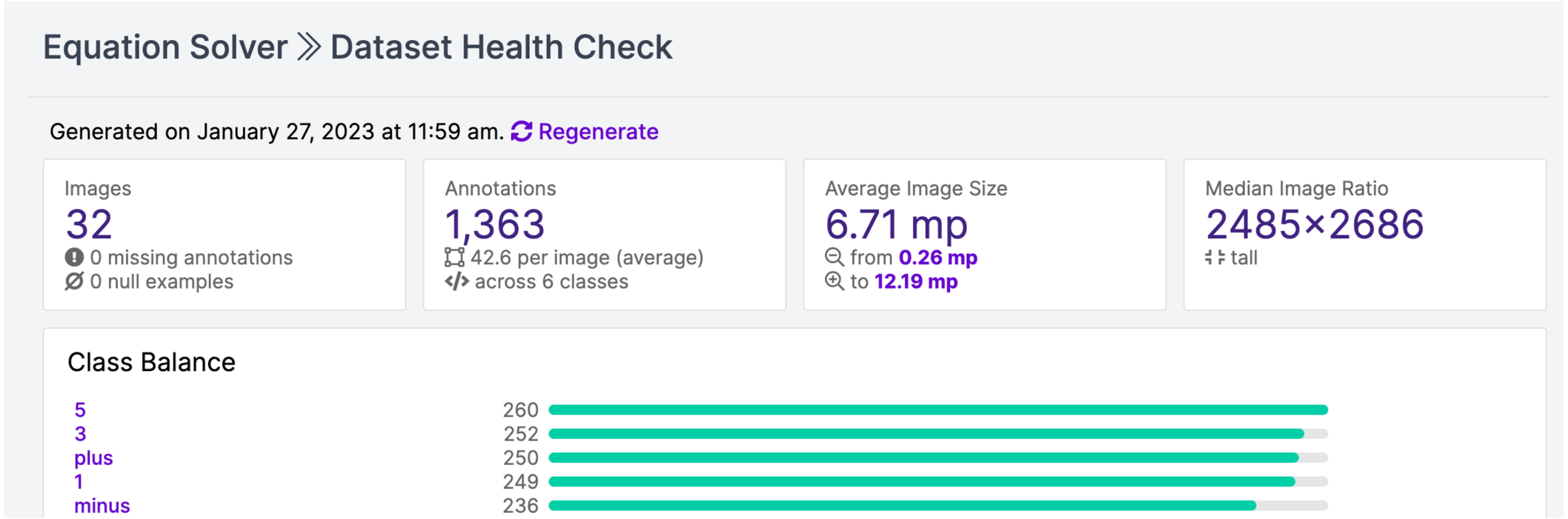
Section 1 (Building a dataset):

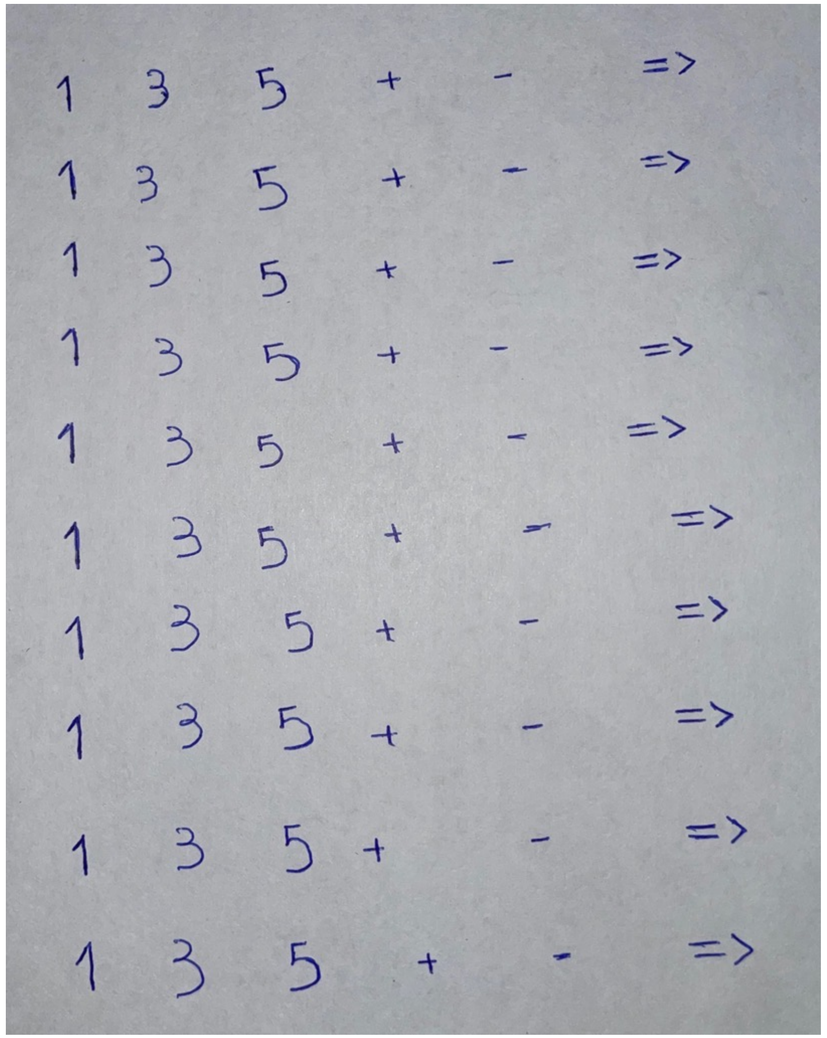
Create a dataset of the following English letters and characters {5,3,1,+,-}
with your handwriting and those around you, which are the classes for your model.

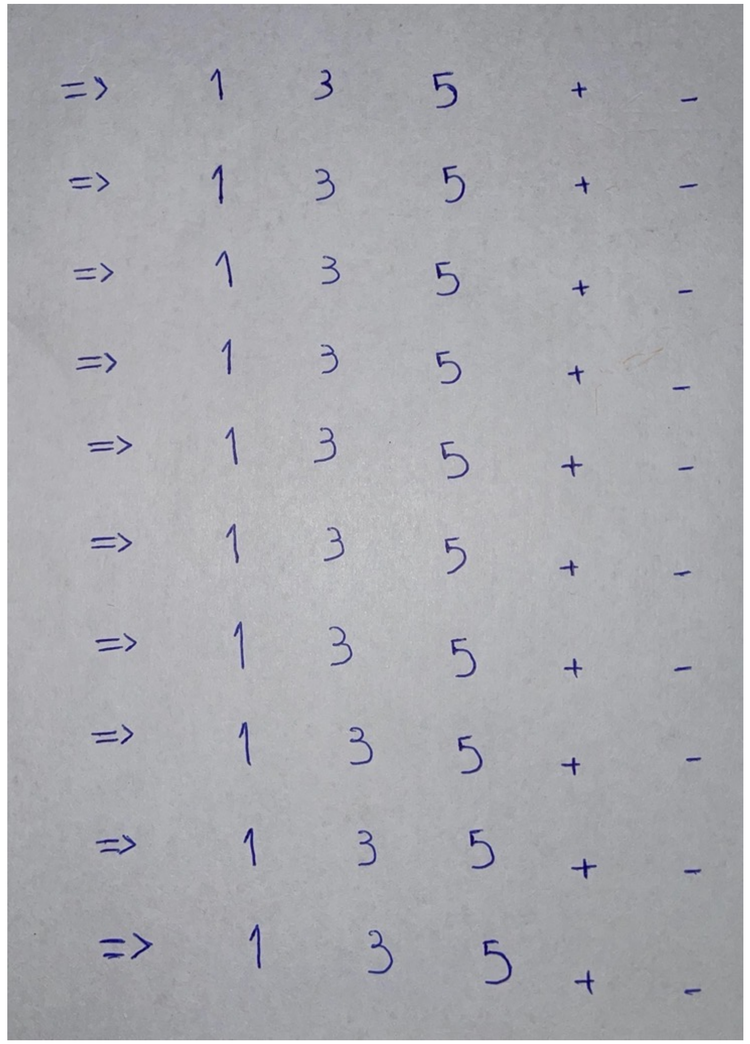
In such a way that you write the characters on paper and in different modes and with the handwriting of several people, which is a data set as general as possible.

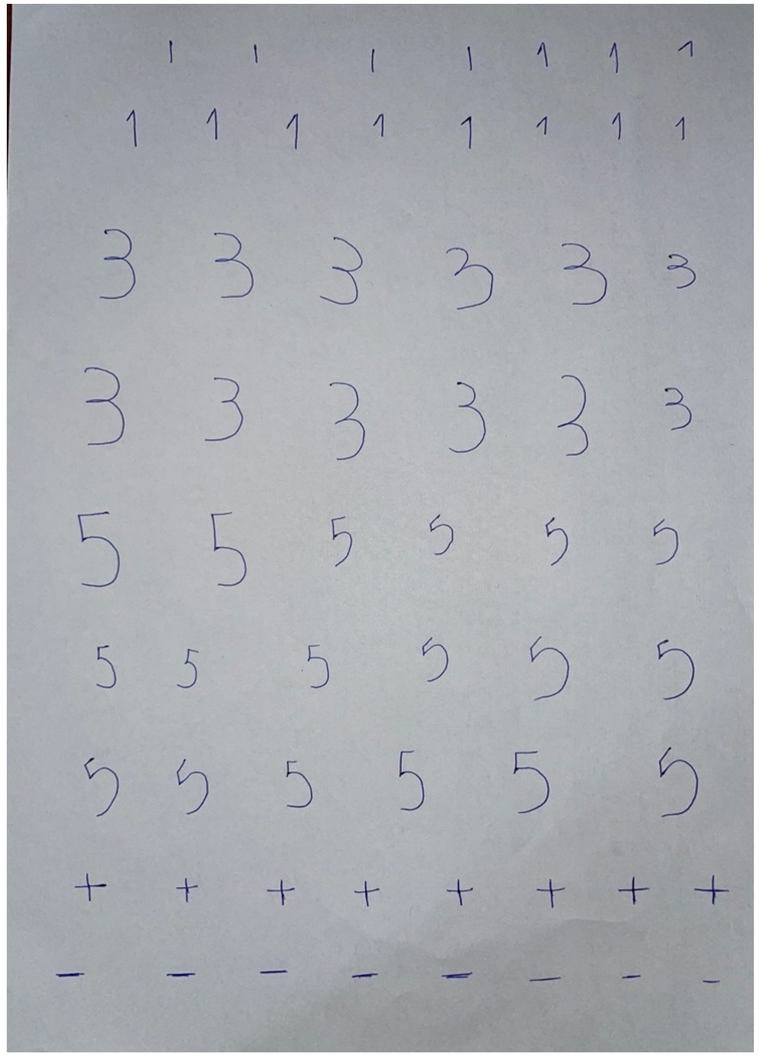


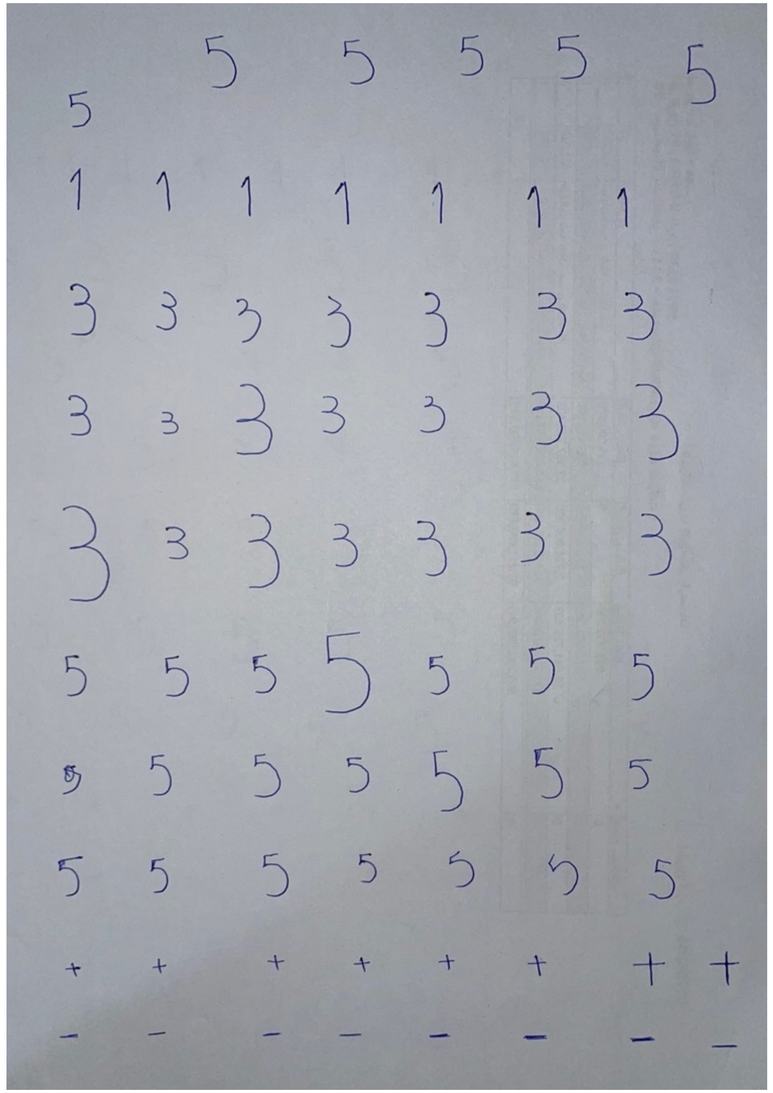
In the collected dataset, it has been tried to collect the numbers 1, 3, and 5 along with - and + signs with different handwriting and sizes from different people so that the neural network can observe the types of writing of each character and train them. As you can see, 32 images containing about 300 repetitions of each character have been collected.

