Final Project Report

Global Wheat Detection

Yoel Bokobza, Oded George, and Ziv Shahar

11 August 2022

Contents

1 Abstract

In this work, we detect wheat heads from outdoor images of wheat plants, including wheat dataset from around the globe. The research problem can be formulated as an object detection problem, where our mission is to predict bounding boxes (BBOXes) around each wheat head (if exist). Our study starts with motivation and introduction. Then we perform an extensive Exploratory Data Analysis (EDA) including finding BBOXes size outliers, exploring the brightness distribution, the effect of the location from which are the images taken, etc. We've conducted many experiments with Faster R-CNN concluding training and validation losses figures to evaluate the effect of different optimizers, learning rates, $l2$ regularization, losses, and model combinations via Weighed Boxes Fusion (WBF). We've found out that using random flip augmentation can dramatically reduce the overfit phenomena. We adopted an object detection ensembled method called WBF to combine 4 different Faster R-CNN models and achieved a private score of 0.5840 on Kaggle.

2 Introduction & Objective

Wheat can be processed into flour for staple foods, and snacks, or fermented into alcohol or biofuel. Wheat's most common growth stages are the green, jointing, heading, filling, and maturity stages. The growth and health statuses of the wheat head have a significant impact on wheat yield and quality from the heading stage to the maturity stage. More specifically, spikes number per unit of ground area are one of wheat production's most critical agronomic factors. Based on this feature, real-time evaluation can assist in monitoring wheat growth, making management strategies, and then provide an early prediction of the wheat yield.

High-precision wheat head recognition is essential for extracting wheat head features and automatically detecting wheat phenotype. Computer vision and deep learning technologies have advanced to the point where the number of wheat heads can theoretically be measured automatically and accurately. Nonetheless, identifying wheat heads via machine vision technology is a complicated and tricky task with multiple obstacles.

In this competition, we will detect wheat heads from outdoor images of wheat plants, including wheat datasets from around the globe. Using worldwide data, we will focus on a generalized solution to estimate the number and size of wheat heads. To better gauge the performance for unseen genotypes, environments, and observational conditions, the training dataset covers multiple regions ('usask_1', 'arvalis_1', 'inrae_1', 'ethz_1', 'arvalis_3','rres_1', 'arvalis_2'). We suggest using a computer vision algorithm for object detection. In computer vision, object detection is the task of finding objects of interest in an image by marking them with a bounding boxes (BBOX), and also classifing the

object within each BBOX. In our task, we want the algorithm to identify the wheat heads by creating a tight BBOX around each wheat head. This allows us to estimate the number of wheat heads by simply counting the number of BBOX in the image, and also the size of each wheat head by calculating the area of any BBOX. In this work we used an ensembling method for object detection of 4 Faster R-CNN models [1].

3 Data Description

The Global Wheat Head Dataset is led by nine research institutes from seven countries. These institutions are joined by many in their pursuit of accurate wheat head detection, including the Global Institute for Food Security, DigitAg, Kubota, and Hiphen. The data is images of wheat fields, with bounding boxes for each identified wheat head. Not all images include wheat heads BBOXes. The images were recorded in many locations around the world. The dataset contains 3422 unique train images and only 10 test images. All of the images in our dataset are of the same size 1024×1024 with 3 channels. The CSV file that Kaggle provide us include image_id which is the unique image ID, width & height which is width and height of the images, and a bounding box, formatted as a Python-style list of [x_min, y_min, width, height] indicating the bottom left corner, width and height of the bounding box surrounding each wheat head, and the corresponding source that belong to ['usask_1', 'arvalis_1', 'inrae_1', 'ethz_1', 'arvalis_3', 'rres_1', 'arvalis_2'].

	image_id	width	height	bbox	source
0	b6ab77fd7	1024	1024	[834.0, 222.0, 56.0, 36.0]	usask_1
1	b6ab77fd7	1024	1024	[226.0, 548.0, 130.0, 58.0]	usask_1
2	b6ab77fd7	1024	1024	[377.0, 504.0, 74.0, 160.0]	usask_1
3	b6ab77fd7	1024	1024	[834.0, 95.0, 109.0, 107.0]	usask_1
4	b6ab77fd7	1024	1024	[26.0, 144.0, 124.0, 117.0]	usask_1

Figure 1: CSV dataset

For the Exploratory Data Analysis (EDA) we first investigate the amount of BBOXes per image which can provide insights into the density of wheat heads in the images.

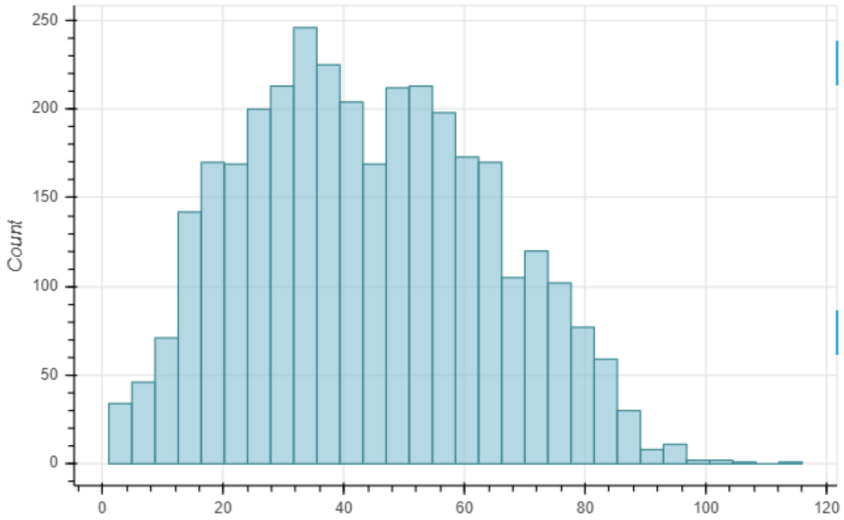


Figure 2: Number of wheat spikes per image histogram

From Fig. 2, we can infer that most of the images contains 20-60 wheat spikes. In addition we can infer that there are no severe outliers corresponding to the number of wheat-heads in an image. Figs. 3, 4 present few examples of images with a small amount of spikes, and with many spikes, respectively.

Figure 3: Images with a small number of spikes

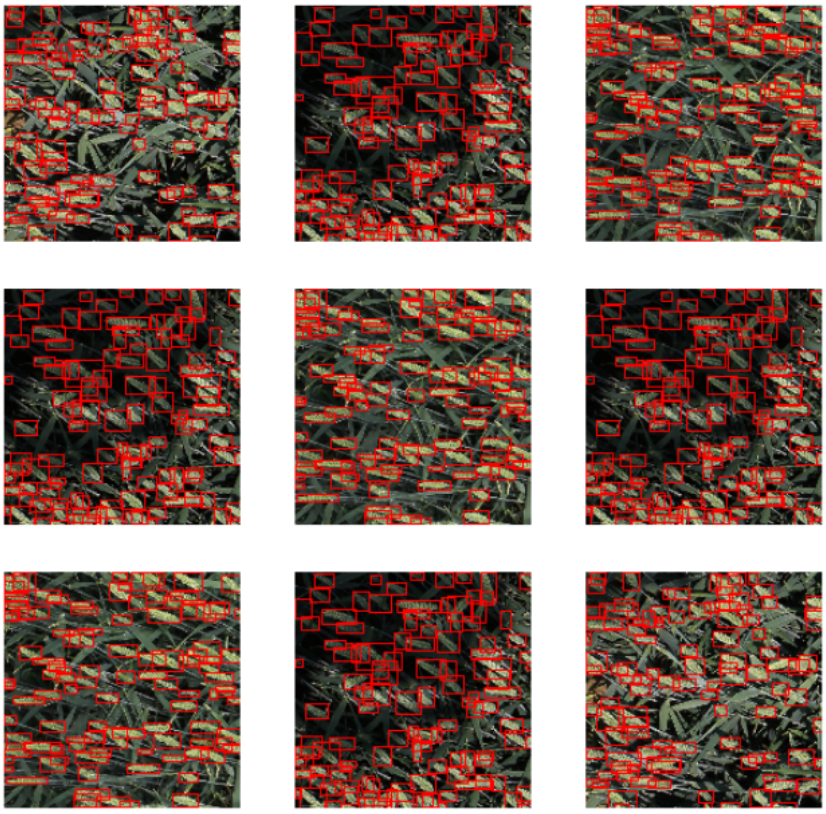


Figure 4: Images with a large number of spikes

Let's also investigate the images source distribution.

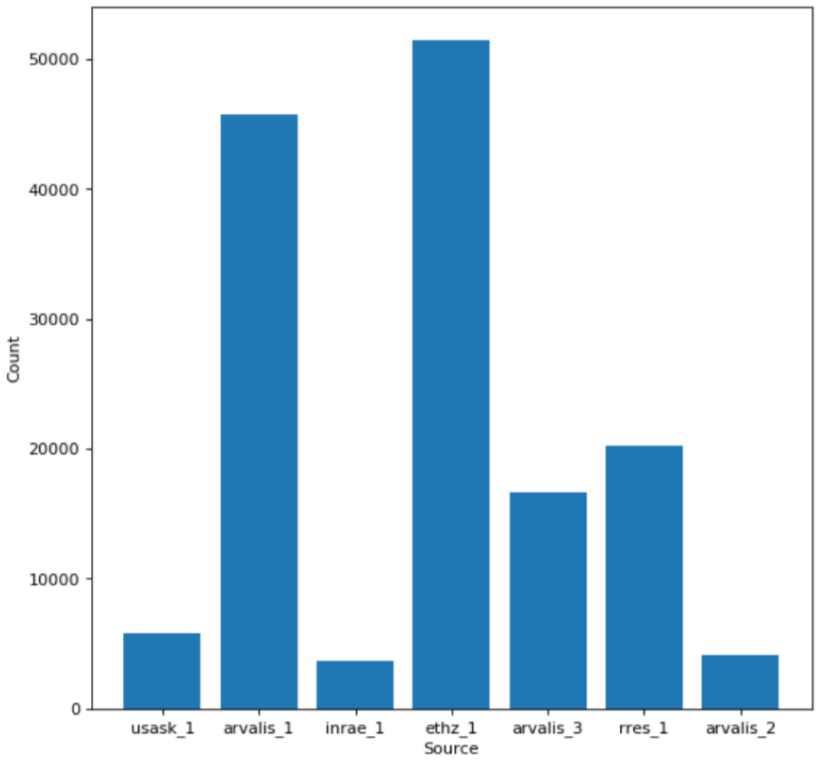


Figure 5: Image source histogram

From Fig. 5 we can observe that most of the images in the dataset are from sources 'arvalis_1', 'ethz_1', 'arvalis_3', and 'rres_1'. This observation may help us with the selection of the augmentations that we will perform over our data. For example, if most of the images from sources 'arvalis_1', 'ethz_1', 'arvalis_3', and 'rres_1' will contain dark images, then we will know that applying a brightness augmentation may help to prevent overfitting the dark images. Therefore, we would like to view several randomly sampled images from each source:

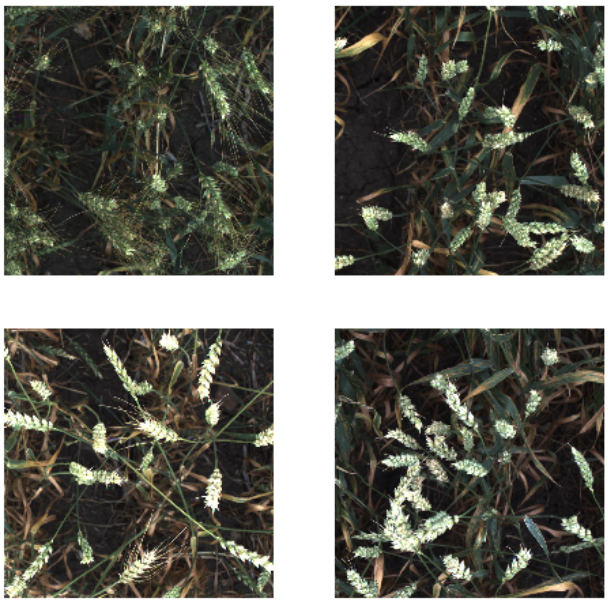


Figure 6: Image examples from source usask_1

Figure 7: Image examples from source arvalis_1

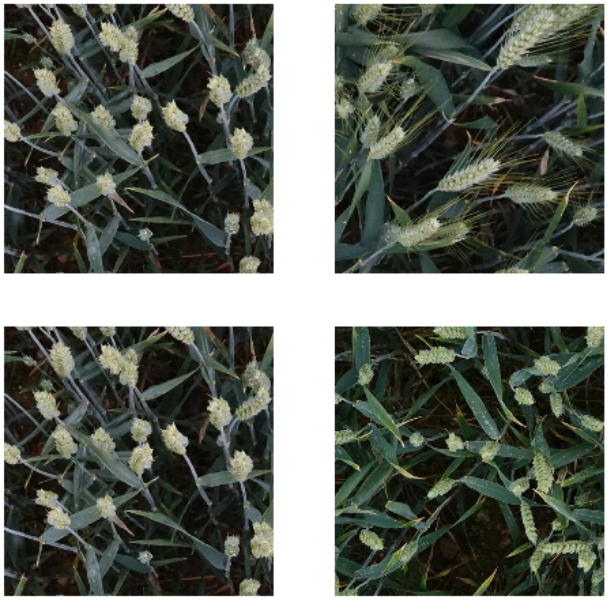


Figure 8: Image examples from source inrae_1

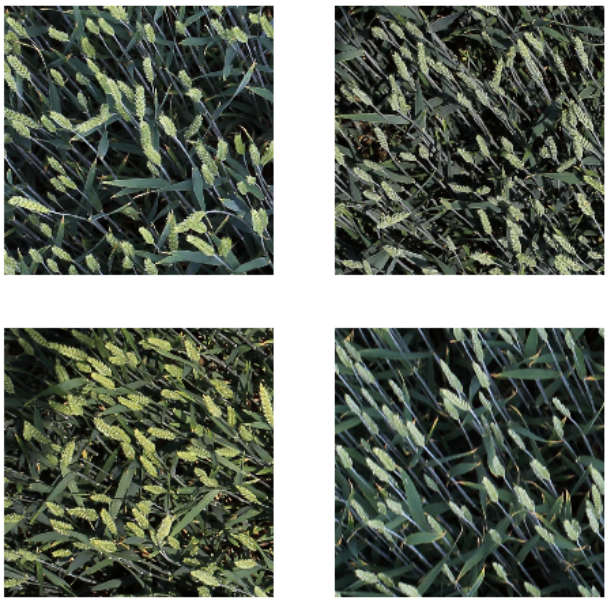


Figure 9: Image examples from source ethz_1

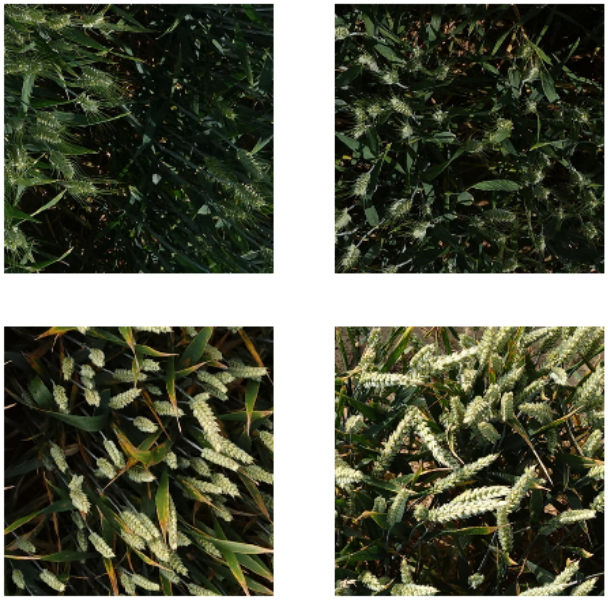


Figure 10: Image examples from source arvalis_3

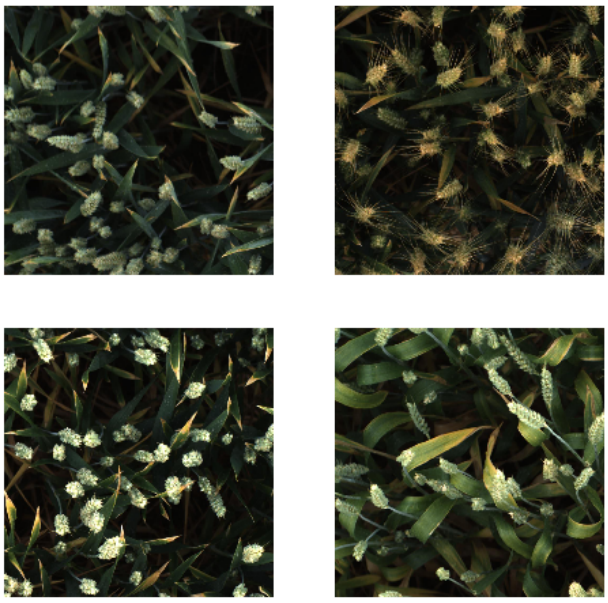


Figure 11: Image examples from source rres_1

Figure 12: Image examples from source arvalis_2

We can see that the majority of the images are darker. Therefore, we may need to

apply some kind of brightness augmentation. We also plot the histogram for images brightness, illustrated in Fig. 13

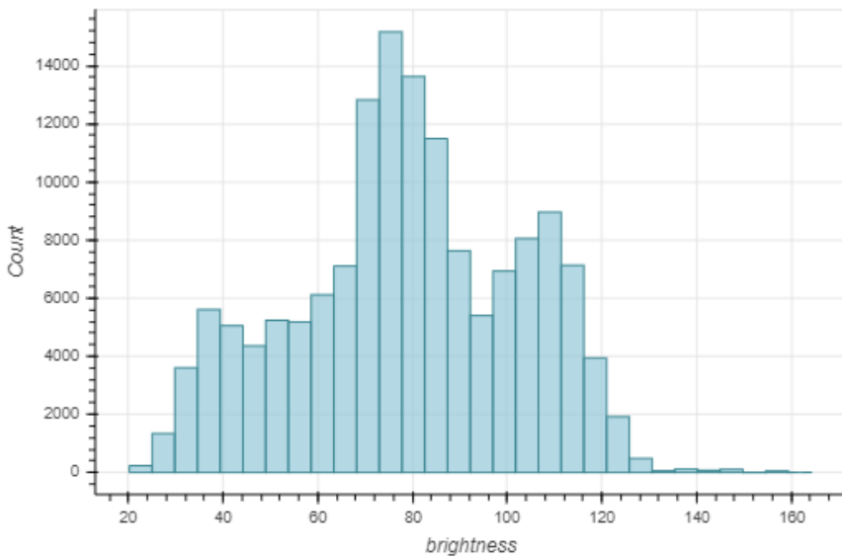


Figure 13: Histogram of images brightness

We can see from Fig. 13, that the amount of images with a dark background is greater than the number of images with a bright background therefore as we've said earlier, a brightness augmentation may increase the variance of the images in the dataset.

Now, let's check whether there are labeling errors in the data, and how we can handle them.

