HW4 – Image Analytics 1 – Report

1. Download the dataset using the link provided above, write the code to train a RetinaNet model to perform wheat detection within the data. As an initial step, you may follow the skeleton code at https://www.kaggle.com/code/jainamshah17/gwd-retinanet-pytorch -train/notebook. Report the results and the relevant metrics from training the initial model.
   1. Our initial model’s specs were below–
      1. Resnet: 50 layers
      2. Epoch: 1
      3. Learning rate: 0.0001
   2. Results
      1. Training Loss



* + 1. Validating Loss



* + 1. Wheat Detection Example

A screenshot of a computer

Description automatically generated with medium confidence

* + 1. The details about the Wheat Detection process can be found in the Code submitted through GitHub.

1. Next, try improving the RetinaNet by optimizing the model from various aspects.
   1. Tried Methods
      1. Different pre-trained model backbones
      2. Different optimizers and learning rates
      3. Modifying the loss functions
   2. Results

| # Of Train data | # Of Test data | Resnet | Optimizer | Learning rate | Epoch | Loss Function  (Class : Reg) | Training | | Validation | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Classification loss | Regression loss | Classification loss | Regression loss |
| 10 | 10 | 50 | SGD | 0.0001 | 1 |  | 0.62393 | 1.05119 | 0.60804 | 0.97538 |
| All | All | 50 | Adam | 0.0001 | 1 |  | 0.11729 | 0.43611 | 0.1552 | 0.48077 |
| All | All | 50 | Adam | 0.0001 | 15 |  | 0.09358 | 0.35027 | 0.17382 | 0.44811 |
| All | All | 50 | Adam | 0.0001 | 5 |  | 0.11383 | 0.39625 | 0.1592 | 0.59779 |
| All | All | 34 | Adam | 0.0001 | 1 |  | 0.16576 | 0.50321 | 0.11885 | 0.41625 |
| All | All | 18 | Adam | 0.0001 | 1 |  | 0.16346 | 0.52697 | 0.18224 | 0.51237 |
| All | All | 18 | Adam | 0.001 | 1 |  | 0.22497 | 0.50544 | 0.20933 | 0.59475 |
| All | All | 18 | Adam | 0.01 | 1 |  | 337.27734 | 0.99551 | 577.3396 | 1.02978 |
| All | All | 18 | SGD | 0.0001 | 1 |  | 1.07306 | 1.04018 | 1.07301 | 1.04802 |
| All | All | 18 | Adam | 0.0001 | 1 |  | 0.19279 | 0.59581 | 0.15951 | 0.48409 |
| All | All | 18 | Adam | 0.0002 | 1 |  | 0.182 | 0.55449 | 0.22882 | 0.5726 |
| All | All | 18 | Adam | 0.0001 | 1 | 1:2 | 0.19051 | 0.55782 | 0.40091 | 0.57733 |
| All | All | 18 | Adam | 0.0001 | 1 | 2:1 | 0.17241 | 0.54898 | 0.24613 | 0.53598 |

* + 1. Different pre-trained model backbones
       1. Resnet 50’s result showed the lowest training loss among Resnets, but Resnet 34 is proved as the one in the Validation part
    2. Different optimizers and learning rates
       1. In the same condition, when Adam was applied as an optimizer, the loss was found to be lower than the case using SGD
       2. It was confirmed that inaccurate detection could occur if the learning rate was excessively high
    3. Modifying the loss functions
       1. It was affirmed that the classification loss and regression loss differed according to the weight of loss function components.   
          However, the model's total loss was the lowest when the weights of both classification loss and regression loss are identical.