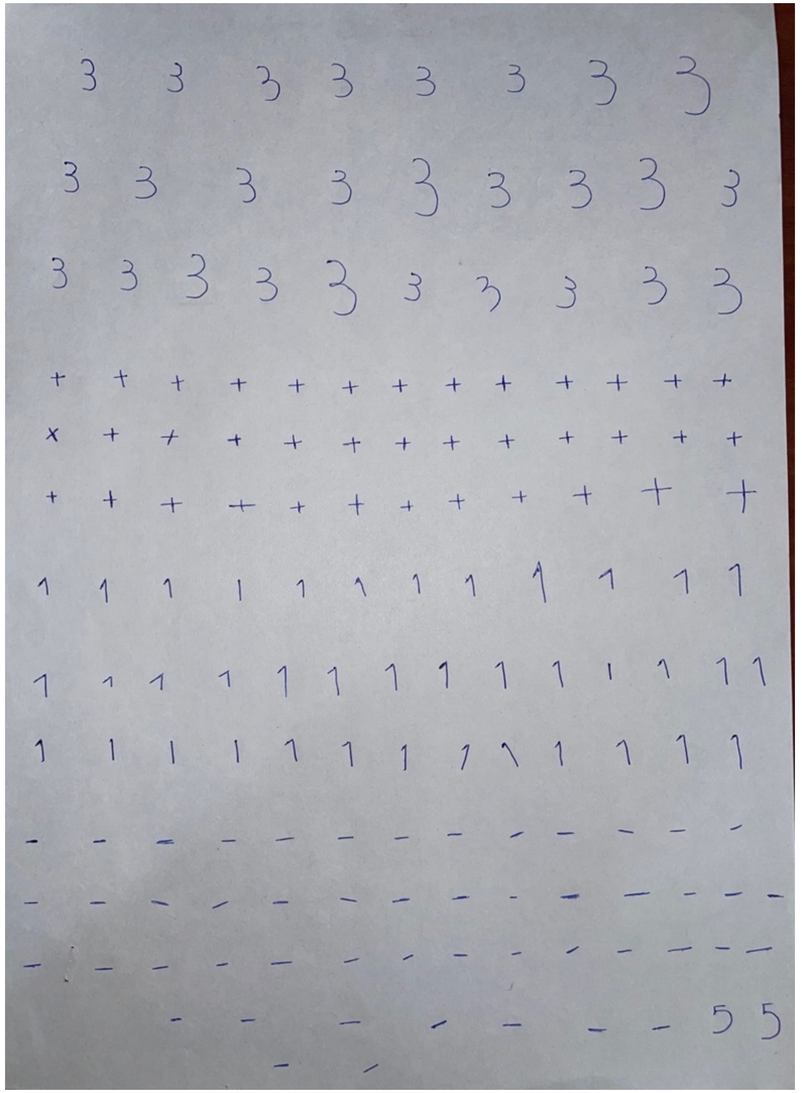
A number of random dataset images

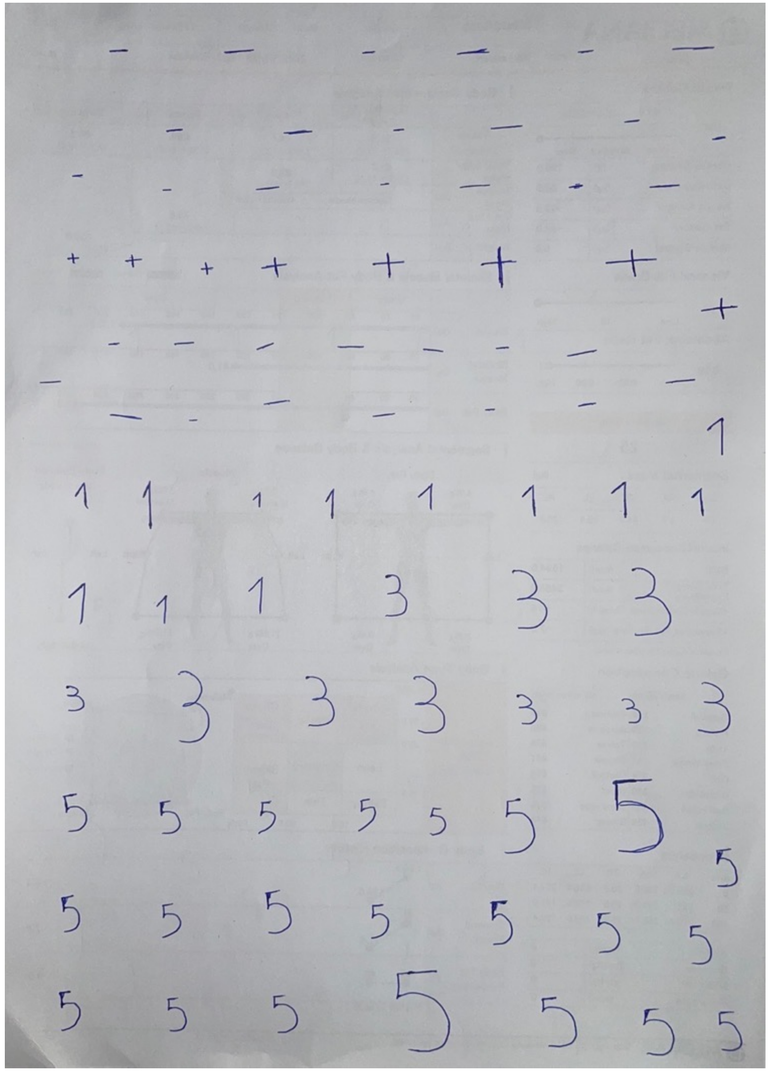

1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>
1 3 5 + - =>

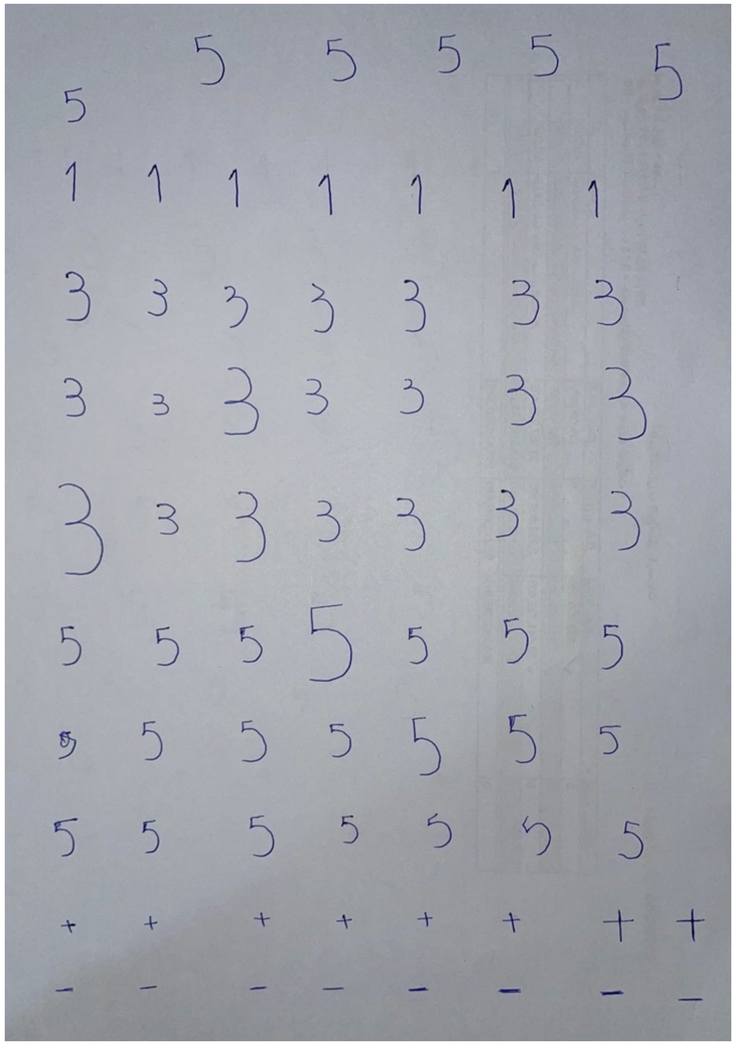

=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -
=> 1 3 5 + -

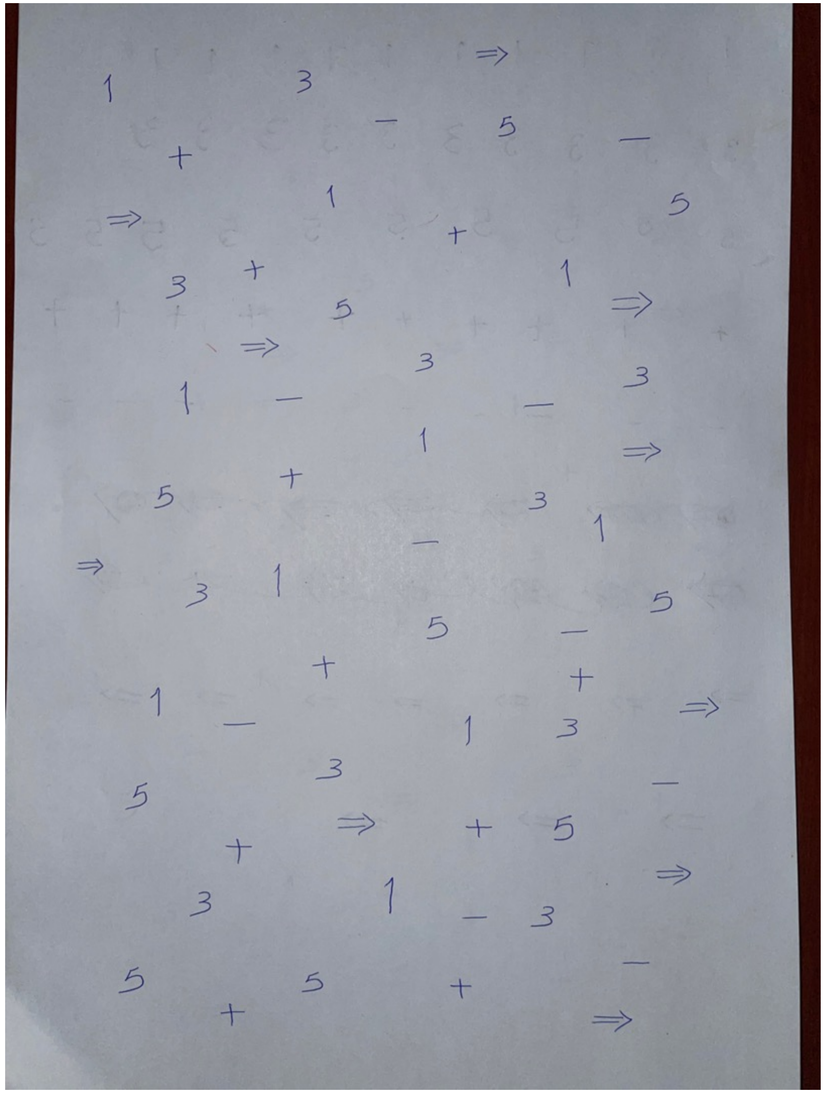

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
3 3 3 3 3 3
3 3 3 3 3 3
5 5 5 5 5 5
5 5 5 5 5 5
5 5 5 5 5 5
+ + + + + + + +
- - - - - - - -

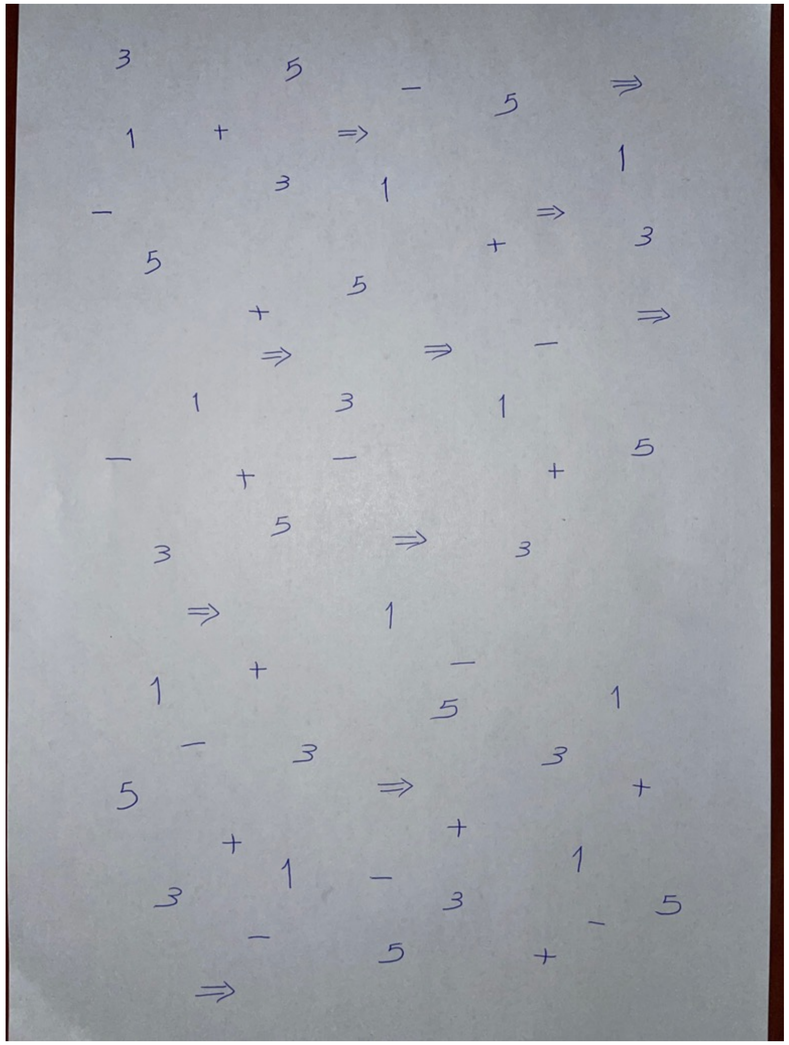












Section 2 (Labeling and Augmentation) :

A: Research the concept of Data Augmentation and especially the issues of Object Detection and explain your findings in a paragraph.

