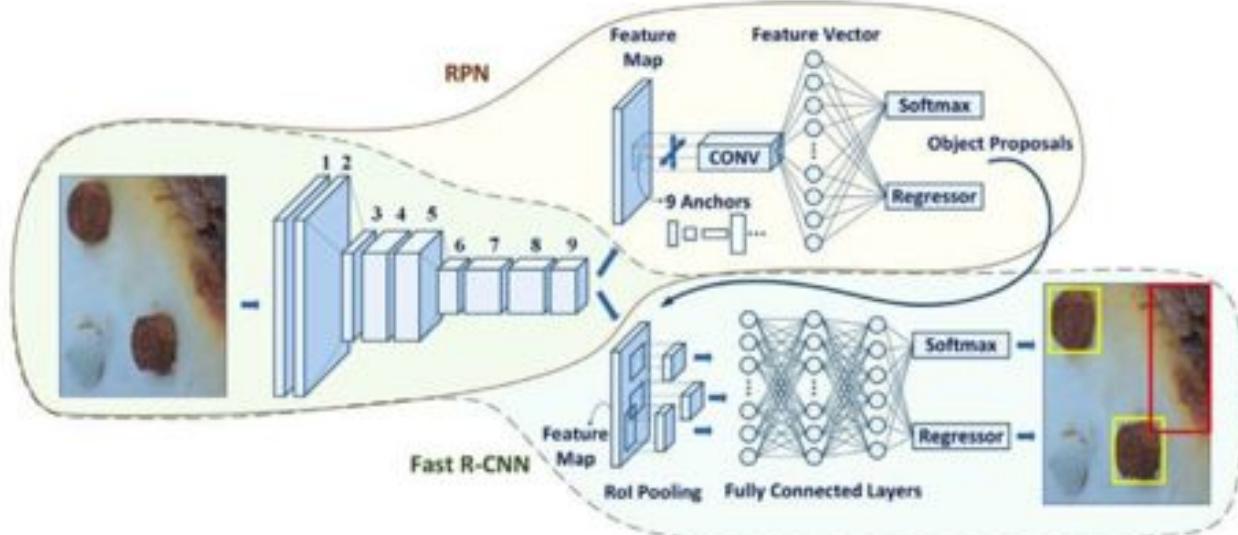
Evaluation of EfficientDet

| Model | APval | mAP@0.5 | RMSE | rRMSE | APtest | R2 |
|--------------|-------|---------|------|-------|--------|------|
| YOLO | 0.75 | 0.86 | 4.33 | 10.3% | 0.70 | 0.96 |
| EfficientDet | 0.72 | 0.81 | 3.44 | 10.6% | 0.63 | 0.97 |



Faster R-CNN Architecture

- Obtain from image:
 - list of bounding boxes
 - label assigned to each bounding box
 - probability for each label and bounding box



Faster R-CNN Process

- Input images are passed through a pre-trained CNN up until an intermediate layer (Transfer Learning)
 - End with a convolutional feature map
- 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
- Region of Interest (RoI) Pooling - Extract previous features which would correspond to the relevant objects into a new tensor
- R-CNN module
 - Classifies the content in the bounding box (or discard it, using “background” as a label)
 - Adjust the bounding box coordinates (so it better fits the object)

Faster R-CNN Hyperparameter Tuning

- DataLoader
 - Batch size
- Optimizer
 - SDG
 - Learning Rate
 - Momentum
 - Weight Decay
 - Nesterov
 - Adam
 - Learning Rate
 - Weight Decay

Faster R-CNN Results (SGD)

```
optimizer = torch.optim.SGD(params, lr=0.0001, momentum=0.95, weight_decay=0.0005,  
nesterov=True)  
num_epochs = 10
```

Epoch: 5 Training Loss: 0.631314

Validation IOU: 0.6625

Epoch: 6 Training Loss: 0.628712

Validation precision increased(0.661849 --> **0.662546**). Saving model ...

Iteration #300 loss: 0.6001112461090088

Validation IOU: 0.6612

Epoch: 7 Training Loss: 0.624271

Iteration #360 loss: 0.5466078519821167

Validation IOU: 0.6604

Epoch: 8 Training Loss: 0.619354

Validation IOU: 0.6576

Epoch: 9 Training Loss: 0.619932

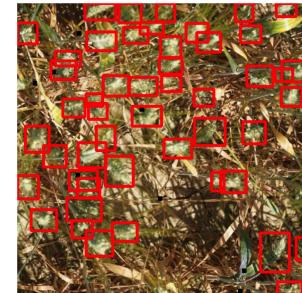
END-----



Faster R-CNN Results (Adam)

```
optimizer = torch.optim.Adam(params, lr=0.0001, betas=(0.9, 0.999), eps=1e-08,  
weight_decay=0.0005, amsgrad=False)  
num_epochs = 10
```

Epoch: 5 Training Loss: 0.656384
Validation IOU: 0.7188
Epoch: 6 Training Loss: 0.647585
Validation precision increased(0.635841 --> 0.718781). Saving model ...
Iteration #300 loss: 0.6084125638008118
Validation IOU: 0.7061
Epoch: 7 Training Loss: 0.641469
Iteration #360 loss: 0.6902428269386292
Validation IOU: 0.6905
Epoch: 8 Training Loss: 0.635929
Validation IOU: 0.6913
Epoch: 9 Training Loss: 0.632648



Faster R-CNN SGD vs Adam

10 Epochs

| Optimizer | Learning Rate | Momentum | Weight Decay | Betas | Epsilon | Precision |
|-----------|---------------|----------|--------------|--------------|---------|-----------|
| SGD | .001 | .005 | .0005 | | | .672453 |
| SGD | .0001 | .95 | .0005 | | | .672577 |
| SGD | .0001 | .0005 | .0005 | | | .672613 |
| Adam | .0001 | | .0005 | (0.9, 0.999) | 1e-08 | .718781 |
| Adam | .001 | | .0005 | (0.9, 0.999) | 1e-08 | .327536 |

Faster R-CNN SGD vs Adam

50 Epochs

| Optimizer | Learning Rate | Momentum | Weight Decay | Betas | Epsilon | Precision |
|-----------|---------------|----------|--------------|--------------|---------|-----------|
| SGD | .0001 | .005 | .0005 | | | .748047 |
| Adam | .0001 | | .0005 | (0.9, 0.999) | 1e-08 | .761707 |

Faster R-CNN Observations

- Adam Optimizer performing better than SGD over a smaller number of Epochs
- Over 50 epochs, Adam still performed better but SGD was comparable
- CoarseDropout increased overall precision after 5 epochs (most runs)
 - Without it, precision hovers around .66-.68
- Resize at 512x512 didn't perform as well as 1024x1024

Conclusion

In this study three types of CNN architectures were investigated:

- (i) YOLO, (ii) EfficientDet, and (iii) Faster R-CNN.

YOLO got the best performance with AP of 0.75, mAP@0.5 of 0.86, and rMSE of 10%.