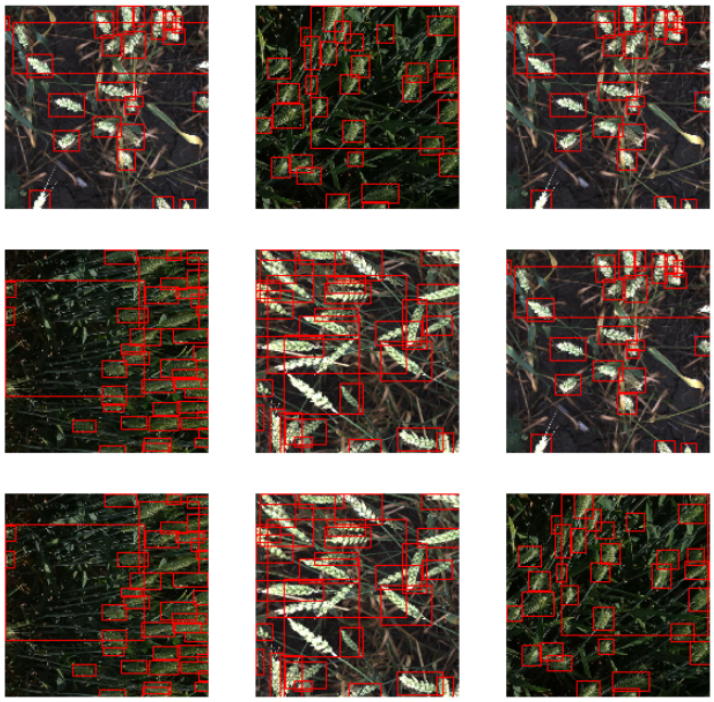


Figure 14: Large BBOXes examples

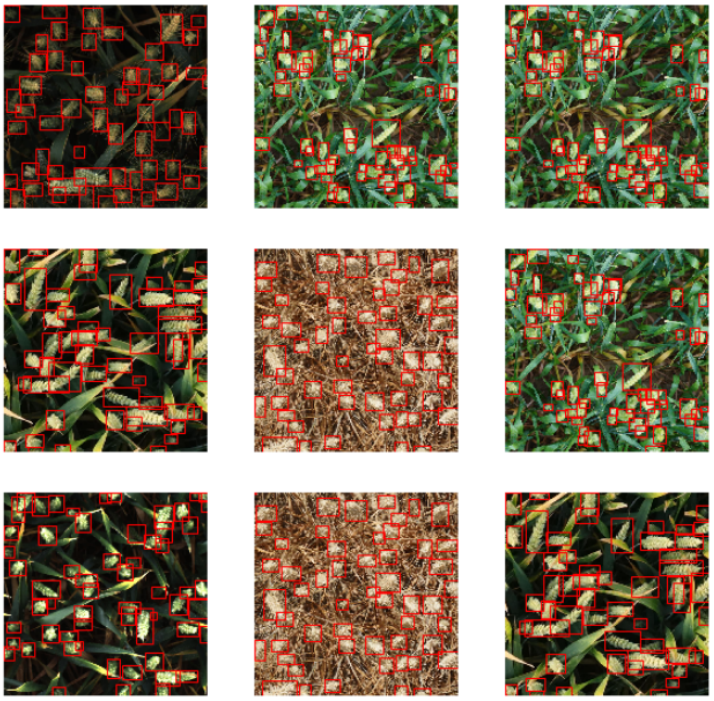


Figure 15: Small BBOXes examples

In Fig. 14, there are few labeled BBOXes which are incorrectly too large, and in Fig. 15, there are images with a very small BBOX. In order to deal with this problem, we define two thresholds, $T_{small} = 30$, and $T_{big} = 150000$. A BBOXes with area which is larger than T_{big}, or smaller than T_{small}, are removed.

4 Methods & Algorithms

In this work, we've used different configurations of Faster R-CNN, e.g, different augmentations, learning rates, losses, optimizers, regularizes, ensembling of different models, etc. When we examined ensembling models, we've used weighed boxes fusion (WBF) method for object detection models combinations [2]. In the following subsections, we will detail these suggested techniques.

4.1 Augmentations

The quantity and diversity of data are important factors in the effectiveness of most machine learning models. The amount and diversity of data supplied during training heavily influence the prediction accuracy of these models. Data augmentation can be utilized to address both requirements, the amount of data and the diversity of the training data needed to create an accurate machine learning model. Data augmentation is a set of techniques used to increase the amount of data in a machine learning model by adding

slightly modified copies of already existing data or newly created synthetic data from existing data. There are several benefits of data augmentation:

- A machine learning model performs better and is more accurate when the dataset is rich and comprehensive. By creating varied instances to train datasets, data augmentation can help improve the performance and results of machine learning models.

- Data collection and labeling can be time-consuming and costly for machine learning models. These operational costs can be removed by transforming datasets using data augmentation techniques.

- It helps smooth out the machine learning model and reduce the overfitting of data.

Some techniques used to augment images are Geometric transformations, Flipping, Cropping, Rotation, Translation, Brightness, etc. The first augmentation that we've used is the flipping augmentation, i.e., each image is flipped horizontally, or/and vertically with probability 0.5. Flipping the images allows the model to be able to detect wheat heads with different orientations. In addition, as shown in the EDA section, most of the images are darker and therefore we will try to apply brightness augmentation to prevent overfitting. We've used the albumentation library to facilitate those augmentations. In the brightness augmentation, we randomly change the brightness of the input image according to two predefined parameters. One defines the factor range for changing brightness ($[-0.2, 0.2]$) and the second defines the probability for applying the transform ($p = 0.5$). Our augmentations are applied online, meaning that each image is randomly transformed before being fed into the network. This method facilitates providing overfitting, as the data set observed by the model is not fixed.

Figs. 16, 17, illustrate the flip and brightness augmentations, respectively applied over some images from the dataset.

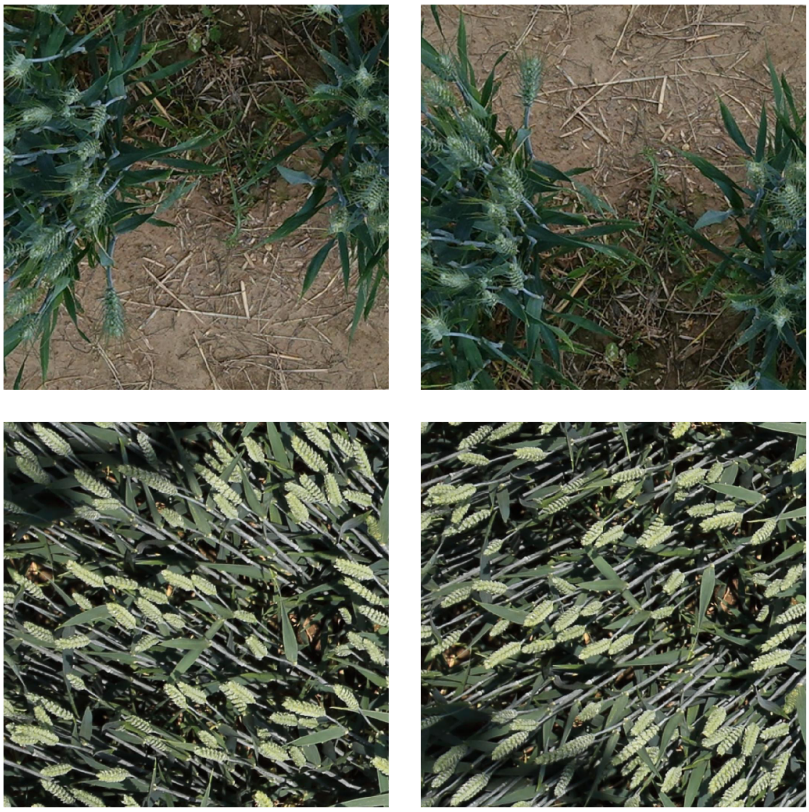


Figure 16: On the left side: Original images. On the right side: the corresponded flipped image

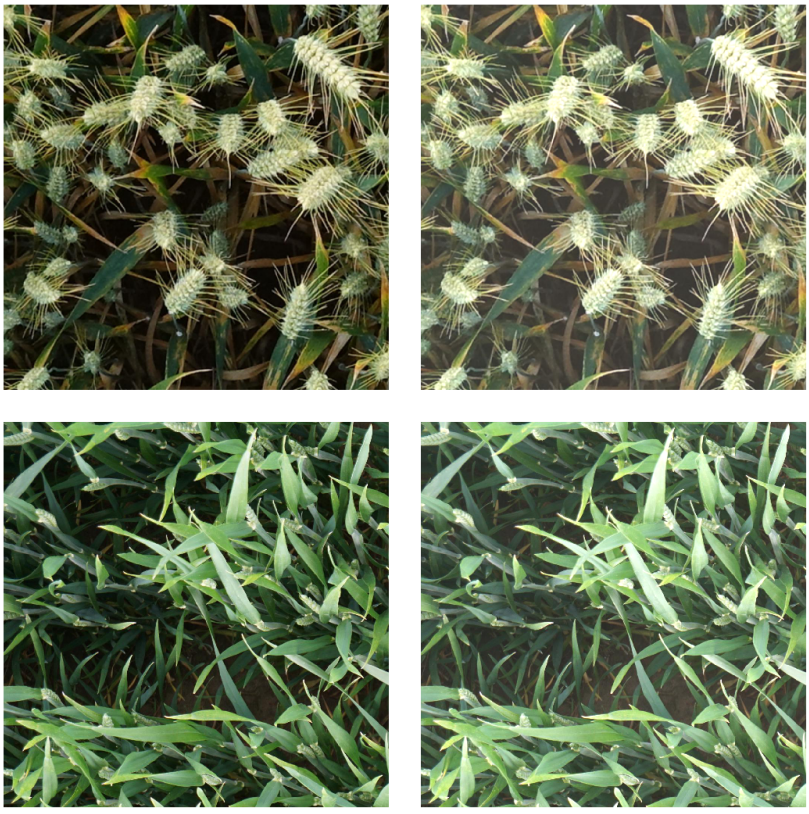


Figure 17: On the left side: Original images. On the right side: the corresponded image after applying the brightness augmentation

4.2 Faster R-CNN

The Faster R-CNN model developed by a group of researchers at Microsoft at 2015. Faster R-CNN is a deep convolutional network used for both object detection and classification, that appears to the user as a single, end-to-end, unified network. The network can accurately and quickly predict the locations of different objects. The Faster R-CNN is an evolution of Fast R-CNN [3], which is an evolution of earlier algorithm called R-CNN [4]. In order to deeply understand the mechanism behind the Faster R-CNN, we have to briefly review the Fast R-CNN and the R-CNN.

R-CNN is considered one of the first deep learning algorithms used for object detection. The idea is to use a classical algorithm as a selective search [5], for suggesting areas in the image which are suspected to contain objects. Then, each cropped region from the original image has been fed into Convolutional Neural Network (CNN) which is used to extract features from the region. These features are fed into an support vector machine (SVM) to classify if there is a presence of an object, and if there is an object, then it also predicts its class. This is also fed into a BBOX regressor to predict the offset of the proposed region to increase the precision of the predicted BBOX. The main disadvantage of this algorithm is that it is a very slow algorithm for real-time applications. The average inference time for a single image is about 47 seconds. Another disadvantage is that algorithm is not a fully-learned algorithm, as it uses the selective search algorithm, which can lead to sub-optimal performance. The large inference time of the R-CNN has motivates the development of the Fast R-CNN.

In Fast R-CNN, instead of passing each proposal region to the CNN (usually about 2000 proposal regions) as in R-CNN, the full image is passed through a fully-convolutional network (FCN) once in order to create the feature map, and then each proposed region is projected into the feature map. The projected proposed region is referred to as region of interest (RoI). This is clearly faster than feed each proposed region into a FCN. Each RoI has fed into the RoI-Pool layer to extract a fixed-length feature vector from each RoIs. The extracted feature vector using the RoI Pooling is then passed to a fully-connected (FC) layers. The output of the last FC layer is split into 2 branches: Softmax layer to predict the class scores, and FC layer to predict the bounding boxes of the detected objects, see Fig. 18.