B: Label the images using the site Roboflow (You have 5 classes, but you can add a flash class as needed.

C: Try to balance and diversify the dataset using the site's Augmentation tools.

D: Briefly explain the reason for using each method.

A) Data augmentation:

Data augmentation is a technique for artificially augmenting a training set by creating modified versions of a dataset using existing data. This includes making minor changes to the dataset or using deep learning to generate new data points, including cropping, flipping, resizing, brightness, contrast, horizontal flipping, and adding noise.

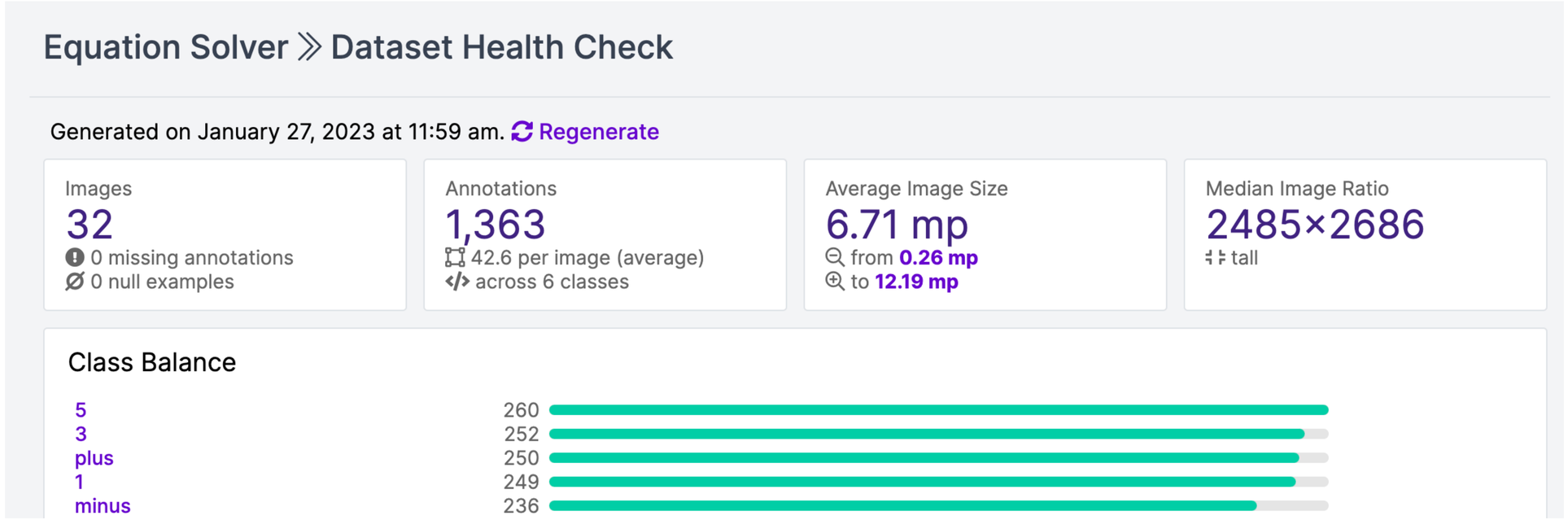
In the construction of artificial networks that are designed for the purpose of object detection, it is very important to be careful in using these techniques that what kind of subject is supposed to be detected, these techniques should be used, and the use all of These techniques are not generally suggested in all tasks! For example, if we are looking at the identification of the number of cars on the street, the type and intensity of using these techniques are varied and different.

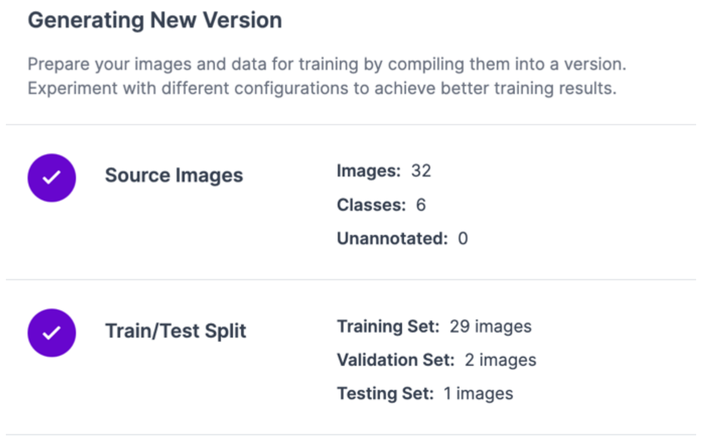
B)

After entering the Roboflow site and uploading the dataset including various images, you should start separating each character and specifying its class. We repeat this until the last character so that all of them have the desired special class.

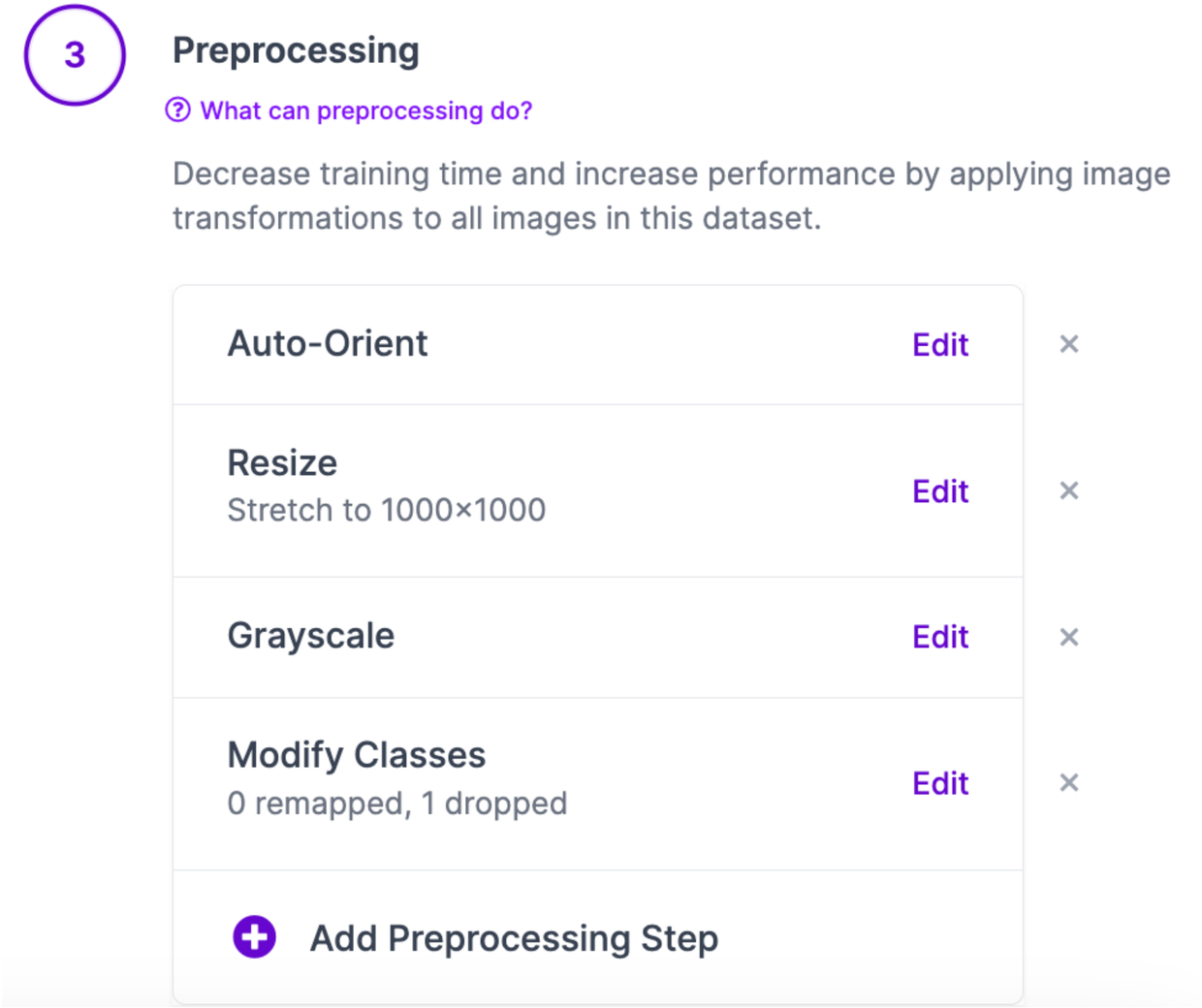
An example of a labeled sheet in the Roboflow tool

Labeled sheets in the Roboflow tool





Labeled data set information



C)

Before setting up and applying Data Augmentation, we first apply pre-processing parameters and model training, or Pre-Processing.

We changed the size of all the images in the dataset to 1000 x 1000 dimensions for more coordination and accuracy of model learning.

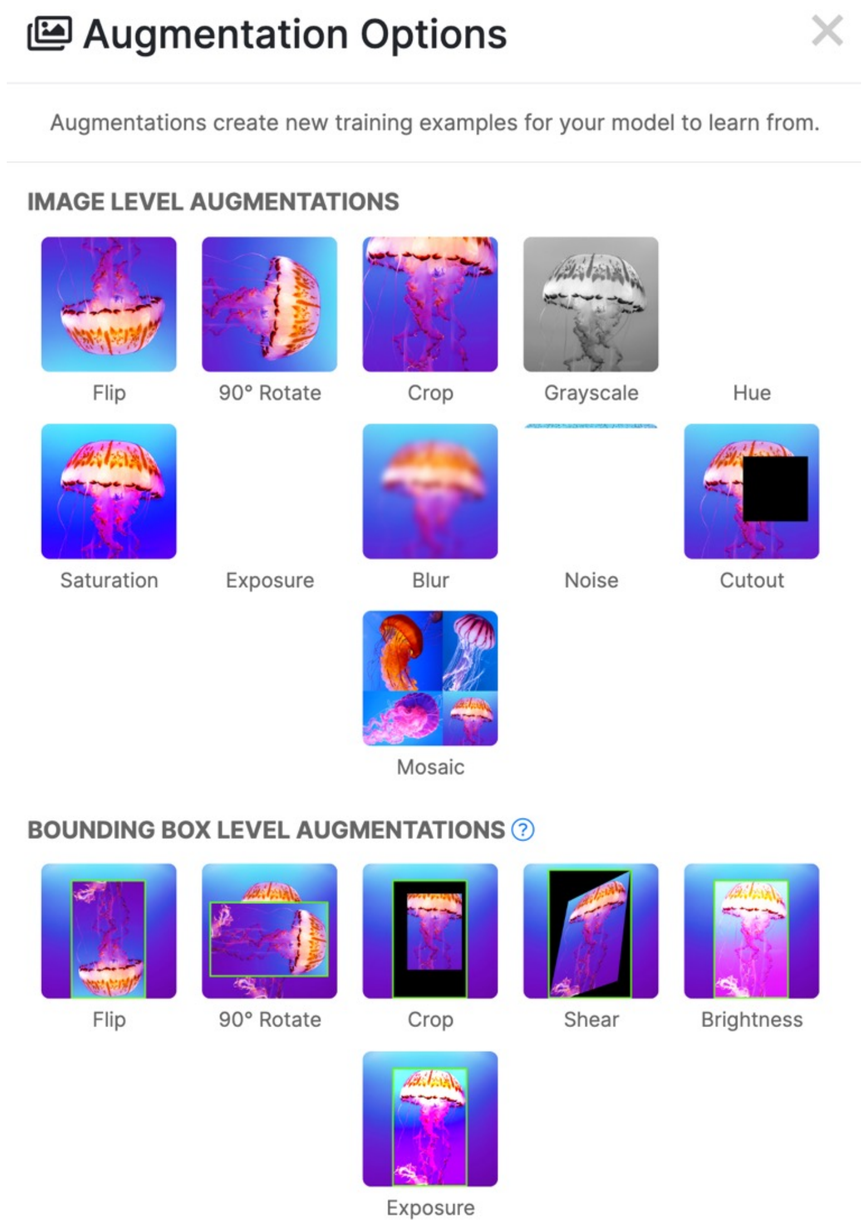
We converted the images to grayscale or all grayscale images.

And then we decided not to use the flash class in solving the question and removed it from the dataset so that the output of the dataset does not have a label as a flash.

We changed the size of all the images in the dataset to 1000 x 1000 dimensions for more coordination and accuracy of model learning.

We converted the images to grayscale or all grayscale images.