Resnet34 /epoch 1 / lr0.0001

Training



Validating



Example

A map of a city

Description automatically generated with medium confidence

Resnet18 /epoch 1 / lr0.0001

Training



Validating



Example

A screenshot of a computer

Description automatically generated with low confidence

Resnet18 /epoch 1 / lr0.001

Training



Validating



Example

A map of a city

Description automatically generated with low confidence

Resnet18 /epoch 1 / lr0.01

Training



Validating

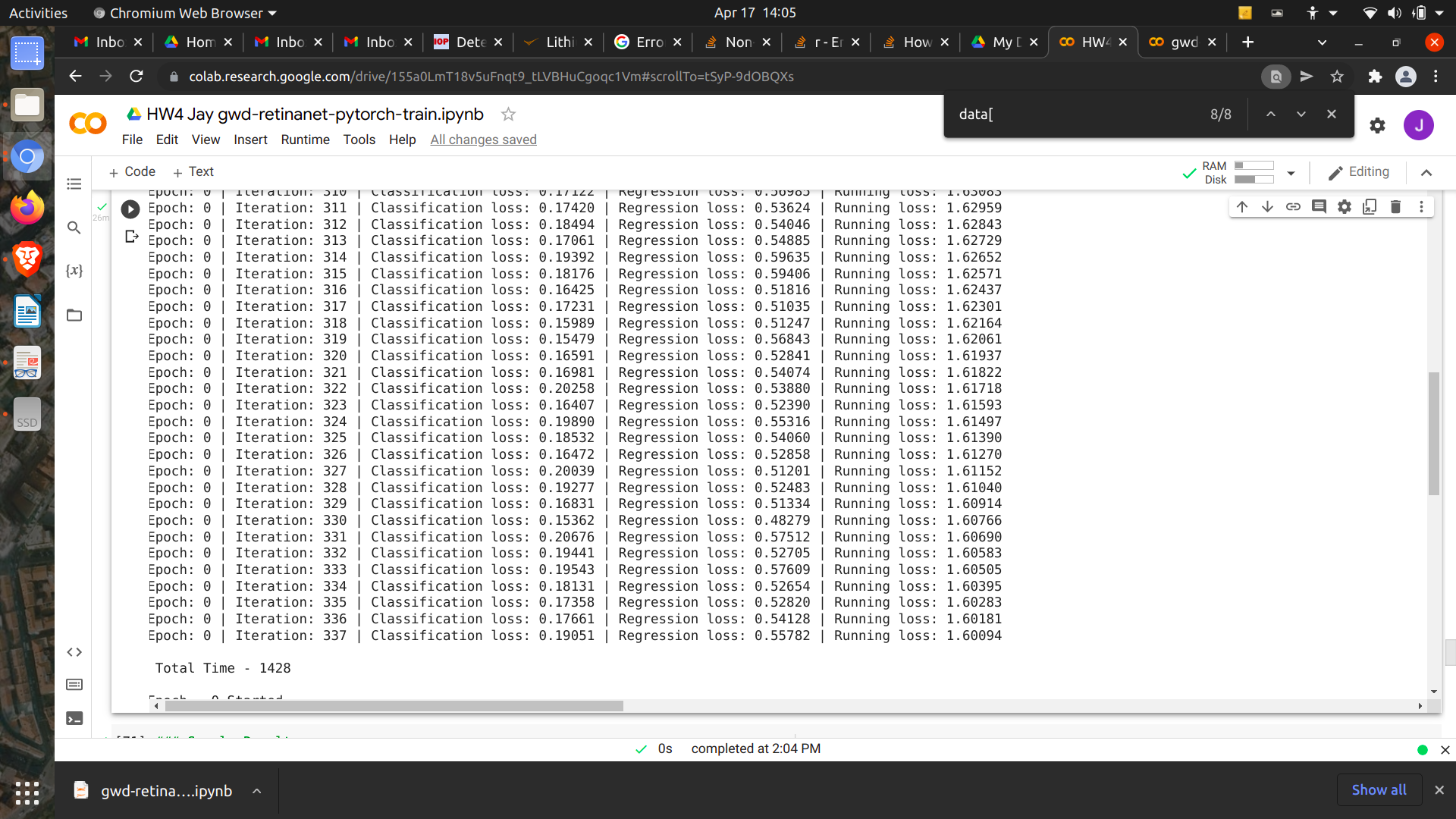


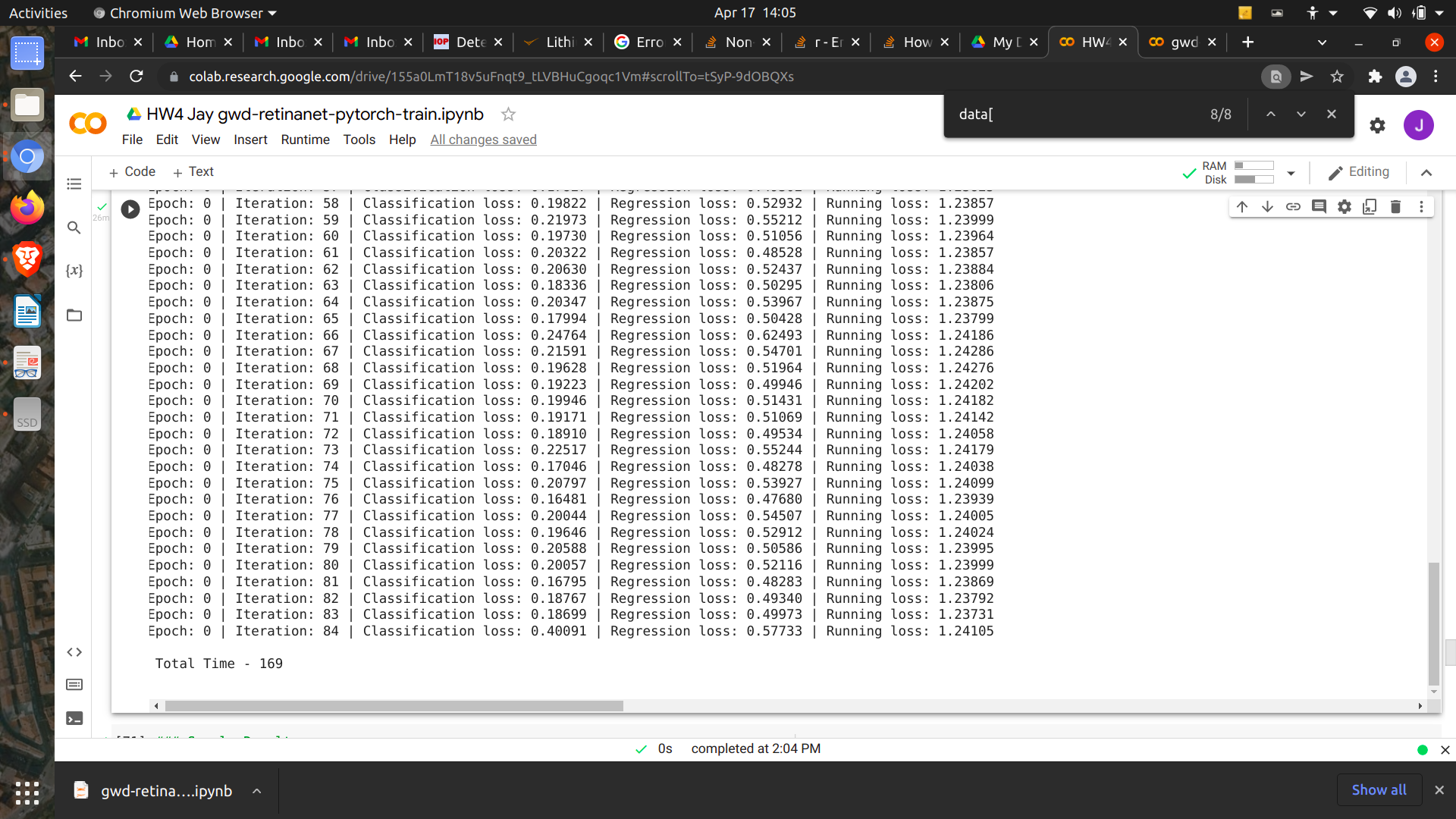
Example

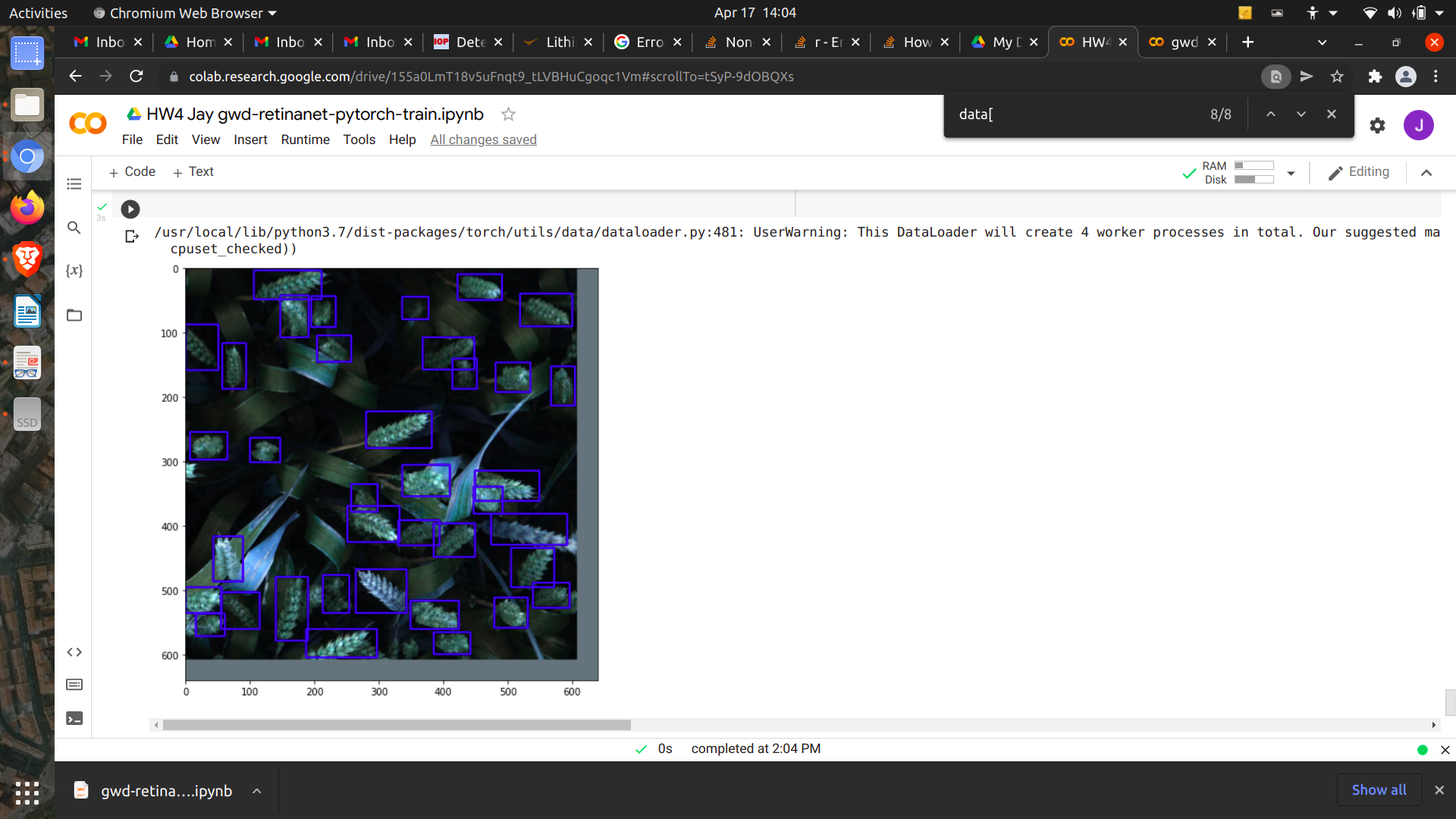
A picture containing text, electronics, display