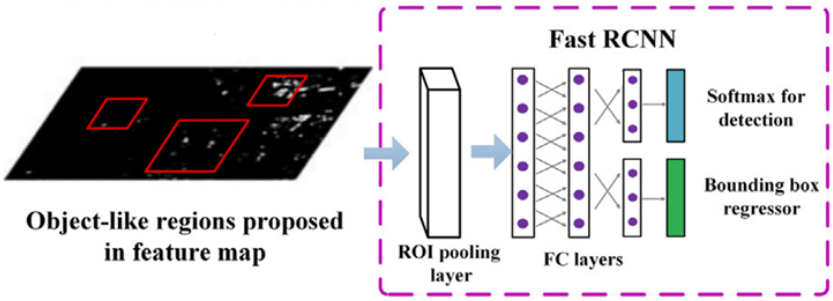


Figure 18: Fast R-CNN illustration

The main contribution of the Faster R-CNN is the introduction of the region proposal network (RPN) which is a FCN that generates proposals with various scales and aspect ratios. Another advantage is that the convolutional computations are shared across the RPN and the Fast R-CNN. This reduces the computational time. The RPN works on the output feature map returned from the last convolutional layer shared with the Fast R-CNN. A sliding window of size $n \times n$ passes through the feature map. For each window, k region proposals are generated. Each proposal is parametrized according to a reference box which is called an anchor box. The 2 parameters of the anchor boxes are scale, and aspect ratio. The k regions are produced from each $n \times n$ patch in the feature map by back projecting this patch into the original image and then performing scaling and changing the aspect ratio according to a set of k predefined transformations. The values from each $n \times n$ patch have fed into the cls layer and the reg layer which are FC networks. The cls layer represents a binary classifier that generates the objectness score per anchor for each region proposal (i.e. whether the region contains an object, or it is not). The reg layer returns k 4-D vectors encoding the coordinates offset of the k boxes. Fig. 19 illustrates the basic RPN architecture.

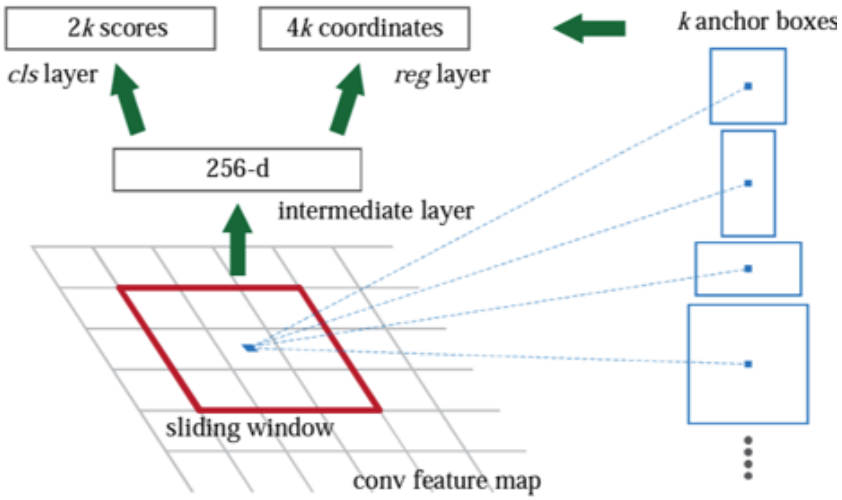


Figure 19: RPN

As some RPN proposals highly overlap with each other, to reduce redundancy, the

authors applied the non-maximum suppression (NMS) algorithm (see appendix 9.3) over the proposed regions based on their cls scores. The proposed regions (RoIs) after NMS have fed into the Fast R-CNN for both classification and fine-tuning the boxes coordinates for the suggested region.

The high-level architecture of the Faster R-CNN is illustrated in Fig. 20.

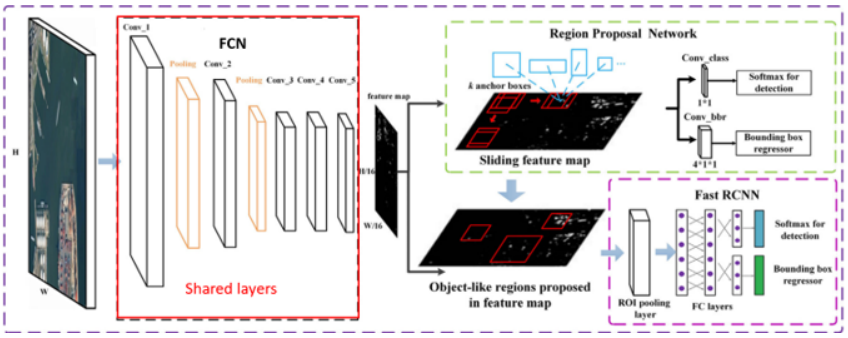


Figure 20: Faster R-CNN architecture

Note that in Fig. 20, the FCN layer is used for both the RPN and the Fast R-CNN.

4.2.1 The Loss Functions

First, let's define the RPN loss function. Let p_i, denote the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive, i.e, its intersection over union (IoU) with a ground truth BBOX is larger than a predefined threshold and is 0 otherwise (negative anchor). For further details about IoU, refer to the appendix 9.1. Correspondingly, t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the corresponded ground-truth box. The loss function for a mini batch is defined as:

$$L^{RPN}(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}^{RPN}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^*\cdot L_{reg}^{RPN}(t_i,t_i^*),$$

where L_{cls}^{RPN} is the binary cross entropy loss, and L_{reg}^{RPN} is the smooth-L1 loss [3]. N_{cls} is equal to the batch size, and N_{reg} is equal to batch size times the number of anchors per region, k. λ is a hyperparameter which is used to balance between the two losses. Note that the L_{reg}^{RPN} loss is multiplied by p_i^*, as we do not want the loss to punish for BBOX misalignment when the anchor is not containing an object.

Regarding the loss of the Fast R-CNN, we define $pp = (pp_0, pp_1, ..., pp_M)$ as a discrete probability distribution (per RoI), over $M+1$ categories, where the $0'th$ category corresponding to no-object. Let pp^* be the indicator of the ground truth class and also let tt^k, and v denote the 4 parameterized coordinates of the predicted bounding box for the $k'th$ class, and the associated ground-truth bounding box, respectively. Note that we predict M BBOX coordinates, one bbox for each class. This decoupled structure allows

the network to generate BBOX coordinates predictions per class without competition among classes. The loss for Fast R-CNN for a specific RoI , defined as

$$L^{FAST}(pp, u, tt^u, v) = L_{cls}^{FAST}(pp, u) + \gamma \mathbb{1}(u \neq 0) \cdot L_{reg}^{FAST}(tt^u, v),$$

where L_{cls}^{FAST} is the multi-class cross entropy loss, and L_{reg}^{FAST} is the smooth-L1 loss. Note that when we calculate the L_{reg}^{FAST} loss we taking into account the box coordinates prediction corresponding to the ground truth class (u), and similarly to the RPN loss, when there is no object in the RoI, i.e., $u = 0$, then the right summand is 0. In addition, γ is a hyperparameter which is used to balance between L_{cls}^{FAST}, and L_{reg}^{FAST}. The total loss defined as

$$L^{TOT} = L^{FAST} + \alpha L^{RPN},$$

for $\alpha \geq 0$ which is used to balance between Fast R-CNN loss and the RPN loss. The network objective is to minimizes this loss function.

4.2.2 Our Faster R-CNN Basic Architecture

In the following sections, in all of the experiments we've used Faster R-CNNS with ResNet50 [6] as the FCN shared layers. As shown in [1], the performance are insensitive to the choice of λ, and γ, therefore we've used $\lambda = \gamma = 1$. We've used transfer learning [7] as our network is pretrained over the COCO dataset [8], and we've freezed the 3 first layers of the ResNet50 before retraining the network. These layer are freezed because they had been trained over the large dataset COCO (more than 200000 labeled images) which have 80 different labels. Thereby, they hopefully will be able to extract "good" features from the input images, while we still allow tuning the weights in the other layers. As we've used transfer learning of a model which had been trained over the COCO data set (in which the number of classes is 80), we've replaced the last classification layer of the Fast R-CNN with a layer of 2 output neurons, because in this task we have two classes: One for wheat, and the other for no wheat. The network is trained end-to-end with the L^{TOT} loss defined previously. The Fast R-CNN FC layers are composed of two 1024 FC layers, and their output is fed into two different FC layers. One for classification with output size 2 followed by softmax, and the other for the BBOX regression with output size 4. The anchor sizes are 32, 64, 128, 256, 512, and the aspect ratios are 0.5, 1, and 2, therefore the number of anchors is $k = 15$. The rectangular window size of the RPN is 3×3.

4.3 Weighted Boxes Fusion

WBF is used for ensembling different object detection models. Unlike NMS, WBF uses confidence scores of all proposed bounding boxes to construct the weighed average boxes.

According to [2], this method significantly improves the quality of the combined predicted BBOXes, compare with NMS. The WBF algorithm steps are the followed:

1. Each predicted box from each model is added to a single list B. The list is sorted in decreasing order of the confidence scores C.

2. Declare empty lists L and F for boxes clusters and fused boxes. Each position in the list L can contain a set of boxes, which form a cluster; each position in F contains only one box, which is the fused box from the corresponding cluster in L.

3. Iterate through predicted boxes in B in a cycle and try to find a matching box in the list F. The match is defined as a box with a large overlap with the box under question ($IoU > thr$).

4. If the match is not found, add the box from the list B to the end of lists L and F as new entries; proceed to the next box in the list B.