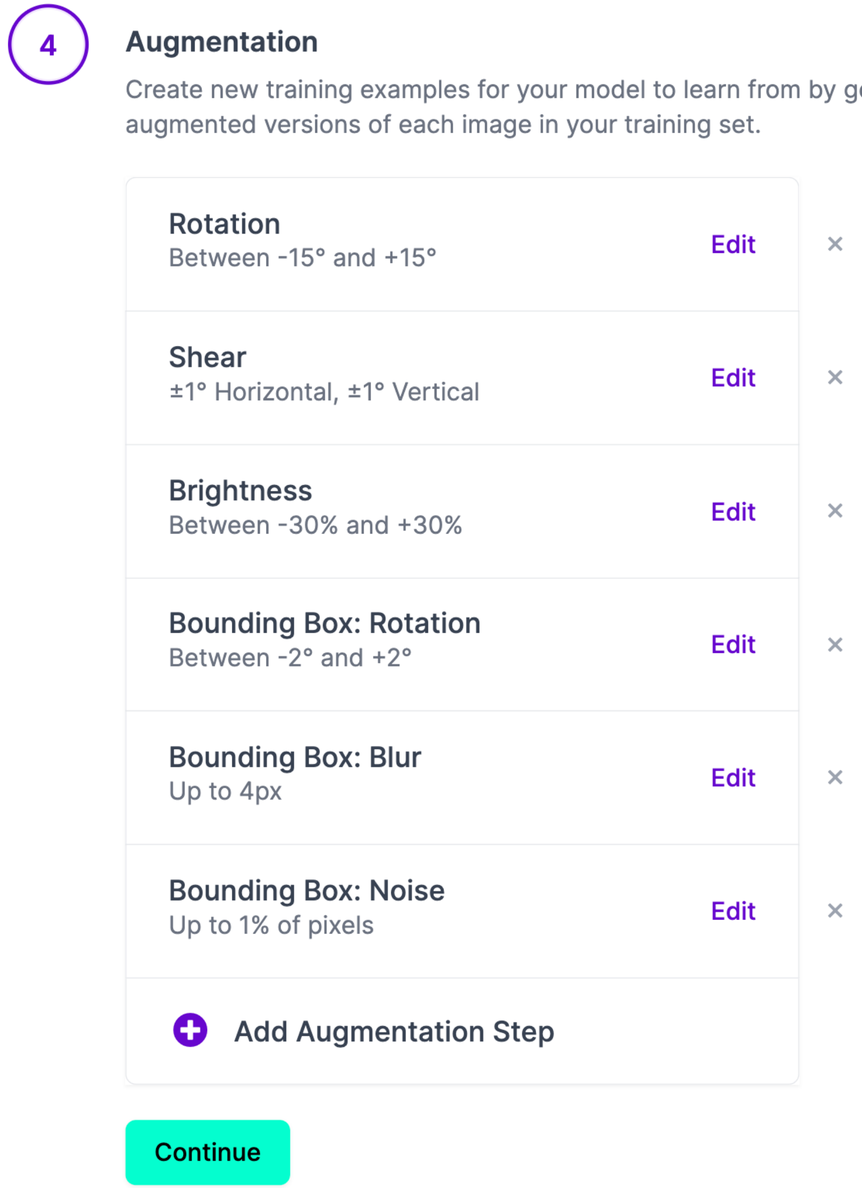
And then we decided not to use the flash class in solving the question and removed it from the dataset so that the output of the dataset does not have a label as a flash.



C)

First, we take a look at the choices or techniques that we can choose from to generate augmented data.

As mentioned, in this site and tool, our hand is open to choose options, but in Object Detection questions, it is an important point that what subject we are looking for inside the model images and to what extent the input images may be outside of They are waiting for us. After considering these things, we start using and applying the desired techniques.



C)

Techniques selected and added to the original dataset to solve this question in order to train the model and neural network

4

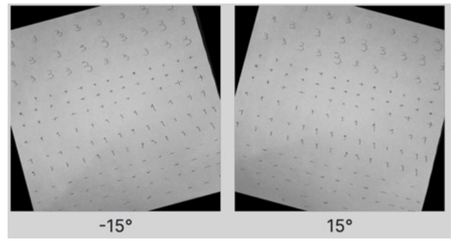
Augmentation

Create new training examples for your model to learn from by generating augmented versions of each image in your training set.

Rotation Between -15° and +15°	Edit	×
Shear ±1° Horizontal, ±1° Vertical	Edit	×
Brightness Between -30% and +30%	Edit	×
Bounding Box: Rotation Between -2° and +2°	Edit	×
Bounding Box: Blur Up to 4px	Edit	×
Bounding Box: Noise Up to 1% of pixels	Edit	×

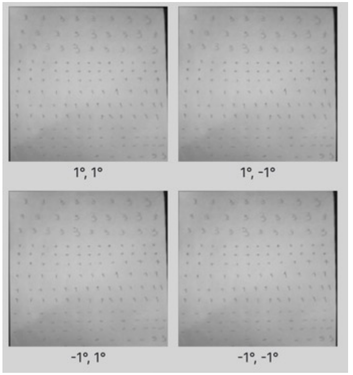
⊕ Add Augmentation Step

Continue



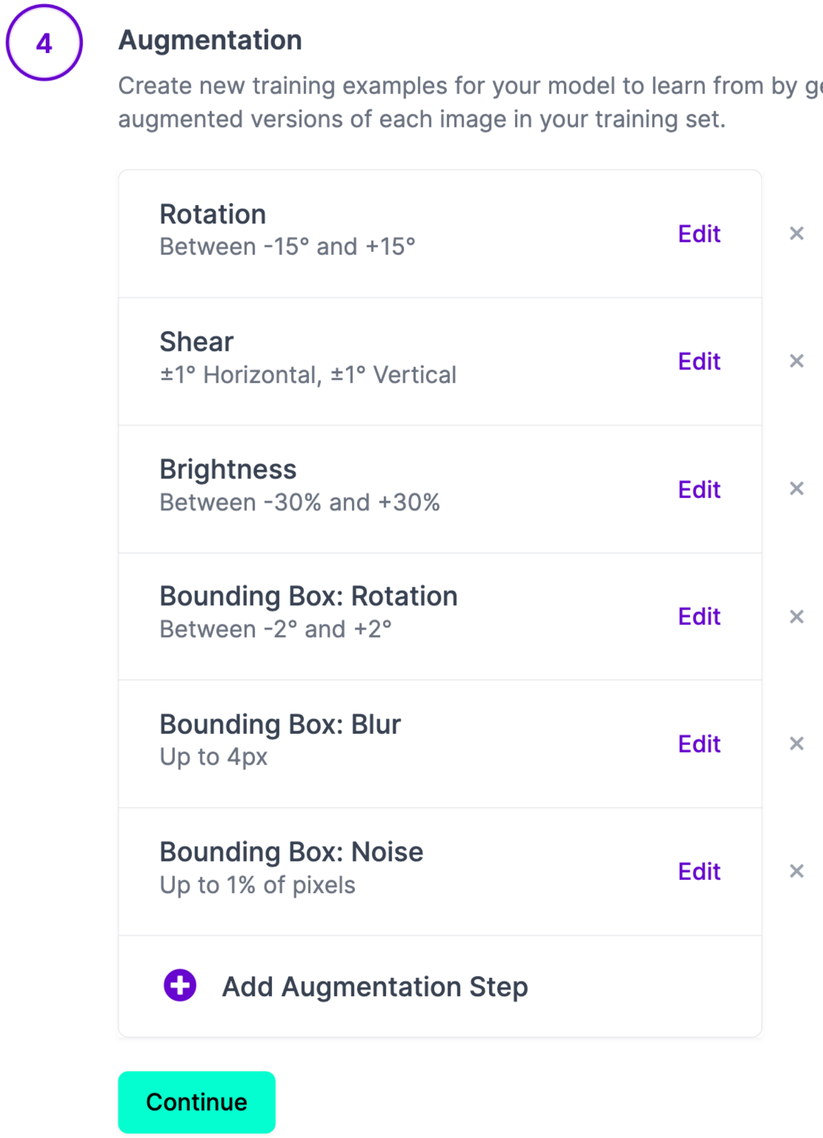
Rotation:

This technique has been added to the original data in order to further familiarize and train the model for slightly rotated characters.



Shear:

This technique is added to the original data to learn characters that are slanted or cropped images to cover numbers that are written in italics.



D)

Bounding Box: Rotation

Bounding Box: Blur

Bounding Box: Noise

These techniques are only applied to the labeled boxes, not to the entire image. These techniques have been used to apply these additional data only on the boxes for more accuracy and the possibility of dealing with the main subjects. Techniques such as rotating the characters and blurring the boxes and adding some salt and pepper noise to the boxes.

4

Augmentation

Create new training examples for your model to learn from by generating augmented versions of each image in your training set.

Rotation Between -15° and +15°	Edit	×
Shear ±1° Horizontal, ±1° Vertical	Edit	×
Brightness Between -30% and +30%	Edit	×
Bounding Box: Rotation Between -2° and +2°	Edit	×
Bounding Box: Blur Up to 4px	Edit	×
Bounding Box: Noise Up to 1% of pixels	Edit	×

