Using ML to Identify Poison Ivy from Look-alikes

Michael Silva

Abstract

This research project set out to develop a machine learning model that detects poison ivy, that could be embeded in a mobile app. Two-hundred and fifty images of poison ivy and three look-alike plant species (box elder, brambles and Virginia creeper) from the iNaturalist project served as the base dataset. Additional data created using data augmentation techniques were used to train the model. YOLO version three and five trained on these data. Evaluation metrics indicate strong performance in detecting poison ivy but hinted at overfitting. A simulated "field test" gauged how well the models generalized to one hundred images not used in the training process. Model performance on the field test lagged behind the levels seen during training, however the models identified poison ivy very reliably. Based on the field test, the YOLO version five model with 1,000 images per plant species preformed the best but additional work could improve the model's performance.

Keywords: Object Detection, YOLO, Neural Networks, Plant Identification, Poison Ivy

Introduction

Research Motivation

Data science should solve problems that improve the human condition. This capstone project sets out to solve a problem faced by my family. In the wake of the COVID-19 pandemic, we have spent more time outdoors hiking. While hiking, I often remind my children to be careful of the poison ivy which is often growing near the edge of the hiking trail. This reminder results in my children repeatedly pointing to plants and asking "is this poison ivy?" Given the recent increase in computing power available to humanity through smartphones, is it possible to develop a machine learning model, that could be embeded in an app, that could help people to identify poison ivy?

Human Detection Methods

Poison ivy is an important plant for people to identify. Identification can prevent exposure to the urushiol in poison ivy, and the associated rashes and discomfort. The adage "leaves of three, let it be" is a common refrain used to teach how to identify poison ivy. Poison Ivy terminates in a central leaf and two lateral leaves. Poison ivy is also recognizable because the lateral leaves often have a mitten-like shape. These two characteristics are the most often used to identify poison ivy, although other methods do exist.

Project Requirements

In order to direct this project, I imposed the following project requirements:

- The developed solution must use images from smartphone as input;
- The machine learning model should (ideally) be embedded in the app, although a server hosted model that interacts with an app would be an acceptable fallback; and
- The model should be accurate and robust at identifying poison ivy

Developing the app is not a requirement at this stage, but developing the machine learning model should be done with this end goal in mind.

Literature Review

For this project, I focused the review of the literature on how to approach this task. This section details the questions I asked, what I gathered from the research, and how this knowledge informed the solution this project pursued.

Research on Datasets

What part of the plant should I focus on?

Identifying a plant based on an image has long been a pursuit of computer vision researchers. Researchers in this field have primarily focused on leaves. The leaves of a plant can easily be collected, preserved, and imaged due to their planar geometry (Wäldchen and Mäder 2017). Also, their availability throughout most of the year makes leaves ideally suited for plant recognition (Wäldchen, Rzanny, et al. 2018). I decided for these reasons I too would focus on the leaves of the plant in building my machine learning model. As previously discussed in this paper, using the leaves makes a lot of sense in the case of poison ivy as it is one of the distinguishing characteristics of the plant.

What data source should I use?

Having settled on using leaves for this task, I needed to find a data source. Computer vision researchers developed databases of leaves for their research such as the Flavia (Wu, et al. 2007), Herbarium (Gaurav, et al. 2006-08), and Cleared Leaves database (Das, et al. 2014). Research in the field often draw on one of these databases in evaluating a method as they provide a common dataset to measure performance. The images in these databases follow the long tradition of botanical field guides where the image of the leave is isolated on a plain white background. Working from the assumption that images captured by a smartphone will be used by the model to make predictions makes it clear that these databases are not well suited to the goals of this research project. The images in these databases are too far removed from a natural environment to serve as reliable training data for a machine learning model.

The images in the iNaturalist project are better suited to serve as the dataset. Participants in the iNaturalist project take pictures of organisms (plants and animals) in their natural state with the iNaturalist app. The image is uploaded to iNautralist.org, where a global community of naturalists crowdsource to help identify the organism (Nugent 2018). An observation reaches "research-grade" when more than two-thirds of two or more identifiers agree on the classification at a level lower than family (Hochmair, et al. 2020). Researchers have used the research-grade observations to examine biodiversity, monitor species, and detect plant diseases (Rossi 2017). An image classifier using the Inception architecture and iNaturalist images has also been developed (Cui, et al. 2018). The iNaturalist project has over 25 million records from every country representing more than 230,000 species, collected by over 700,000 people, and identified by 90,000 people (Seltzer 2019). This dataset is superior to all other datasets examined in this project.