5. Recalculate the box coordinates and confidence score in $F[pos]$, using all T boxes accumulated in cluster $L[pos]$, with the following fusion formulas:
$C = \frac{\sum_{i=1}^{T} C_i}{T}$, $X1,2 = \frac{\sum_{i=1}^{T} C_i \cdot X1,2_i}{\sum_{i=1}^{T} C_i}$, $Y1,2 = \frac{\sum_{i=1}^{T} C_i \cdot Y1,2_i}{\sum_{i=1}^{T} C_i}$

6. After all boxes in B are processed, re-scale confidence scores in F list:
$C \leftarrow C \cdot \frac{T}{N}$

4.4 Ensembled Model

Neural network models are nonlinear and have a high variance, which can be frustrating when preparing a final model for making predictions. Ensemble learning combines the predictions from multiple neural network models to reduce the variance of predictions and reduce generalization error. To ensemble Faster R-CNN models, we've used the WBF algorithm described in Section 4.3. The Model architecture is illustrated in Fig. 21

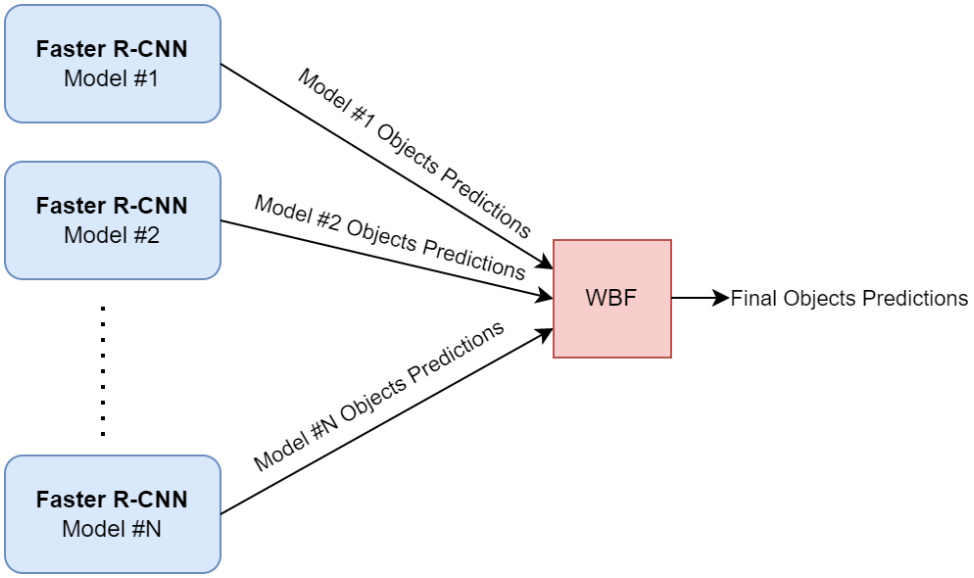


Figure 21: Our proposed ensembled architecture, with N different models ensembled via WBF

In the WBF algorithm we've used the IoU threshold $\mathbf{thr} = \mathbf{0.55}$ for considering boxes as a match. After applying the WBF algorithm, we were removed bounding boxes with confidence score smaller than 0.7.

4.5 Learning Rate Scheduler

Learning rate schedule is a method used to adapt the learning rate online during model training [9]. Learning rate schedule was shown to be a very efficient method to improve model performance as it allows the model to use a high learning rate at the beginning of training and then, gradually decrease it to allow the loss to be closer to a local optimum as the training proceeds. Note that this method is sometimes combined with a "self" adaptive learning rates methods such as Adam [10].

5 Challenges & Difficulties

- The main difficulty we have faced in this project stems from the fact that we don't have a GPU except for that provided by Kaggle which is limited in terms of RAM, and therefore our computational resources are limited. Another difficulty is the time limitation that Kaggle has allocated for us. The running time over GPU for submission is limited to 6 hours per run. The overall time limitation for running a notebook over GPU is 40 hours per week. To cope with it, we opened different users in Kaggle and made our experiments over all of these notebooks parallelly. To cope with the RAM limitation, we limited our batch size to 16 while training.

- There was a problem with the labeled data as some of the ground truth BBOXes are mistakenly too large or too small, as illustrated in Figs. 14, 15. In Section 3, we proposed a simple solution for this problem.

- The size of the dataset is small, thereby to increase the generalization of the model and reduce overfitting, we applied random augmentations over any image in an online fashion, when loading a batch.

- When applying WBF over more than four models, the GPU is running out of memory. Therefore, in such cases, we reduced our test batch size from 4 to 1. This solved this problem, but it dramatically increased the inference time.

Identifying wheat heads via machine vision technology is a complicated and tricky task with multiple obstacles:

- Wheat heads vary dramatically in size, posture, shape, and texture depending on wheat varietals and growth stages. Take wheat heads as an example, their edges shapes are irregular, and some of their colors are similar to the leaves in particular growth stages.

- The automatic identification of wheat heads is significantly hampered in the diverse field environment due to mutual shielding between distinct wheat organs and the uneven and unstable natural illumination.

- Different growth environments for wheat also impact the effect of the detection model. Hence, a machine learning model that can detect wheat heads in various situations with solid generalizability is required.

Faster R-CNN is known as a very good model for object detection, which can well generalize to provide good results over unseen data. We've also used transfer learning for a model which had been pretrained over COCO dataset that contains many images with different classes, which allows the model to work well over images with different backgrounds, objects, brightness, etc. We also tried to apply brightness augmentation to cope with the fact that there is a variation in the brightness of the images in the dataset, as shown in the EDA section.

6 Experiments

In this section we will present the Faster R-CNN (presented in Section 4) performance with different hyper-parameters configurations, losses functions, and ensembles. The data has been split into 80% training and 20% validation after it was randomly shuffled. The loss used in the following experiments is the L^{TOT} loss defined in 4.2.1, and we've used

this loss for evaluating the train and validation performance. The training batch size in all of our experiments is 16. In some of the following experiments, we have used the step learning rate scheduler. Using this scheduler we've defined the initial learning rate to be constant for 15 epochs, and reduced it by 0.1 after any 15 epochs.
The performance on this Kaggle competition is evaluated via mean average precision (mAP) (see appendix 9.2). The results for each experiment had been evaluated via mAP and presented in Tables 1, 2, 3, 4

6.1 Augmentations

This section will examine how the augmentations influence the model's performance. In this sub-section, we've used the Adam optimizer, with learning rate $lr = 1e-4$, weight decay $5e-4$, and learning rate scheduler.

6.1.1 No Augmentations

Fig. 22 present the train and validation losses when no augmentation has been applied.

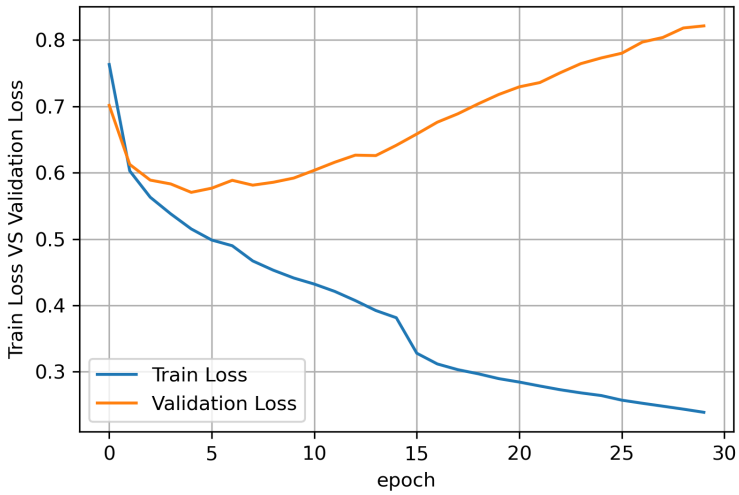


Figure 22: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler, and no augmentations

Note that the model is highly overfitted as expected because the training set is relatively small

6.1.2 Random Flip Augmentation

When applying the random flip augmentation, as described in Section 4.1, the performance are improved compared with the performance without applying any augmentation

as shown in Fig. 22. The train and validation losses when applying random flip augmentation are described in Fig. 23.

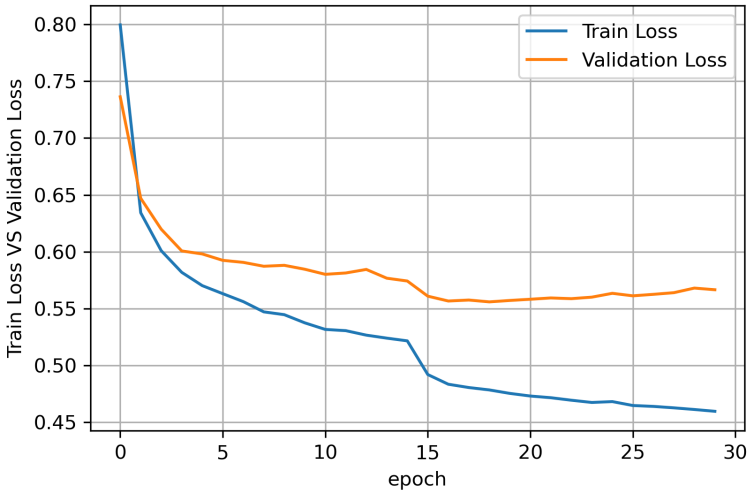


Figure 23: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler, and random flip augmentation

The performance here is much better due to the introduction of the online random flip augmentation which prevents the algorithm from overfitting.

6.1.3 Random Brightness, and Random Flip Augmentations

Here, we've also tried to apply random brightness augmentation. Fig. 24, demonstrates the train and validation losses for this scenario.

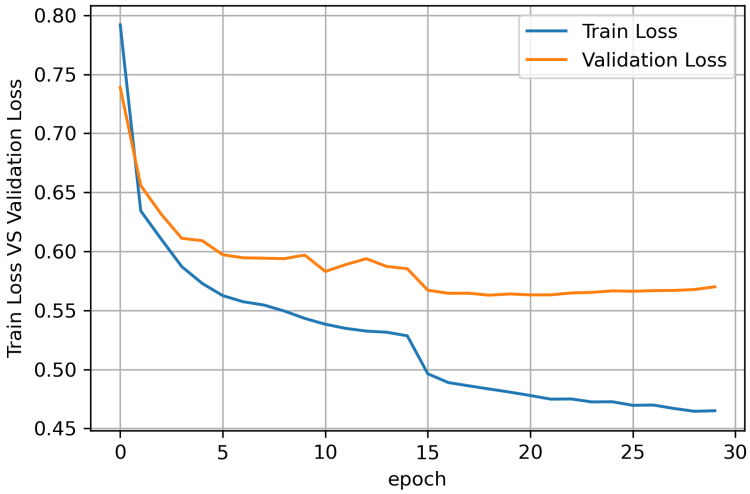


Figure 24: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler, random flip and random brightness augmentations

Surprisingly, the brightness augmentation didn't provide any performance improvements in terms of validation loss, compare with random flip augmentation only. One possible reason for that is that by applying random flip only, the model can well generalize. Another explanation is that this specific brightness augmentation with its specific parameters is not optimal. Unfortunately, due to the time limitation, and our limited resources we didn't manage to try other configurations of the brightness augmentation. In Table 1, we can see that also in terms of mAP, the random flip augmentation yielded the best performance. Therefore for the next experiments, we've used the random flip augmentation.

Table 1: Kaggle public and private final score for different augmentations

	Private Score	Public Score
No augmentations	0.5103	0.5914
Random flip augmentation	0.5823	0.6617
Random flip and brightness augmentations	0.5792	0.6415

6.2 Stochastic Gradient Descent with Momentum Optimizer

In the following experiments we've used stochastic gradient descent (SGD) with momentum 0.9.