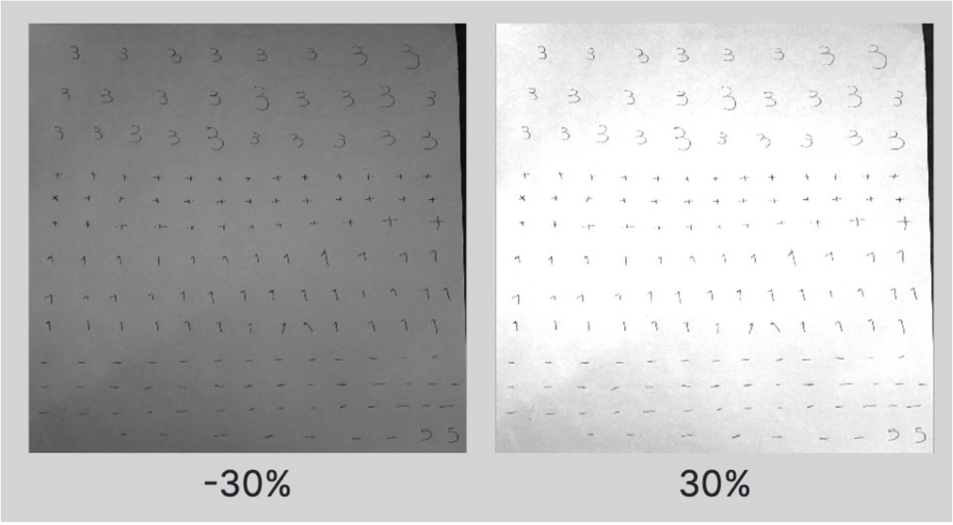
⊕ Add Augmentation Step

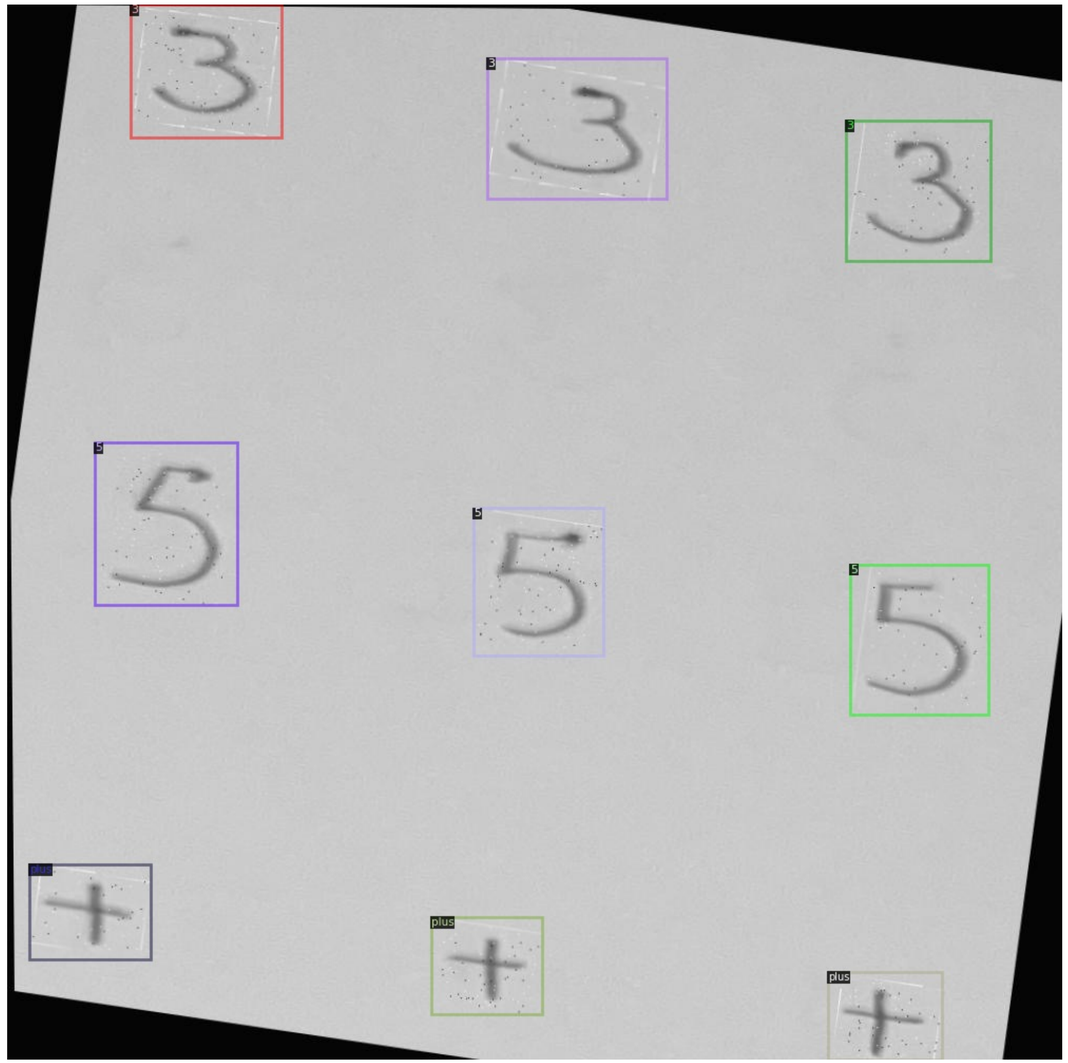
Continue

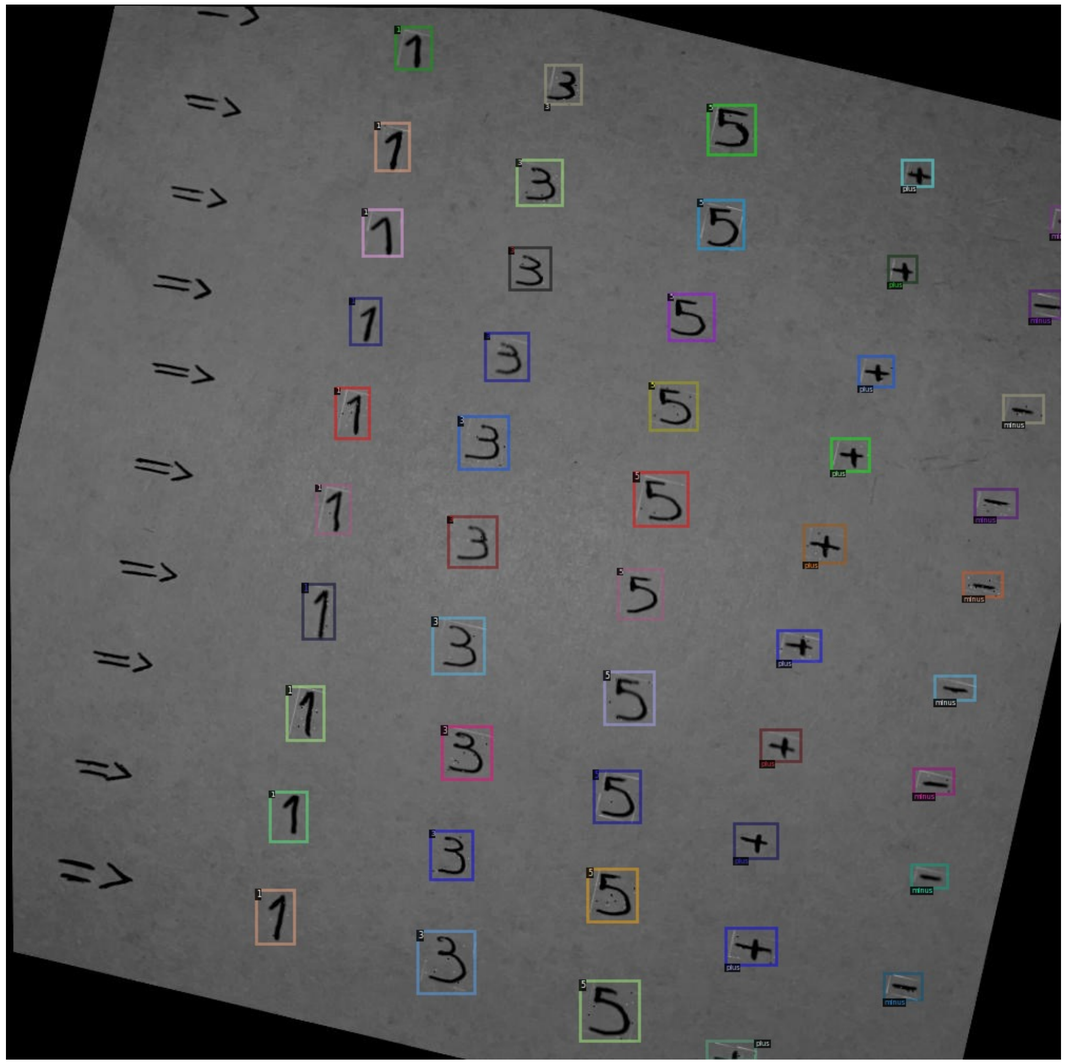
D)

Brightness:

This technique, which has been applied to all the images, has been used to adjust the light and brightness conditions in order to train the model to see when faced with weak or very strong light conditions, which may have an effect on the neighborhood values of the characters, such as noise.







An example of images added to the dataset

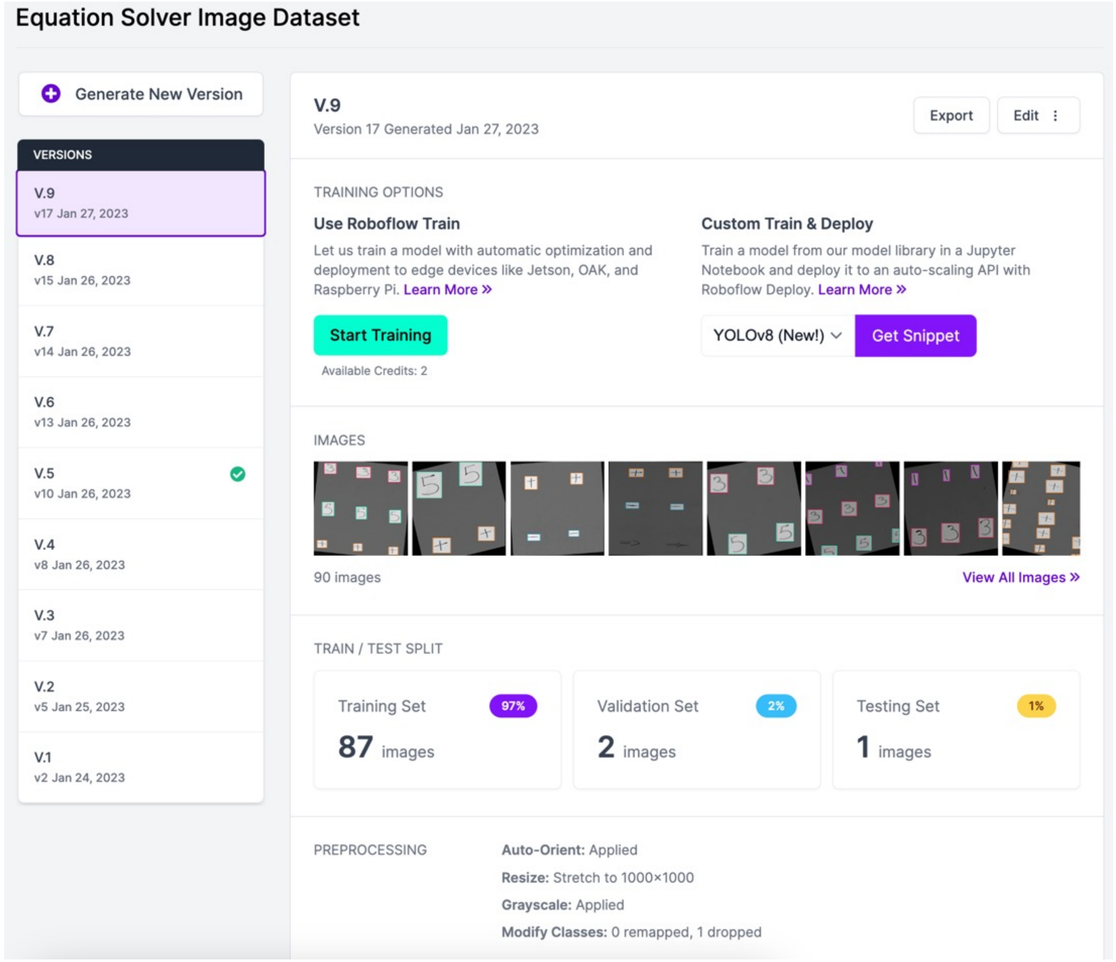
Section 3 (model training):

A: Transfer the model to Google Colab using the link provided by the labeling tool (without downloading).

B: Use Detectron2 to train the model and report the accuracy of the model and explain the criteria you used to measure it and give the model a similar image of your choice and display the result. Make the model calculate the answer for each equation given in the test images (each equation starts with an arrow) and write an image next to it (like the sample image).

A)

In order to output the dataset set from the Roboflow site, a version of it must be produced first.



AUGMENTATIONS

Outputs per training example: 3

Rotation: Between -15° and +15°

Shear: ±1° Horizontal, ±1° Vertical

Brightness: Between -30% and +30%

Bounding Box: Rotation: Between -2° and +2°

Bounding Box: Blur: Up to 4px

Bounding Box: Noise: Up to 1% of pixels

DETAILS

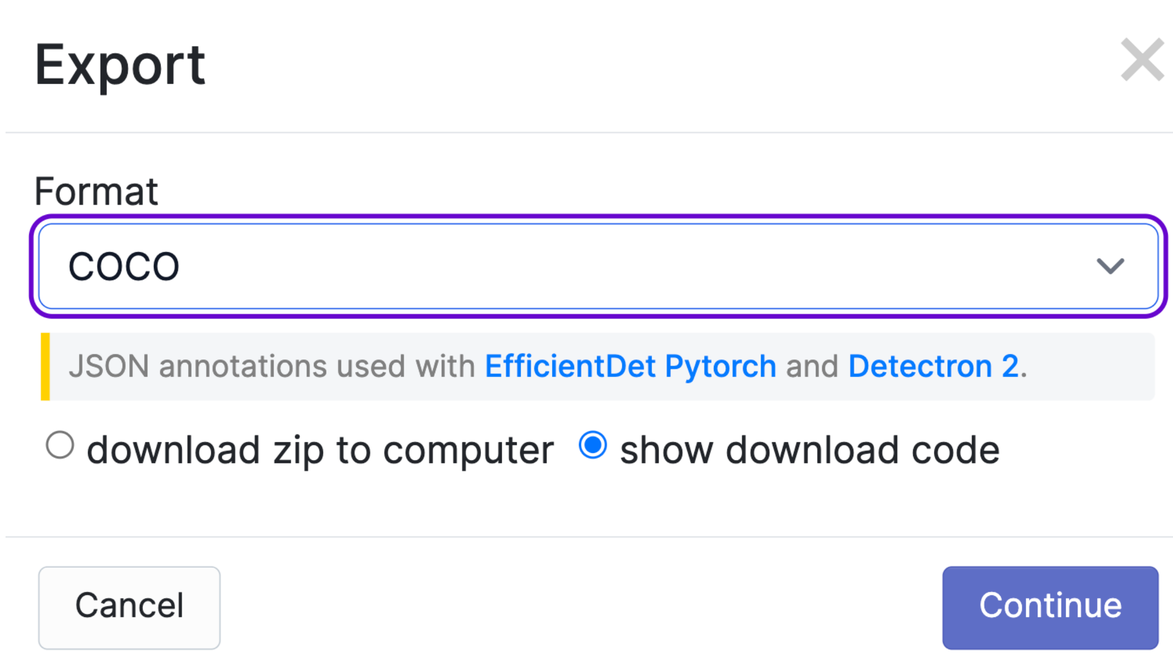
Version Name: V.9

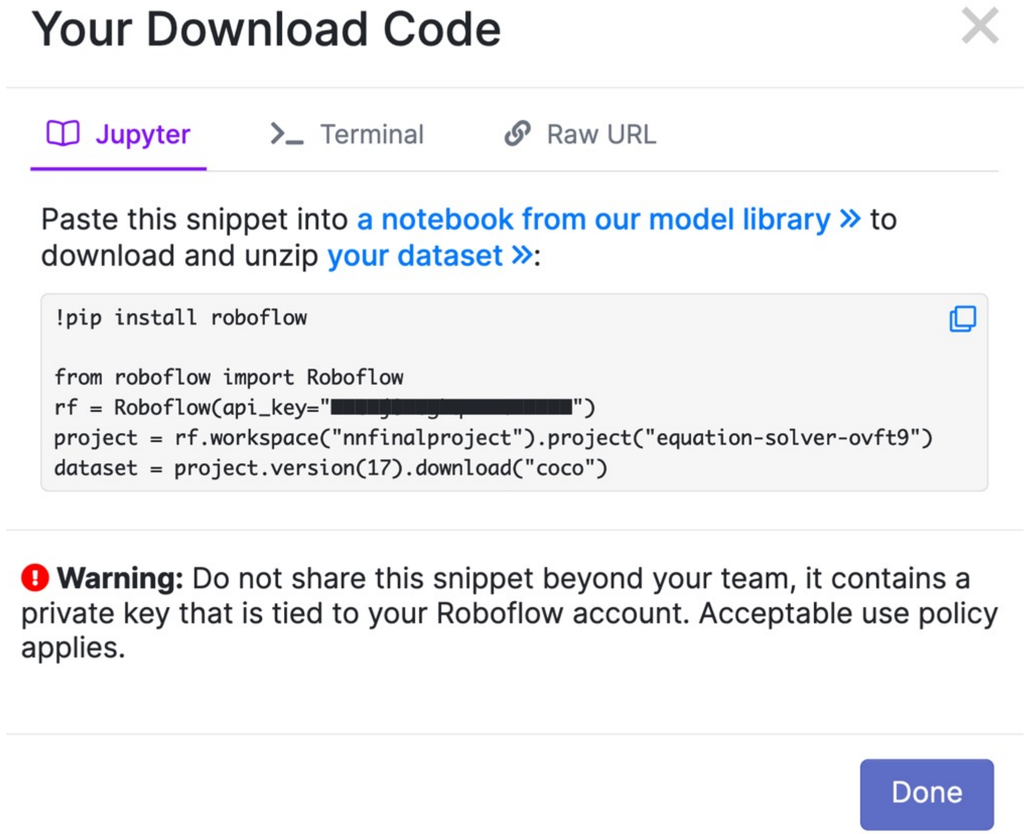
Version ID: 17

Generated: Jan 27, 2023

Annotation Group: Digits

The latest and final version produced from the dataset





After producing a version of the dataset, it is time to output it in the desired format and shape. To output from this step and start training the model with Detector 2, we considered the output to be Json Coco. Then we have to enter the download link of the dataset into the project to download the dataset to continue the work in the project, which includes separated images of training, testing, and validation along with the position and class of character labels in a separate file next to the images.

After running the code that we received as an output from Roboflow, the files related to the dataset are downloaded and added to the project.

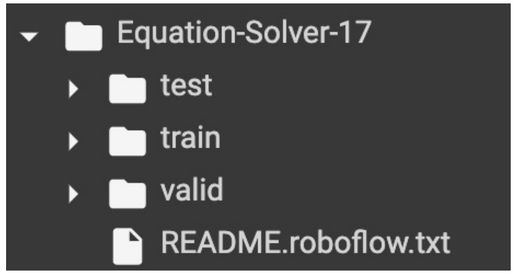
Install Roboflow (Annotating Tool) and add custom dataset

Dataset contains 1, 3, 5, +, -

19s

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="PXcvjOAegkqHxwtTwmYF")
project = rf.workspace("nnfinalproject").project("equation-solver-ovft9")
dataset = project.version(17).download("coco")
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting roboflow
  Downloading roboflow-0.2.29-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.0/49.0 KB 7.3 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from roboflow) (1.15.0)
Collecting cycler==0.10.0
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
```



Implementation method

Mounting Google Drive!