Description automatically generated

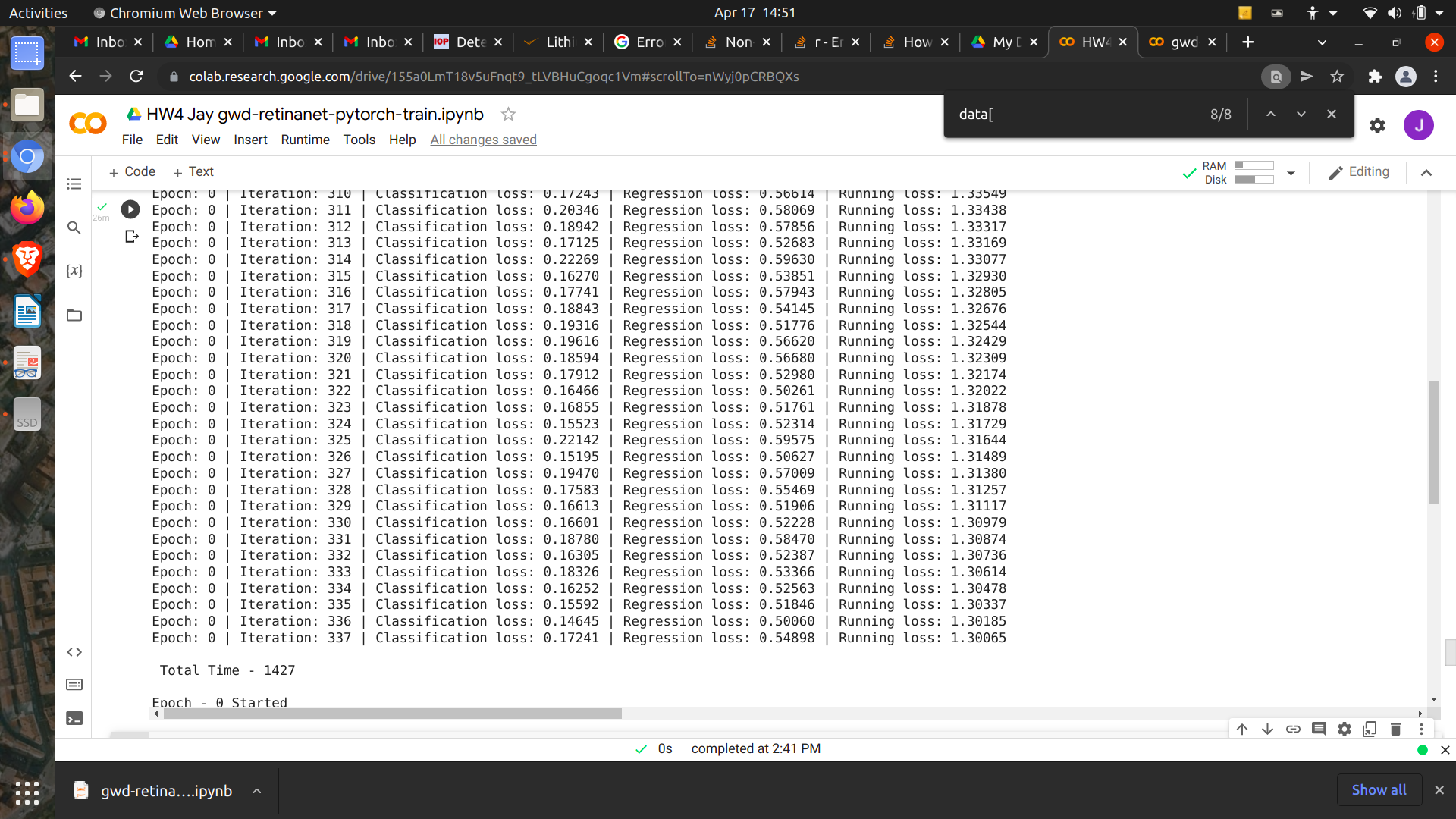
Resnet18 /epoch 1/lr0.0001/ 1:2 (Classification Loss:Regression Loss)