Model#1: $lr = 5e-4$, weigh decay 0, no scheduler

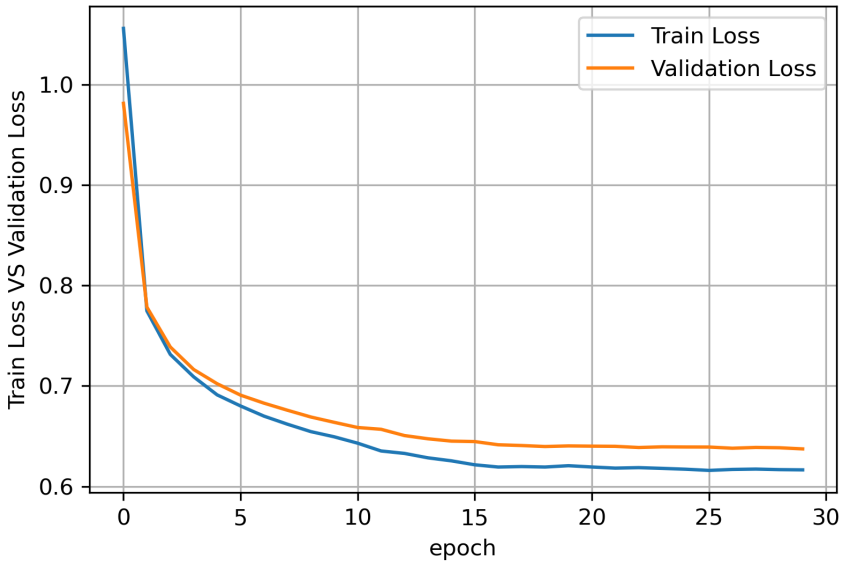


Figure 25: Model#1: SGD with momentum optimizer, $lr = 5e-4$, weigh decay 0, without lr scheduler

Model#2: $lr = 5e-3$, weigh decay $5e-4$, no scheduler

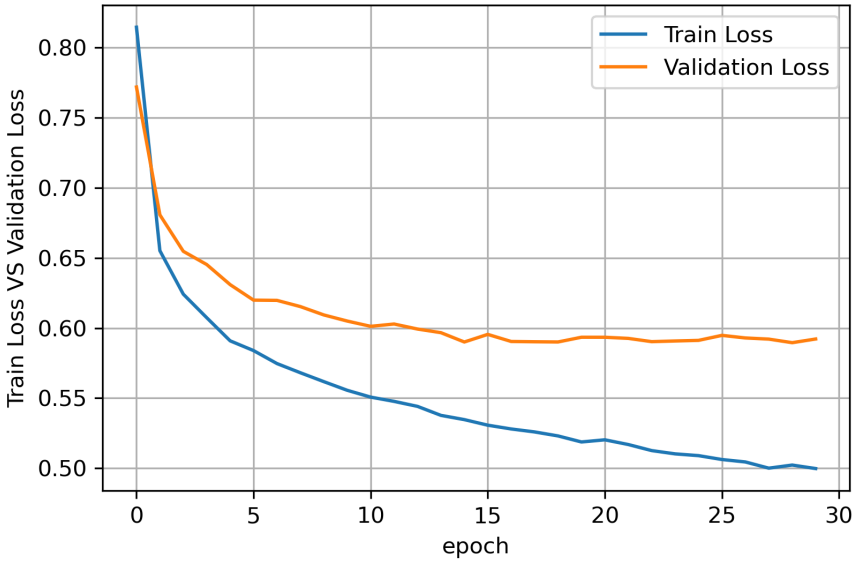


Figure 26: Model#2: SGD with momentum optimizer, $lr = 5e-3$, weigh decay $5e-4$, without lr scheduler

Model#3: $lr = 5e-3$, weigh decay $5e-4$, with scheduler

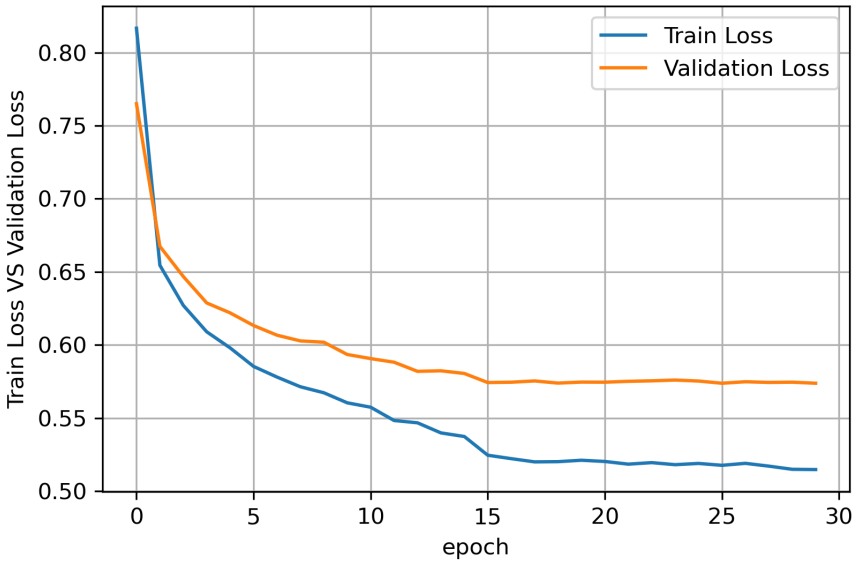


Figure 27: Model#3: SGD with momentum optimizer, $lr = 5e-3$, weigh decay $5e-4$, with lr scheduler

In Fig. 25, it seems that the training loss and the validation loss are really close and there is no overfit observed even though we didn't apply weight decay in this scenario.

Table 2: Kaggle public and private final score

	Private Score	Public Score
Model#1	0.5598	0.6279
Model#2	0.5744	0.6414
Model#3	0.5767	0.6490

In this case, the train and the validation loss started to saturate after approximately 15 epochs and stabilized on ~ 0.64. In Fig. 26, we've used a larger learning rate, $lr = 5e-3$. It is noted that both the train and validation losses are smaller compared with the case of $lr = 5e-4$ presented in Fig. 25. Note that in this case we've used weight decay of $5e-4$ to reduce overfitting, but we can still see a relatively large gap between the train and validation loss. To cope with the saturation of the validation loss after 15 epochs in both scenarios illustrated in Figs, 25 and 26, in Fig. 27, we've used the learning rate scheduling algorithm described on the begging of this section. In this scenario, after the $15'th$ epoch, the learning rate is reduced to $lr = 5e-4$, which seems to be not helpful, as the curves of both train loss and validation loss stay almost fix after the $15'th$ epoch. Nevertheless, in the following sub-section we will see that the learning rate schedule can be very helpful.

6.3 Adam optimizer

The Adam optimizer used in this sub-section is with $\beta_1 = 0.9, \beta_2 = 0.999$. Because our computational resources are limited, and as Adam is less sensitive to learning rate tuning as it is an adaptive algorithm, as described in [10], we've considered the same learning rate $lr = 1e-4$ in all of the following experiments.

Model#4: $lr = 1e-4$, weigh decay $5e-4$, without lr scheduler

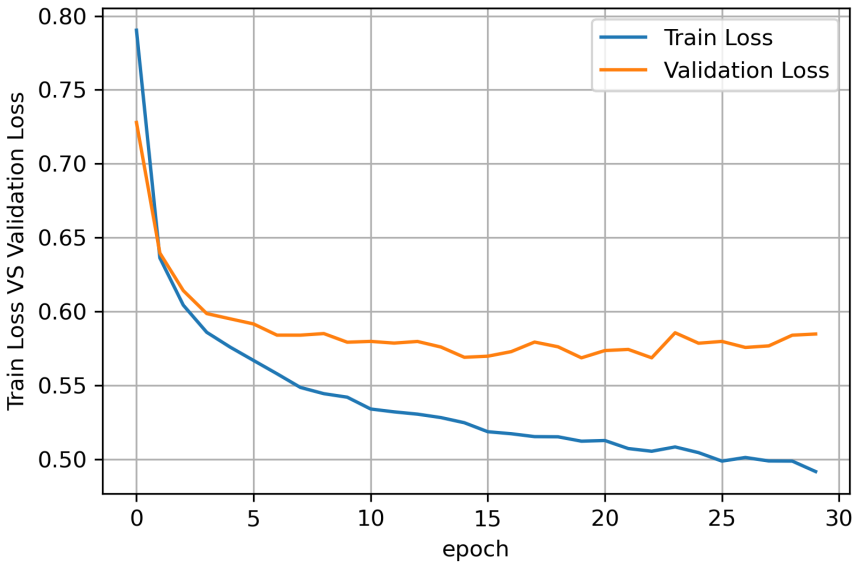


Figure 28: Model#4: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, without scheduler

Model#5: $lr = 1e-4$, weigh decay 0, with lr scheduler

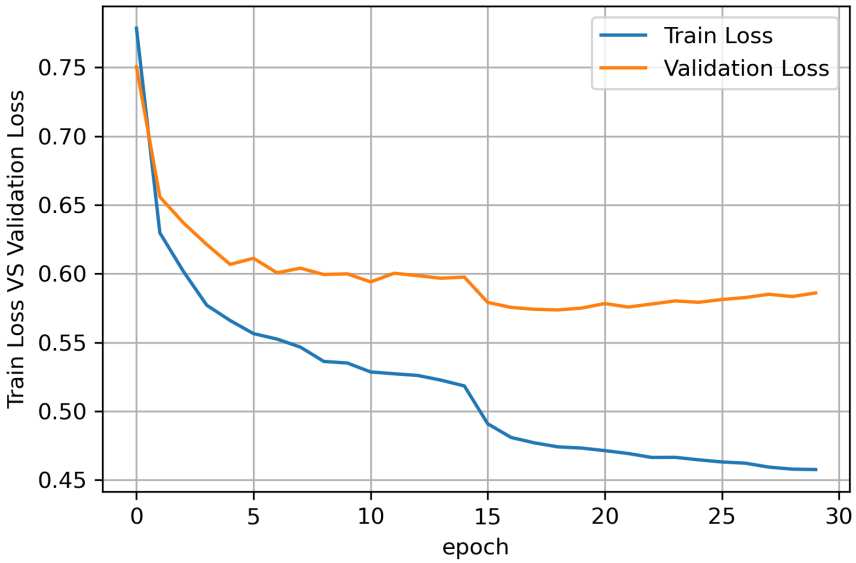


Figure 29: Model#5: Adam optimizer, $lr = 1e-4$, weigh decay 0, with scheduler

Model#6: $lr = 1e-4$, weigh decay $5e-4$, with lr scheduler

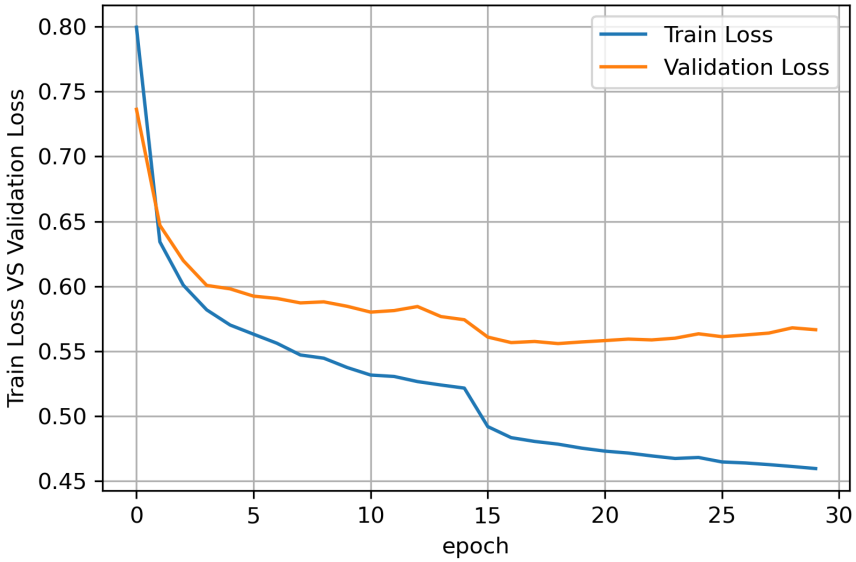


Figure 30: Model#6: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler

Table 3: Kaggle public and private final score

	Private Score	Public Score
Model#4	0.5751	0.6480
Model#5	0.5583	0.6173
Model#6	0.5823	0.6617