```
[1]  from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Install Roboflow (Annotating Tool) and add custom dataset

Dataset contains 1, 3, 5, +, -

```
[2]  !pip install roboflow

     from roboflow import Roboflow
     rf = Roboflow(api_key="PXcvjOAegkqHxwtTwmYF")
     project = rf.workspace("nnfinalproject").project("equation-solver-ovft9")
     dataset = project.version(17).download("coco")
```

```
Collecting cycler==0.10.0
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Collecting pyparsing==2.4.7
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 KB 9.9 MB/s eta 0:00:00
```

Mounting Google Drive to save the model.

Entering the dataset link into the project and downloading it to train the model.

Install Dependencies of Detectron2

```
[3]  # install dependencies: (use cu101 because colab has CUDA 10.1)
     !pip install -U torch==1.5 torchvision==0.6 -f https://download.pytorch.org/whl/cu101/torch_s
     !pip install cython pyyaml==5.1
     !pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
     import torch, torchvision
     print(torch.__version__, torch.cuda.is_available())
     !gcc --version
```

download the utilities needed to install Doctoron 2.

There is an inconsistency between the version of Coda installed on the Colab and the detector's requirement, which we fixed.

Install Detectron2 (Object Detection Tool)

```
     !pip install detectron2==0.1.3 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torc
```

install Detectoron 2.

Import Necessary Libraries

```
[5]  import numpy as np
     import cv2
     import random
     from google.colab.patches import cv2_imshow
     import matplotlib.pyplot as plt
     import os

     import detectron2
     from detectron2.utils.logger import setup_logger
     setup_logger()

     # import some common detectron2 utilities
     from detectron2 import model_zoo
     from detectron2.engine import DefaultPredictor
     from detectron2.config import get_cfg
     from detectron2.utils.visualizer import Visualizer
     from detectron2.data import MetadataCatalog
     from detectron2.data.catalog import DatasetCatalog
```

Register Train/Validation/Test Dataset for Detectron2! from Roboflow custom dataset

```
[6]  from detectron2.data.datasets import register_coco_instances

     register_coco_instances("my_dataset_train", {}, "Equation-Solver-17/train/_annotations.coco.json", "Equation-Solver-17/train"
     register_coco_instances("my_dataset_val", {}, "Equation-Solver-17/valid/_annotations.coco.json", "Equation-Solver-17/valid")
     register_coco_instances("my_dataset_test", {}, "Equation-Solver-17/test/_annotations.coco.json", "Equation-Solver-17/test")
```

importing the libraries needed to solve the question into the project.

Libraries like:

Numpy

CV2

Googlecolab

cv2_imshow

Matplotlib

Os

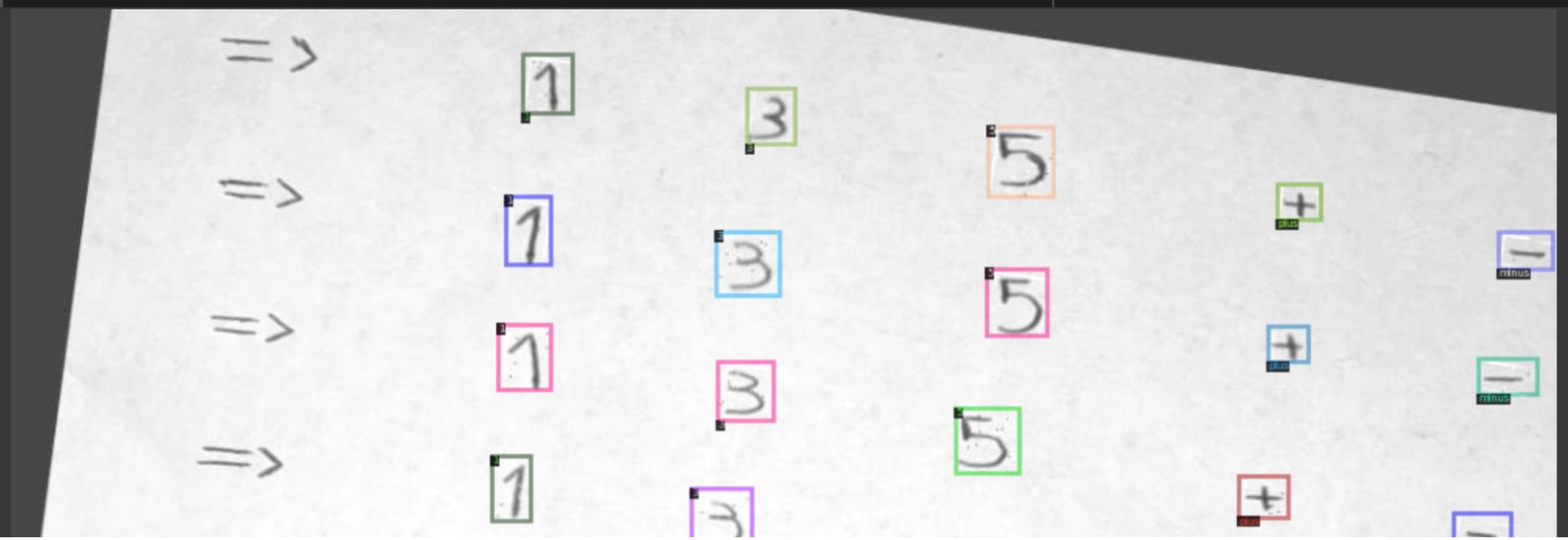
Detectron2

And…

Now we enter the address of the data downloaded from Roboflow to Detectoron in 3 parts of training data - validation and test so that it can identify them.

Visualize training data

```
my_dataset_train_metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")

import random
from detectron2.utils.visualizer import Visualizer

for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=1)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
```



We entered the given images along with the JSON file of the positions of the boxes into the detector. We display them in the form of 3 random images to make sure that everything is entered correctly in the doctor.

Setup Train Configuration

```
[ ] cfg = get_cfg()
    cfg.OUTPUT_DIR = "drive/MyDrive/Equation_solver"
    cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))
    cfg.DATASETS.TRAIN = ("my_dataset_train",)
    cfg.DATASETS.TEST = ("my_dataset_val",)

    cfg.DATALOADER.NUM_WORKERS = 4
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml")
    cfg.SOLVER.IMS_PER_BATCH = 4
    cfg.SOLVER.BASE_LR = 0.02


    cfg.SOLVER.WARMUP_ITERS = 1000
    cfg.SOLVER.MAX_ITER = 1600
    cfg.SOLVER.STEPS = (1000, 1600)
    cfg.SOLVER.GAMMA = 0.05

    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6

    cfg.TEST.EVAL_PERIOD = 100
```

meanwhile, we consider the distance between the boxes and the generated clusters to be the shortest distance between the boxes and the clusters.

We change the saved address of the model to save the model inside the drive to continue the project so that we don't need to train the model every time.

We use RCNN faster neural network for Object Detection.

We set the desired configuration to train the model.

A number of 4 images are entered into the training process by 4 parallel threads.

Learning rate: 0.02

The repetitions of the training process are done from 1000 to 1600 as needed

We considered batch size 64.

Validation is done after 100 repetitions.

Load Train Engine of Detectron2

```
from detectron2.engine import DefaultTrainer
from detectron2.evaluation import COCOEvaluator

class CocoTrainer(DefaultTrainer):

  @classmethod
  def build_evaluator(cls, cfg, dataset_name, output_folder=None):

    if output_folder is None:
        os.makedirs("coco_eval", exist_ok=True)
        output_folder = "coco_eval"

    return COCOEvaluator(dataset_name, cfg, False, output_folder)
```

Train Model

```
[ ] !rm -r output
```

```
rm: cannot remove 'output': No such file or directory
```

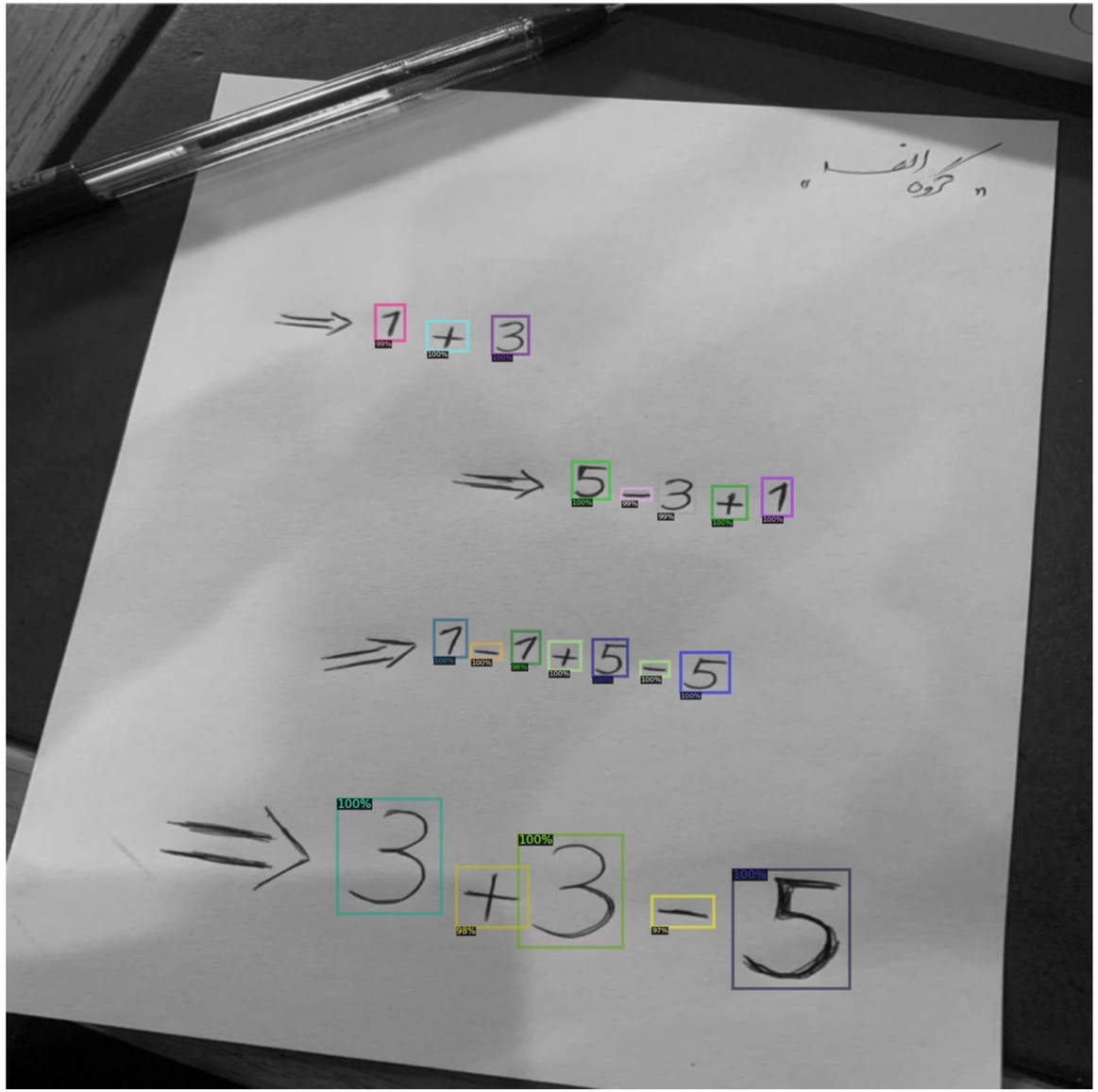
```
[ ] os.makedirs(cfg.OUTPUT_DIR)
    trainer = CocoTrainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()
```

Load the Detectron 2 training engine and start training according to those configurations.

Evaluate/Test Model

```
[ ]  cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
     cfg.DATASETS.TEST = ("my_dataset_test", )
     cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.9  # set the testing threshold for this model
     predictor = DefaultPredictor(cfg)
     test_metadata = MetadataCatalog.get("my_dataset_test")

     from detectron2.utils.visualizer import ColorMode
     import glob

     for imageName in glob.glob('Equation-Solver-17/test/*jpg'):
       im = cv2.imread(imageName)
       outputs = predictor(im)
       v = Visualizer(im[:, :, ::-1],
                      metadata=test_metadata,
                      scale=2
                       )
       out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
       cv2_imshow(out.get_image()[:, :, ::-1])
```

We test the trained model on the image and test data to see if it correctly recognizes the boxes and characters or not. We can consider a threshold limit to display boxes with different possibilities. Here we consider 0.9.



We can see that it has recognized the boxes and characters of the test image correctly and with high accuracy.