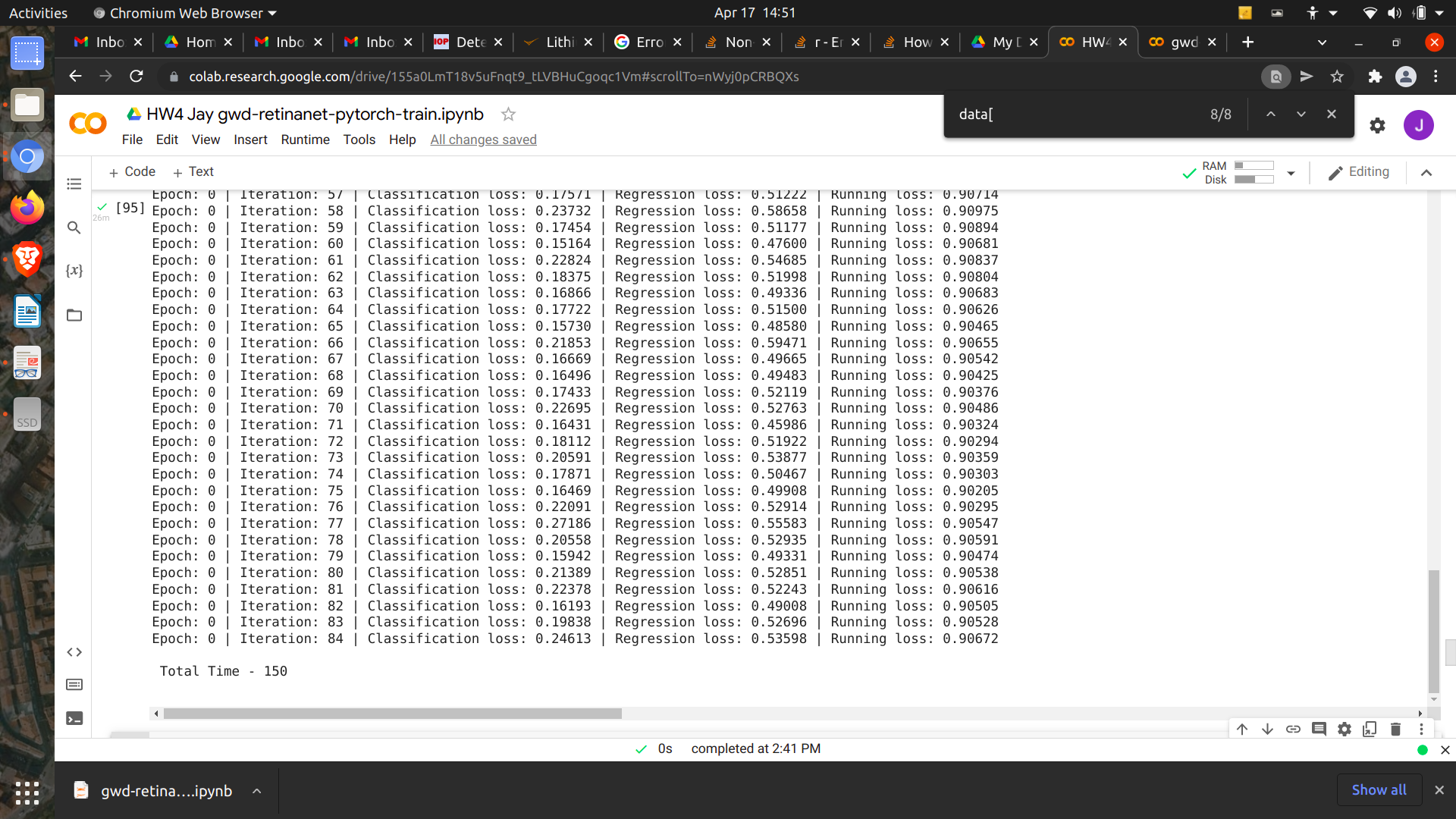


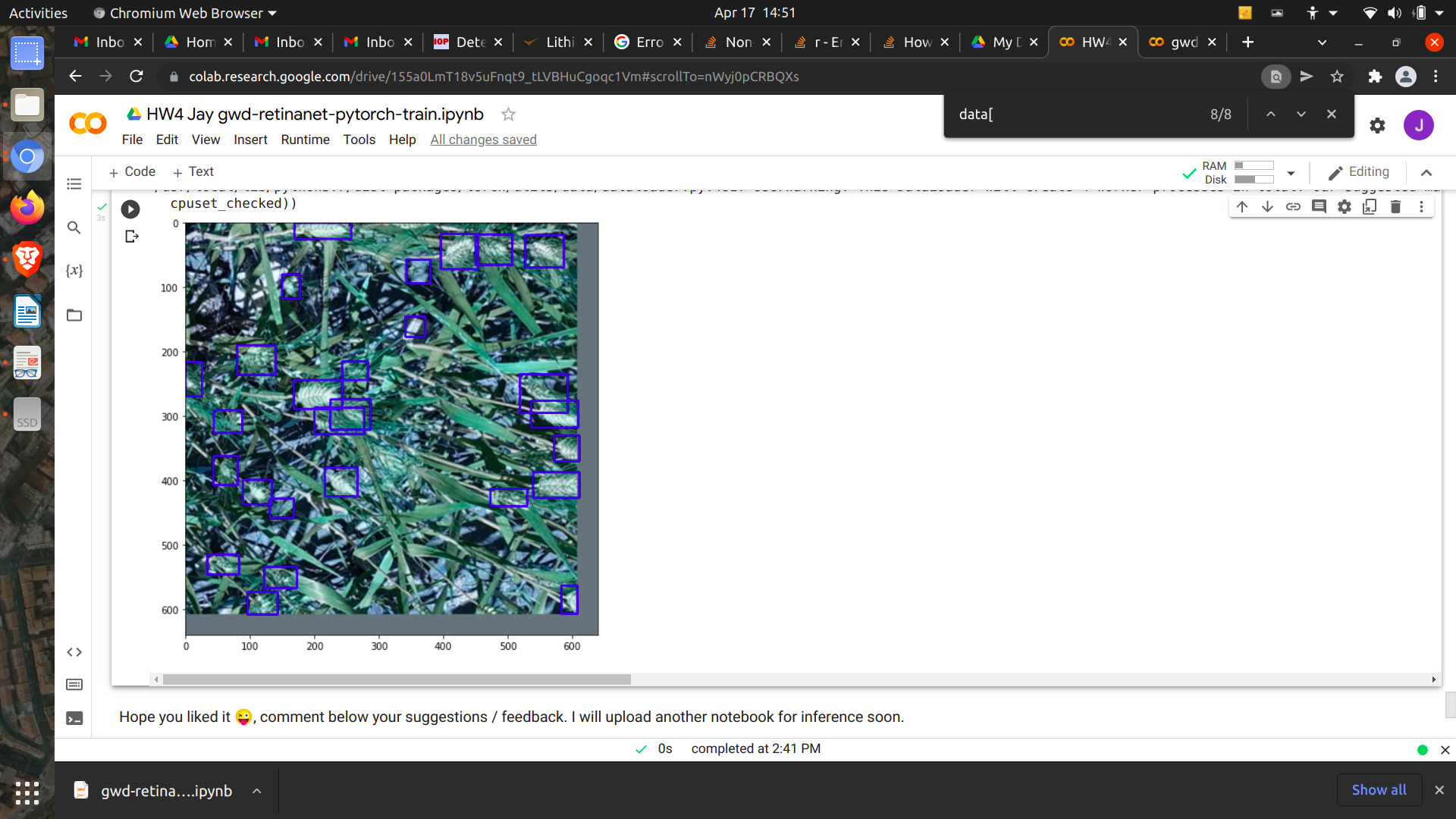




Resnet18 /epoch 1 /lr 0.0001/ 2:1 (Classification Loss:Regression Loss)







Resnet50 /epoch 5 / lr0.0001



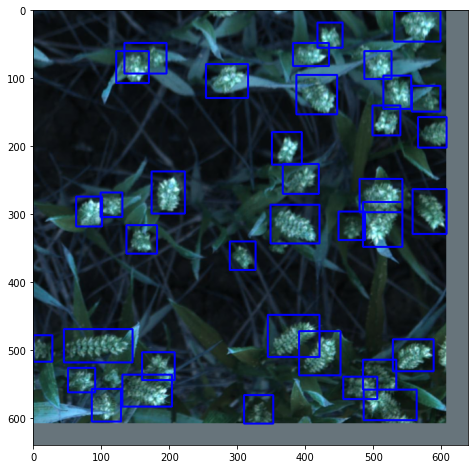




Resnet50 /epoch 15 / lr0.0001







1. Finally, include a discussion on your understanding of RetinaNet and how does it differ from other object detection models, such as Fast-RCNN and Faster-RCNN
   1. First of all, RetinaNet is a one-stage object detection model, which has a simple and unified network including a backbone network and two task-specific subnetworks. The Backbone network computes a convolutional feature vector of the entire image set, and the two task-specific subnetworks are composed of convolutional object classification part and convolutional bounding box regression.
   2. RCNN requires a lot of time to train the model because it is needed a huge amount of pre-classified regions. In order words, the algorithm is fixed so that no more learning process occurs at the stage. Retinanet is an one stage object detection algorithm where the bounding boxes are predicted without any region proposals. In contrast, fast RCNN and faster RCNN are 2 stage detectors and use region proposal network to predict the bounding boxes and a pooling layer for the subsequent regression and classification. One-stage detectors have high inference speeds and two-stage detectors have high localization and recognition accuracy