When comparing the results for Adam without scheduler (Fig. 28) with the performance of the SGD in 27, it seems that in both cases the same validation loss is achieved and the private and public score in both cases is similar. From Figs. 29, 30, it can be noticed that after approximately 15 epochs the losses begin to saturate but then, due to the learning rate scheduler, at the $15'th$ epoch, the train and the validation losses start to drop again and become smaller than the losses when no scheduling is applied. In addition, in contrast to Model#5 described in Fig. 29, in Fig. 30, a weight decay of $5e-4$ is applied and it has resulted in a smaller validation loss and a smaller gap between train and validation losses. The best private and public scores were achieved with Model#6 which also demonstrated the smallest validation loss.

In all of the previous experiments, we've used $\alpha = 1$ in the L^{TOT} loss defined in Section 4.2.1, meaning that we are assigning the same weight for both L^{RPN}, and L^{FAST} while training. This might be sub-optimal. During the training of the previous models, we've noticed that the L^{FAST} loss is much larger than the L^{RPN} loss. Therefore, we've tried

two things: The first one is to set $\alpha = 0.25$, to assign a smaller weight to the L^{RPN} loss (referred to as Model#7). The second is to set $\alpha = 1$ for 15 epochs, and then $\alpha = 0$ for the last 15 epochs (referred to as Model#8). The idea here is to stop the RPN training after 15 epochs and training only the Fast R-CNN because training them together might be problematic as the RoIs suggested for the Fast R-CNN are dependent on the RPN, so it might be better to stop training the RPN and let the Fast R-CNN training over a fixed RPN. In both of these scenarios, we used $lr = 1e-4$, weight decay $5e-4$, and learning rate scheduler. Note that in these scenarios evaluating the validation loss as the summation over the losses may be meaningless as the losses are weighted during training. Thereby, we plotted the 4 losses $L_{cls}^{FAST}, L_{reg}^{FAST}, L_{cls}^{RPN}, L_{reg}^{RPN}$ for both training and validation.

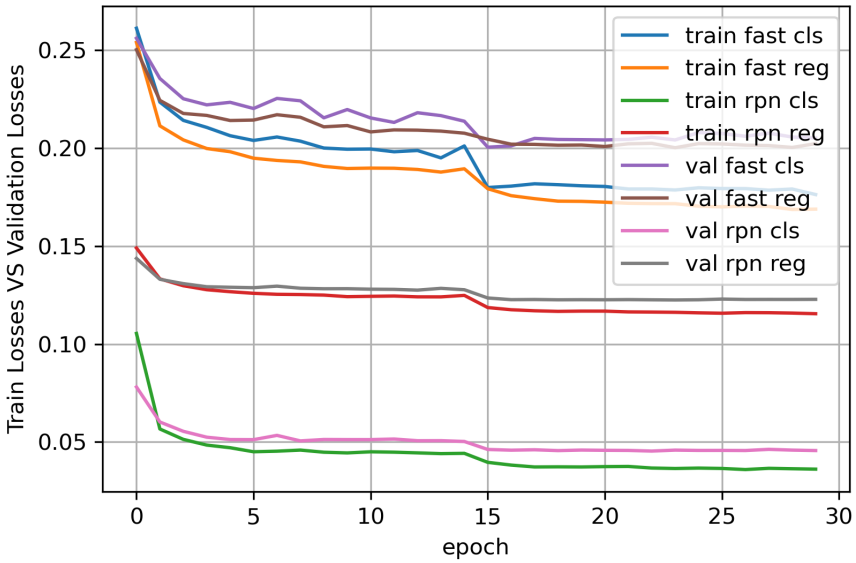


Figure 31: Model#7: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler, and $\alpha = 0.25$

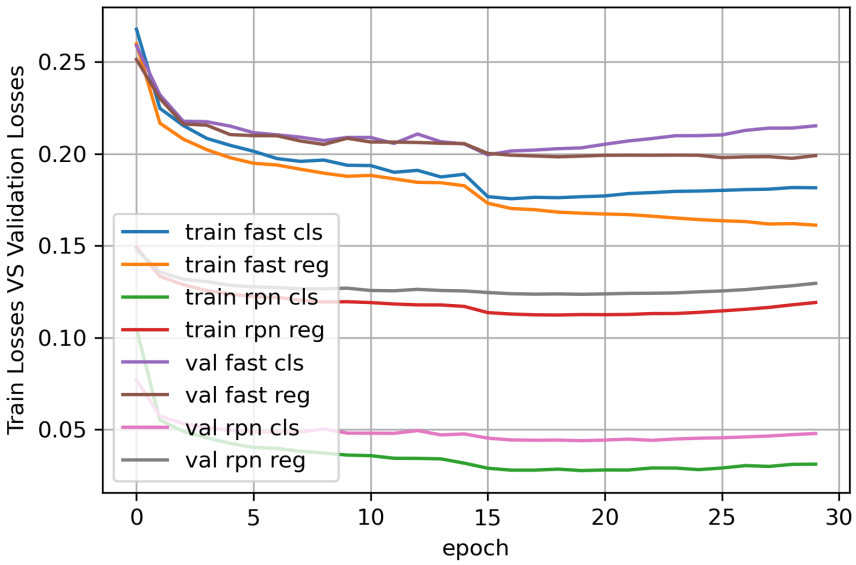


Figure 32: Model#8: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler, and $\alpha = 1$ for the first 15 epochs, and $\alpha = 0$ for the last 15 epochs

Note that in both models (Model#7, and Model#8), the losses associated with the Fast R-CNN are still much larger. For both models described in Figs. 31, 32, the mAP score from Kaggle became worst compared to the score achieved with Model#6.

Table 4: Kaggle public and private final score

	Private Score	Public Score
Model#7	0.5505	0.6099
Model#8	0.5797	0.6451

6.4 Ensembled Models

In this section we've applied the ensembled model described in Section 4.4 over different models.

First, we trained 5-fold cross validation models and ensembled them, where the first fold model is Model#6. The four rest folds are trained over the same network with the same hyper-parameters as Model#6. The train/validation loss curves for fold number 1 are already shown in Fig. 30.

Fig. 33 present the train and validation losses curves as function of the epoch for the four other folds.

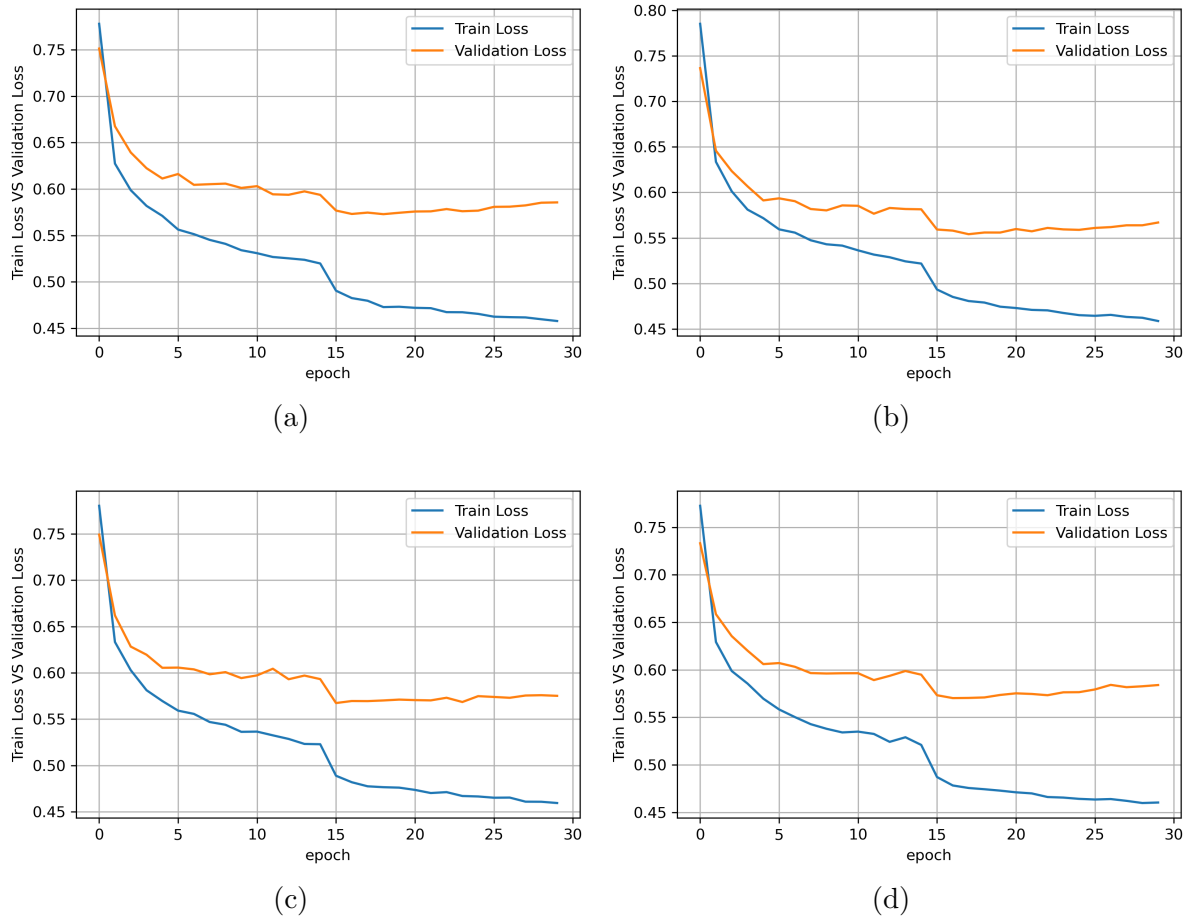


Figure 33: Adam optimizer, $lr = 1e-4$, weigh decay $5e-4$, with scheduler for different folds: (a) - fold#2, (b) - fold#3, (c) - fold#4, (d) - fold#5

The score achieved with the 5-fold ensembled model is **0.5825** on private, and 0.6601 on public. which is a little better than the previous best result in terms of private score.