Stage 1 : Removing Boxes on a single char

if 2 boxes are at same position, we remove the box with lower score(probability)

Get Information of Predicted Boxes!

```
[ ]  instances = outputs["instances"]
     detected_class_indexes = instances.pred_classes.cpu().numpy()
     detected_boxes = instances.pred_boxes.tensor.cpu().numpy()
     detected_scores = instances.scores.cpu().numpy()

     # example box1 :
     print(detected_class_indexes[0])
     print(detected_boxes[0])
     print(detected_scores[0])
```

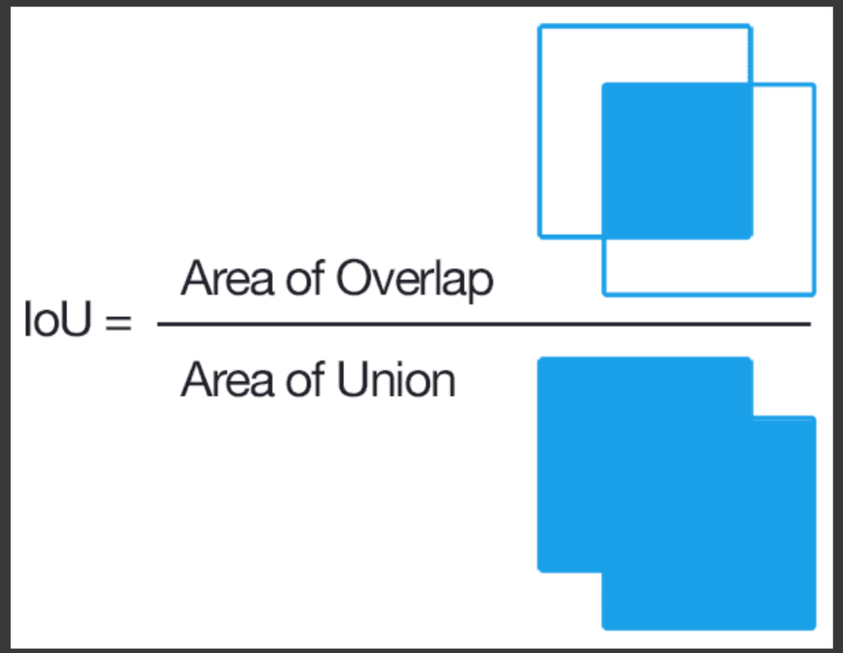
```
3
[520.86865 423.16602 555.6429  457.57733]
0.99968815
```

Now we will continue the project. We saved the trained model and now we save the predicted boxes along with their class and position along with their probability in variables to use them.

For example, you can see the information in the first box.

Calculate boxIOU!

Boxes Similarity measurements




```python
def boxIOU(box1, box2):

  b1x1, b1y1, b1x2, b1y2 = box1
  b2x1, b2y1, b2x2, b2y2 = box2

# intersection
  innerx1 = max(b1x1, b2x1)
  innerx2 = min(b1x2, b2x2)

  innery1 = max(b1y1, b2y1)
  innery2 = min(b1y2, b2y2)

  intersection = max(0, (innerx2 - innerx1)) * max(0, (innery2 - innery1))

# union
  aera1 = (b1x2-b1x1) * (b1y2-b1y1)
  aera2 = (b2x2-b2x1) * (b2y2-b2y1)

  union = aera1 + aera2 - intersection

  return intersection / union
```

In the first step, we have to remove the boxes that are randomly placed on top of each other and all of them show the same character and keep the one that has the highest probability as the main box of that character. We use the IOU similarity criterion to detect the similarity of boxes that are close to each other.

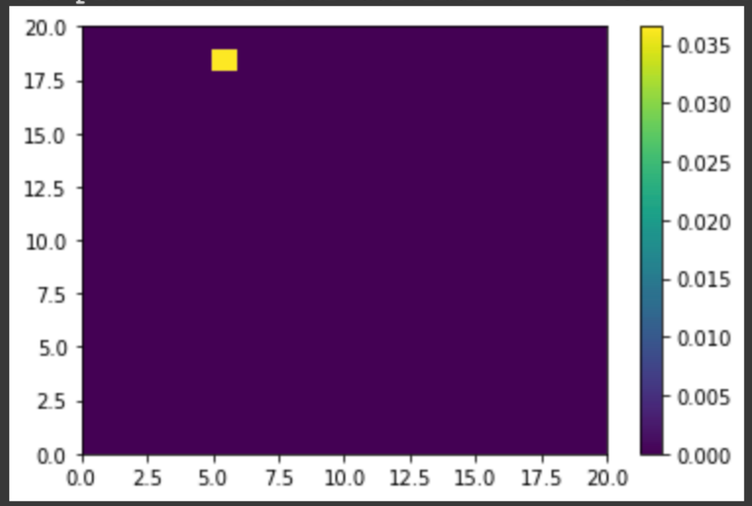
For example, in such boxes, one of them should remain with more probability.

These two boxes share a lot of IOU and it is higher than our threshold value which is 0.3. As a result, the box remains with a probability of 100%, and the other box is deleted.

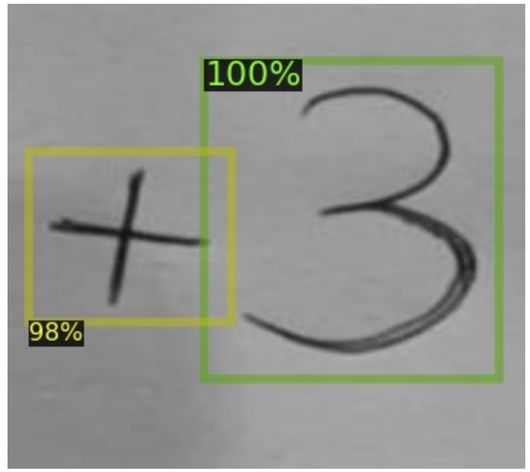
```
[12] n_boxes = len(detected_boxes)

     IOU_mat = np.zeros((n_boxes,n_boxes))

     for i, box1 in enumerate(detected_boxes):
       for j, box2 in enumerate(detected_boxes):
           if i <= j :
             continue
           IOU_mat[i,j] = boxIOU(box1, box2)
```

```
[14] print(IOU_mat[18,5])
     plt.pcolormesh(IOU_mat)
     plt.colorbar()
```

```
0.03655587509274483
<matplotlib.colorbar.Colorbar at 0x7f51821ab4c0>
```



We calculate the similarity of the boxes inside the image and store it in the IOU_mat matrix. As you can see, only two boxes 18 and 5 are similar by 0.03, but this amount is not so much that we merge them. We considered the integration threshold limit to be 0.3.



```
[15] merge_th = 0.3
     merge_indices = np.where(IOU_mat > merge_th)
     remove_indices = []

     for index in zip(merge_indices[0], merge_indices[1]):
       print(index)
       index1, index2 = index
       print(detected_scores[index1], detected_scores[index2])

       if detected_scores[index1] > detected_scores[index2]:
         remove_indices.append(index2)
       else:
         remove_indices.append(index1)
```

```
[16] sorted([remove_indices], reverse=True)

     for index in sorted([remove_indices], reverse=True):

       detected_class_indexes = np.delete(detected_class_indexes, index, axis = 0)
       detected_boxes = np.delete(detected_boxes, index, axis = 0)
       detected_scores = np.delete(detected_scores, index, axis = 0)

     remove_indices = []
```

```
[17] n_boxes = len(detected_boxes)

     IOU_mat = np.zeros((n_boxes,n_boxes))

     for i, box1 in enumerate(detected_boxes):
       for j, box2 in enumerate(detected_boxes):
           if i <= j :
             continue
           IOU_mat[i,j] = boxIOU(box1, box2)
```

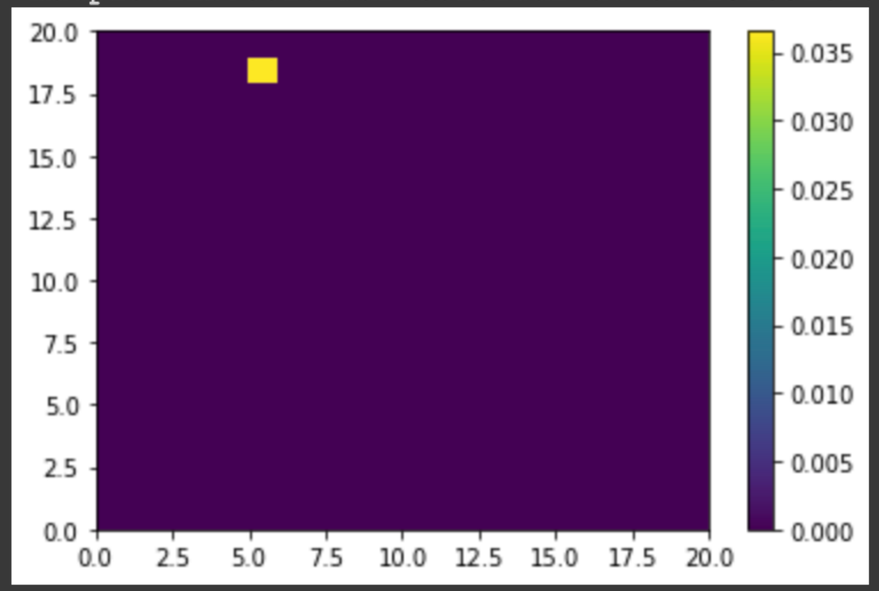
We considered the integration threshold limit to be 0.3. And we consider the boxes whose similarity is more than this value, and among those two boxes, the one with the least probability is put in the deletion list.

Then we have to remove them from the end of the list of predicted information from the boxes.

Then we draw the IOU_mat matrix again for further review.

```
[18] print(IOU_mat[18,5])
     plt.pcolormesh(IOU_mat)
     plt.colorbar()
```

```
0.036555875092744 83
<matplotlib.colorbar.Colorbar at 0x7f518528f610>
```



Then we draw the IOU_mat matrix again for further review.

Stage 2 : Find Equations

```
[18] def y_IOU(box1, box2):

       b1x1, b1y1, b1x2, b1y2 = box1
       b2x1, b2y1, b2x2, b2y2 = box2


     # intersection


       innery1 = max(b1y1, b2y1)
       innery2 = min(b1y2, b2y2)

       intersection =  max(0, (innery2 - innery1))

     # union
       h1 = b1y2 - b1y1
       h2 = b2y2 - b2y1

       union = h1 + h2 - intersection

       return intersection / union
```

In the second step, it is time to calculate y_IOU, a measure that checks how similar the boxes are to each other in a y dimension. In other words, it checks which of the boxes are on a line.

Find central distance beteen two boxes

```
[19] def x_distance_right(box1, box2):

         b1x1, b1y1, b1x2, b1y2 = box1
         b2x1, b2y1, b2x2, b2y2 = box2

         c1 = (b1x1 + b1x2) / 2
         c2 = (b2x1 + b2x2) / 2

         return np.abs(c2 - c1)
```

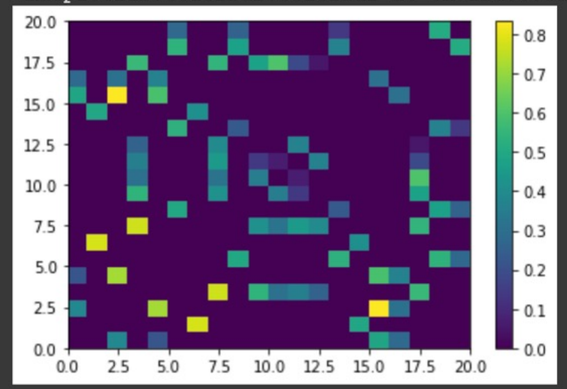
Using this function, we calculate the distance on the x dimension between both boxes. The distances are calculated from the center of the two.

Calculate yIOU_mat and central distances between all boxes

```
[20] n_boxes = len(detected_boxes)

     yIOU_mat = np.zeros((n_boxes, n_boxes))
     distance_mat = np.ones((n_boxes, n_boxes)) * 200

     for i, b1 in enumerate(detected_boxes):
         for j, b2 in enumerate(detected_boxes):
             if i <= j:
                 continue
             distance_mat[i, j] = x_distance_right(b1, b2)
             distance_mat[j, i] = x_distance_right(b1, b2)

             yIOU_mat[i, j] = y_IOU(b1, b2)
             yIOU_mat[j, i] = y_IOU(b1, b2)
```

```
[21] plt.pcolormesh(yIOU_mat)
     plt.colorbar()
```

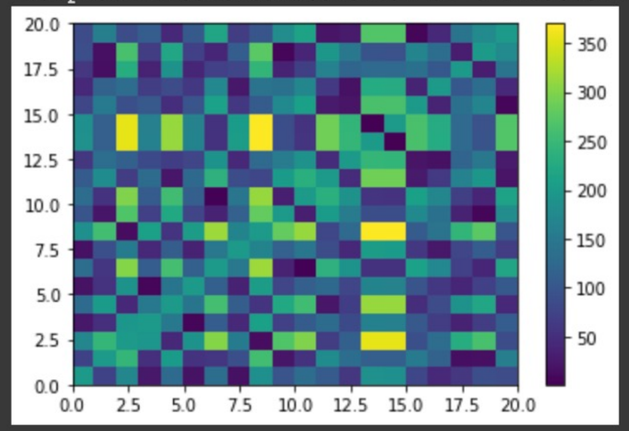
```
<matplotlib.colorbar.Colorbar at 0x7fba806cadf0>
```



We complete the y_IOU matrix and calculate this measure for both boxes and store it inside the overall y_IOU matrix for further checks.

```
[22] plt.pcolormesh(distance_mat)
     plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7fba805f13a0>
```



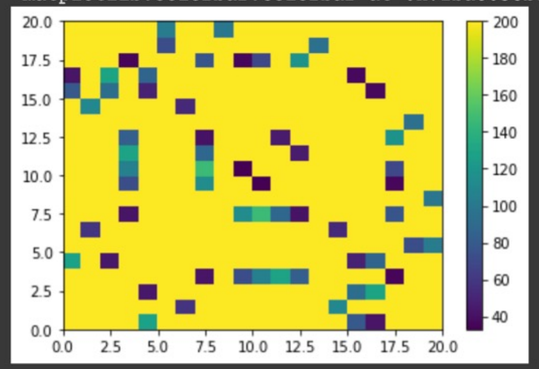
We also calculate the distance between both boxes and store it in the corresponding matrix like the y_IOU matrix.

Then we set the similarity of the boxes that are on the same line and their x-axis distance is greater than the threshold of 150 to 0. Because it is possible that several equations are written on the same line and we must separate them from each other and not consider similarities.