We also tried other combinations of trained models. We combined Model#3, Model#6, Model#8, and the fifth fold from the 5-fold training. Using this model we achieved the best score so far in terms of private score (**0.5840**). The public score is also high and is equal to 0.6590.

7 Results

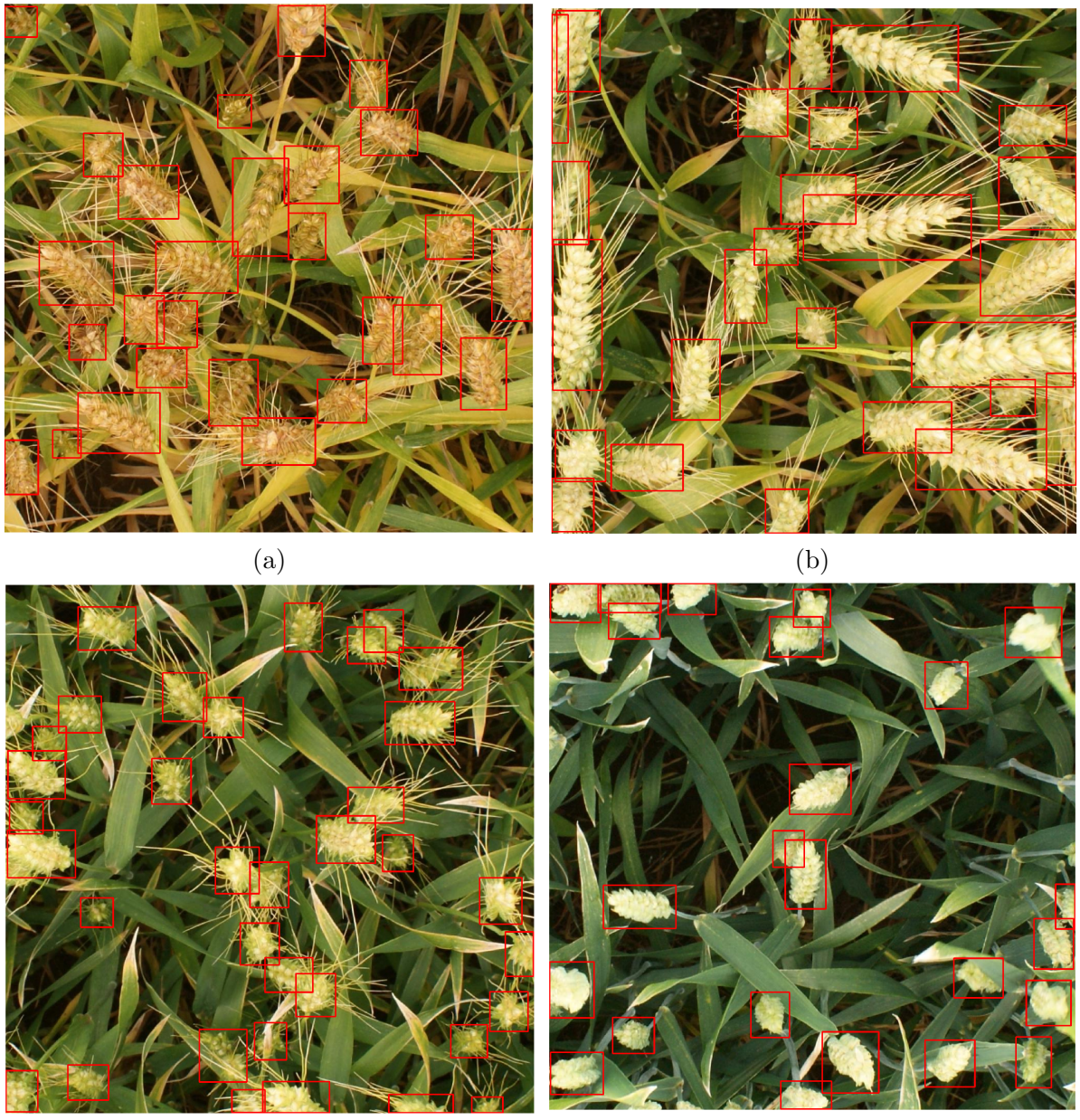
The best result in terms of the private score was achieved by the second ensembled model described in the previous section, which is the ensembling of Model#3, Model#6, Model#8, and the fifth fold from the 5-fold training. The figures corresponding to each of these models are Figs. 27, 30, 32, and 33d. To summarize this ensembled model achieved a **0.584** private score, which is the highest score among all our experiments. In terms of public score, Model#6 achieved the best result, which is 0.6617 (see Fig. 30, and Table.

3). As we are more interested in the private score, our best model is Model#8, described at the beginning of this section.

Competition Notebook	Run	Private Score	Public Score	Best Score
Global Wheat Detection	54.8s - GPU	0.5840	0.6590	0.584 V9

Figure 34: Kaggle best results

7.1 Visualization of the Ensembled Model predictions



(a) (b)

(c) (d)

Figure 35: The suggested ensembled model (Model#8) prediction over 4 randomly chosen images from the testset

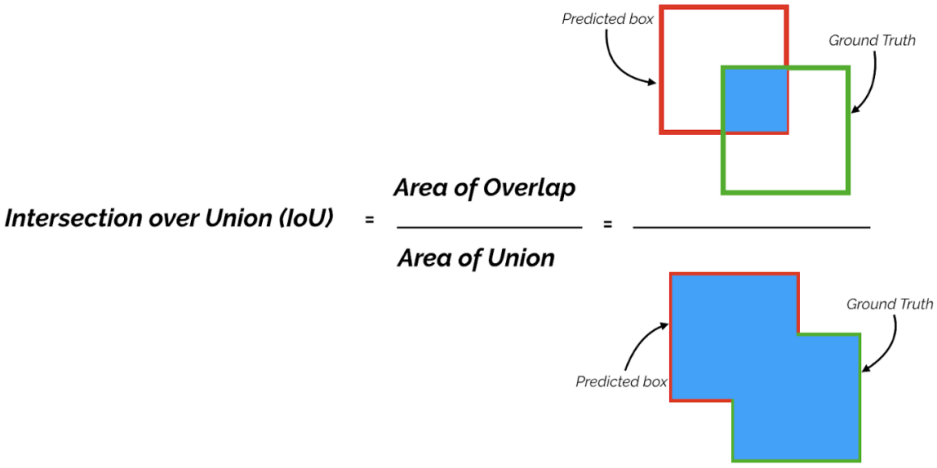
8 Conclusions & Summary

With the advancement of machine vision technology, computer vision detection algorithms have made wheat head detection and counting feasible. To accomplish this traditional task, a high-precision wheat head detection model with strong generalizability was presented in this project. During this project, we faced many challenges and considered many theoretical aspects and advanced techniques like transfer learning, and WBF for ensembling different object detection models. We came to several interesting conclusions when we tried different configurations such as augmentations, losses, learning rate, scheduling, optimizers, regulizers, ensembling, etc. To summarize, this project was very interesting and challenging, whether it is in the way of understanding the data and cleaning it, or the choice of the models for object detection and smartly combining them. Our best final (private) score is **0.584**.

9 Appendices

9.1 Intersection over Union

Object detection systems make predictions in terms of a bounding box and a class label. For each BBOX, we measure the overlap between the predicted BBOX and the ground truth BBOX. This is measured by IoU and defined as follow: For two BBOXes, A, and B, $IoU(A, B) = \frac{A \cap B}{A \cup B}$.



9.2 Mean Average Precision

mAP is a metric used to evaluate object detection models such as Fast R-CNN, YOLO, etc. At each point calculating an average precision value with specific threshold value of IoU . The threshold values range from 0.5 to 0.75 with a step size of 0.05. In order to calculate average precision for a specific IoU and for specific class we need to calculate 4 attributes: true positive (TP), true negative (TN), false positive (FP), false negative (FN).

- TP - The model predicted a label and matches correctly as per ground truth.
- TN - The model does not predict the label and is not a part of the ground truth.
- FP - The model predicted a label, but it is not a part of the ground truth.
- FN - The model does not predict a label, but it is part of the ground truth.

Based on that we can compute the average precision for specific threshold t as follow $AP_t = \frac{TP_t}{TP_t + FP_t + FN_t}$, where the sub-index t determine that IoU greater than t considered as match. Then, for any IoU threshold in the range $[0.5, 0.75]$ with step size 0.05, we calculate AP_t as described above, and then averaging over all the thresholds:

$$mAP = \frac{1}{|thresholds|} \sum_t AP_t$$

9.3 Non-Maximum Suppression

Recent object detection and localization techniques are based on deep neural networks with multiple anchors and multiscale feature maps, therefore, output a bundle of bounding box candidates around target objects on a given input image. NMS is a popular post-processing method to find a representative bounding box from multiple bounding boxes of a target object by filtering out the inferior bounding box candidates.

Algorithm 1 NMS

Input: A list of Proposal boxes B, corresponding confidence scores S and overlap threshold N

Output: A list of filtered proposals D

1: **while** $|B| \neq 0$ **do**
2: Select the proposal with highest confidence score, remove it from B and add it to the final proposal list D (initially D is empty)
3: Now compare this proposal with all the proposals — calculate the IoU of this proposal with every other proposal, if the IoU is greater than the threshold N, remove that proposal from B
4: Take the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D
5: Once again calculate the IoU of this proposal with all the proposals in B and eliminate the boxes which have high IoU than threshold
6: **end while**

$|\cdot|$ represents the cardinality of a finite set.

10 References

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[2] Roman Solovyev, Weimin Wang, and Tatiana Gabruseva. Weighted boxes fusion: Ensembling boxes from different object detection models. *Image and Vision Computing*, 107:104117, 2021.

[3] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[5] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[9] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.