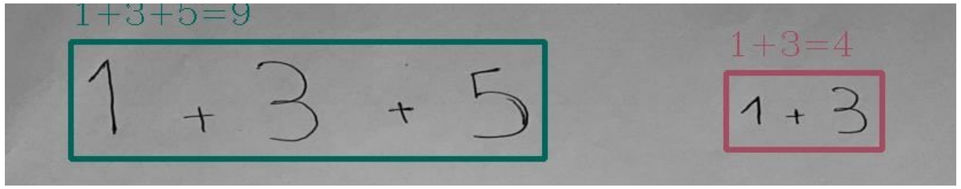
```
[23] distance_th = 150

     yIOU_mat[distance_mat > distance_th] = 0
     distance_mat[yIOU_mat == 0] = 200
```

```
     plt.pcolormesh(distance_mat)
     plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7fba8058bd90>
```



As :



Then we consider the distance of the boxes that have nothing similar to each other as a high value like 200, which means that these two boxes do not belong to the same equation.

Find Equations by hierarchical clustering

```
[25] from sklearn.cluster import AgglomerativeClustering
     import random
```

```
[26] clu = AgglomerativeClustering(n_clusters=None,
                                   distance_threshold=150,
                                   affinity='precomputed',
                                   linkage='single',
                                   compute_full_tree=True)
```

```
[27] clu_labels = clu.fit_predict(distance_mat)
```

```
equations_indices = []
res_image = im.copy()

for l in range(clu_labels.max() + 1):

    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)

    equations_indices.append(list(np.where(clu_labels == l)[0]))

    for idx in np.where(clu_labels == l)[0]:
        x1, y1, x2, y2 = (detected_boxes[idx]).astype(int)
        cv2.rectangle(res_image, (x1, y1), (x2, y2), (r, g, b), 3)

plt.imshow(res_image)
```

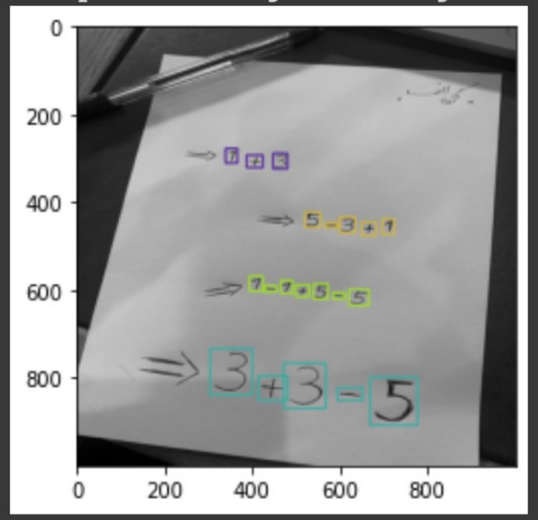
Then we used hierarchical-type clustering to find and distinguish the equations.

Since we do not know the number of clusters and equations, we set the related parameter to None and set the box clustering criterion to the distance between the boxes, and since we have already calculated the distance matrix and the data We do not give the main to the algorithm. We considered the pre-computed parameter.

Meanwhile, we considered the distance between the boxes and the formed clusters as a Single link, so that the shortest distance between the boxes and the clusters is the criterion for their integration.

Then, after clustering and observing the labels of the clusters, we draw them with fixed colors for each cluster.

```
<matplotlib.image.AxesImage at 0x7fb9f20ada60>
```




```
[29] print(equations_indices)

[[5, 8, 13, 18, 19], [1, 6, 14], [3, 7, 9, 10, 11, 12, 17], [0, 2, 4, 15, 16]]
```

Then, after clustering and observing the labels of the clusters, we draw them with fixed colors for each cluster.

The number of boxes that belong to a cluster is stored in the corresponding list of the same cluster.

Calculate Equation!

```
label_to_str = [None, 1, 3, 5, '-', '+']

def calculate_equation(equation_indices,
                       detected_class_indexes,
                       prediction_boxes,
                       box_margin=10):

    def sorter(idx):
        # we want sort this by x1
        return detected_boxes[idx][0]

    equation_indices = sorted(equation_indices, key=sorter)

    eq = ''
    result = 0
    add_next = True
    sub_next = False

    for idx in equation_indices:

        detected_symbol = label_to_str[detected_class_indexes[idx]]
        eq = eq + str(detected_symbol)

        if detected_symbol == '+':
            add_next = True
            sub_next = False
        elif detected_symbol == '-':
            add_next = False
            sub_next = True
        else:
            if add_next:
                result = result + detected_symbol
                add_next = False
                sub_next = False
            elif sub_next:
                result = result - detected_symbol
                add_next = False
                sub_next = False

    x1 = int(prediction_boxes[equation_indices][:, 0].min() - box_margin)
    y1 = int(prediction_boxes[equation_indices][:, 1].min() - box_margin)
    x2 = int(prediction_boxes[equation_indices][:, 2].max() + box_margin)
    y2 = int(prediction_boxes[equation_indices][:, 3].max() + box_margin)

    return eq, result, (x1, y1, x2, y2)
```

Then, to convert the boxes into real equations, we need to convert the numbers of the classes of the predicted boxes into the real numbers and characters of the question, and we do this with this array.

At first, we arrange the boxes with respect to the x-axis to find out which box comes after the previous box, in order to respect the order of the characters of the equation.

If we reach the + character in the equation, we raise the + flag and we also do the same for the - character.

Then, according to whether we saw + or - in the equation, we calculate the equation according to the current character and store the result in the result.

Then, using the minimum and maximum x, and y between the boxes of each equation and applying a distance limit in the function input, we consider a rectangle between each equation.

```
[31] calculate_equation(equations_indices[0],
                        detected_class_indexes,
                        detected_boxes)

('3+3-5', 1, (294, 723, 786, 917))
```

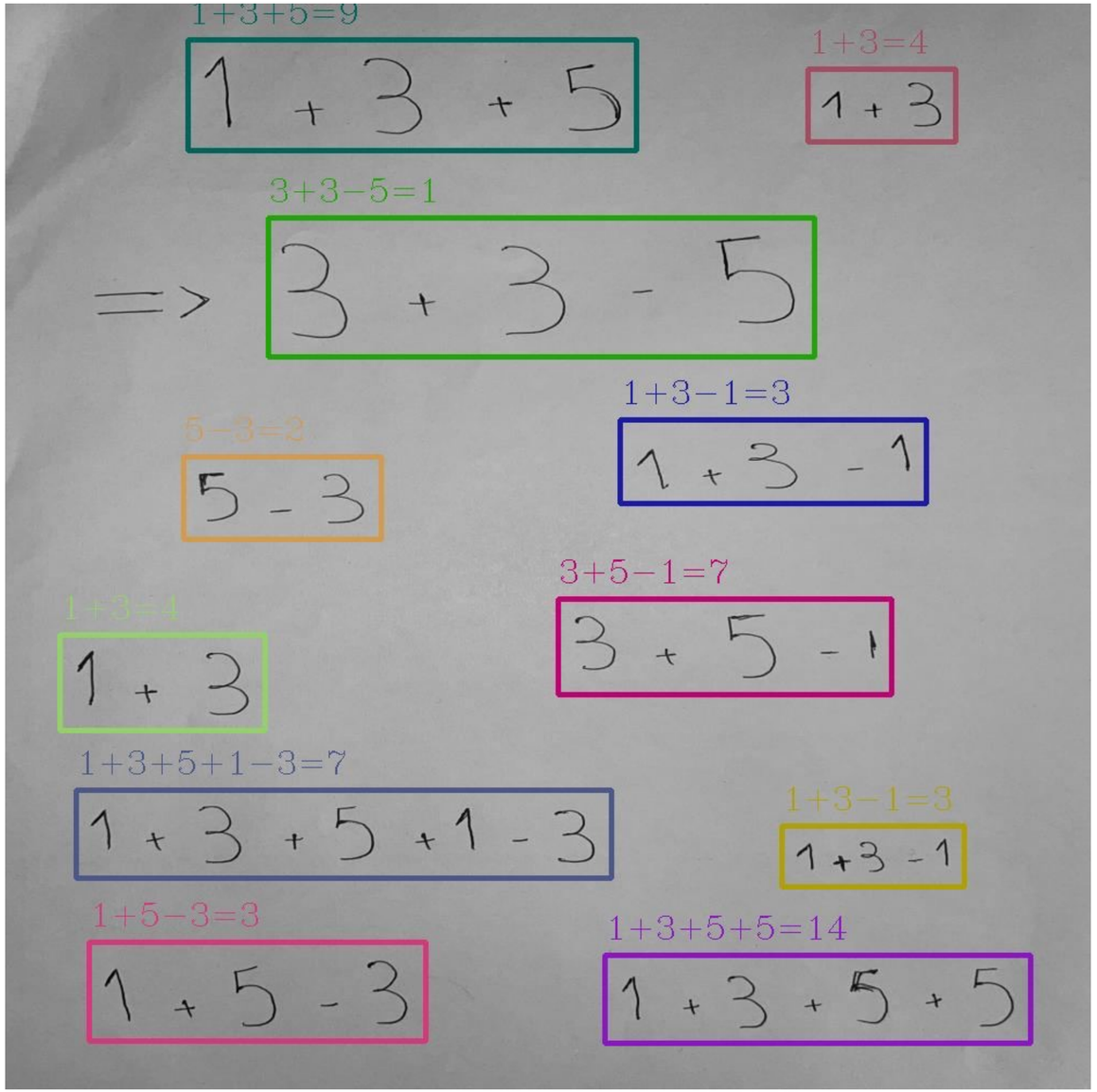
Finally, in the function output, we return the identified equation along with the calculated solution and the location of the equation.

For example, you can see the execution of the function for the first equivalent:

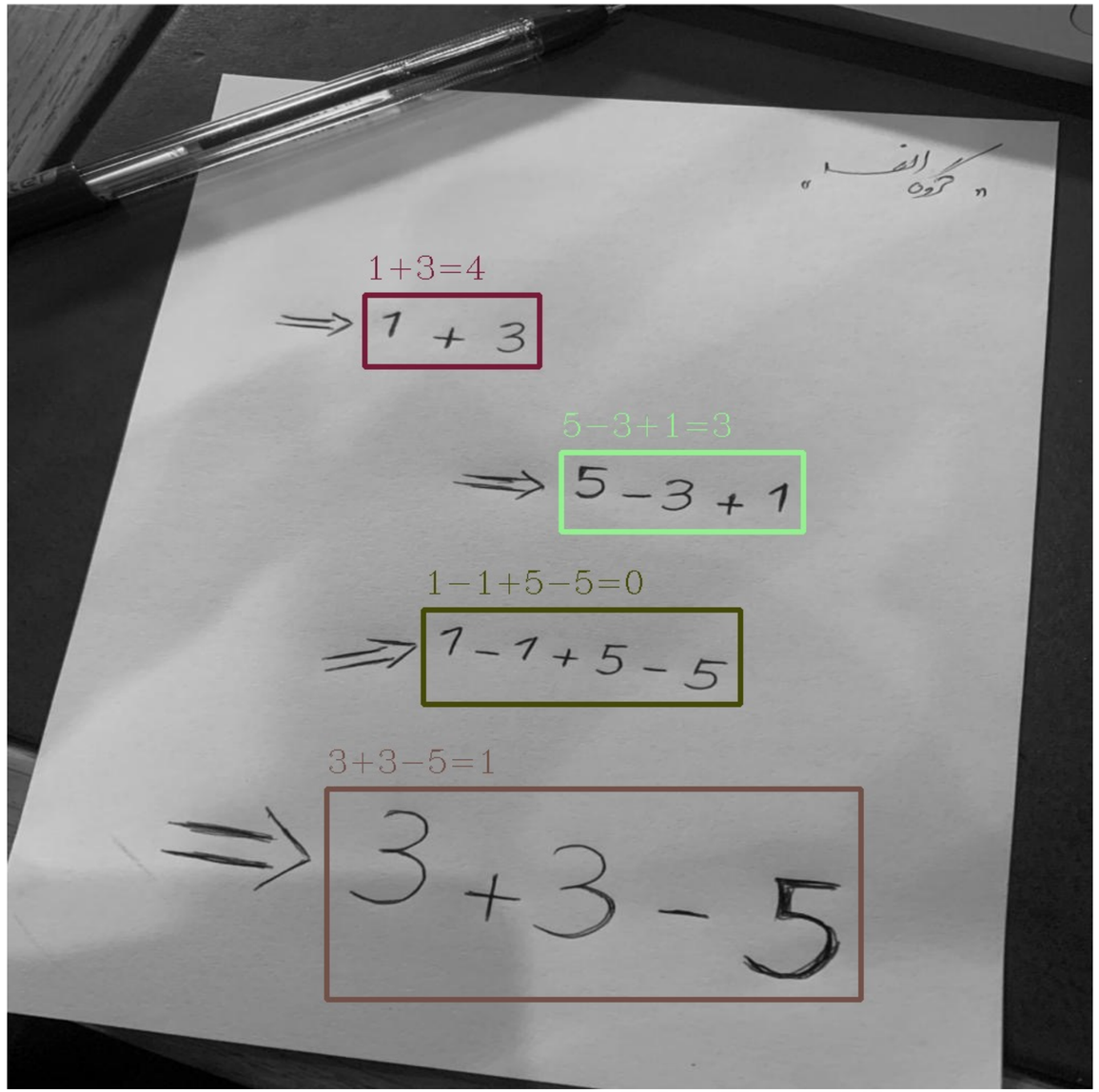
Final Result

```
res_image = im.copy()

for l in range(clu_labels.max() + 1):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(max(0, 127 - r - g), 255)

    eq, res, box = calculate_equation(equations_indices[l],
                                      detected_class_indexes,
                                      detected_boxes)

    cv2.rectangle(res_image, (box[0], box[1]), (box[2], box[3]), (r, g, b), 3)
    eq = eq + f'={res}'
    cv2.putText(res_image, eq, (box[0], box[1] - 15), cv2.FONT_HERSHEY_COMPLEX, 1, (r, g, b), 1)

cv2_imshow(res_image)
```

Finally, we execute this function for all equations and draw them according to the location returned from the function related to the calculation of equations. A fixed color is used for each equation.



The output image of the sample sheet



The output image of the test sheet