Challenges/Opportunities with using the iNaturalist Dataset

Working with in-situ images introduces a few conditions that are significantly different than the leaf databases. Plants in the wild can be damaged by insects, pathogens, large herbivores, or the weather. Consequently, the plant featured in the image may be less than the "ideal specimen." Another condition that is very common in the wild is occultation, or part of the plant being hidden behind something else (i.e. a leaf, a branch, a stem, another plant, etc.). This challenge is not present in the traditional leaf databases.

Also, iNaturalist makes no strict protocol on how to photograph the specimen. Consequently, the orientation, angle, and closeness to the subject varies significantly. Lighting conditions also differ considerably. Some pictures rely on flash photography, while others rely solely on natural lighting. Some of the images are taken in less than ideal lighting. The scanned leaves databases follow more stringent protocols and offer more consistency in the image quality.

However, these conditions in the iNaturalist dataset are ideal for this research project. These images, with all their challenges, form a better representation of what the end user of an app would likely experience and photograph in the wild. It offers a higher level of fidelity than the scanned leaf databases.

Research on Techniques

How have others identified plants?

Having settled on a dataset, I needed to determine what technique to use. Plant identification is a difficult task. The challenge, in part, lies in the fact that there is high variability within a plant species. The size, color, and shape of the leaves are just a few of the possible differences observed within the same plant species. The high variability within a species coupled with the small differences between species makes plant identification such a challenge (Šulc and Matas 2017).

Researchers in this field have a long history of developing image classifiers to identify plants. Image classifiers developed by researchers using the leaf databases discussed in the previous section relied on significant preprocessing of the image such as isolating the leaf from any background, remove noise from the image, and enhancing contours or veins in the leaves, so the computer can extract meaningful features and identify the plant (Wu, et al. 2007) (Aakif and Khan 2015) (Jin, et al. 2015). This kind of pre-processing would not be a possibility in a mobile app.

That is not to say that it is not possible. Others researchers have been successful in developing image classifiers that run on mobile devices. Leafsnap (Kumar, et al. 2012) completes automatic visual recognition of a plant from an image. The developers claim their plant identification app can currently recognize 90% of all known species of plants and trees. Although they do not share the architecture of their model, it is doubtful they preprocess the input images.

Neither of these approaches are sufficient. The amount of preprocessing performed by the first group is too computationally intensive and not appropriate for the mobile platform. The image classifier like Leafsnap has a major limitation. Poison ivy often grows next to benign look-alikes. A user would have to get close to poison ivy to ensure it is the only plant in the picture, which is obviously undesirable.

The way to work around the problem of keeping the user away from poison ivy is to develop an object detector. An object detector solves two computer vision tasks: object localization and image classification. Object localization involves predicting a bounding box around one or more objects in an image. Once a bounding box is determined, the object detector classifies what is found in the bounding box. This approach is superior as it would be able to differentiate poison ivy from look-alikes without sacrificing the safety of the users.

What object detection technique should I use?

Having decided to use object detection I researched approaches. Early object detectors relied on a sliding window paradigm (Papageorgiou and Poggio 2000) (Felzenszwalb, et al. 2010) (Dollar, et al. 2014). The computer would pass a sliding window across the image and identify objects within the window. The object detector would make a second pass to classify the objects identified in the first past. This approach is still the dominant paradigm in object detection.

Deep convolutional neural networks (CNNs) recently advanced the field of object detection. Their regression techniques accurately localize objects based on deep features. The first successful application of CNNs to object detection is Regional Convolutional Neural Networks (R-CNN) (Girshick, Donahue, et al. 2014). One shortcoming of the R-CNN is that it is slow. Fast R-CNN (Girshick 2015) and Faster R-CNN (Ren, et al. 2015) improved on the approach.

Around the same time Faster R-CNN came out, a novel object detection paradigm debuted. It differed from all its predecessors as it combined the object localization and classification tasks. Since "You Only Look Once" at the image, the model is named YOLO. YOLO models achieved unparalleled speeds compared to other CNNs (Du 2018). Since its release in 2016 (Redmon, Divvala, et al. 2016), YOLO underwent further development with version two released in 2017 (Redmon and Farhadi 2017), version three in 2018 (Redmon and Farhadi 2018), and version four (Bochkovskiy, Wang and Liao 2020) and version five in 2020. Researchers use YOLO models to automatically recognize license plates (Hendry and Chen 2019), identify breast masses in mammograms (Al-masni, et al. 2018), judge the growth stage of apples in orchards (Tian, et al. 2019), and detect diseases and pests in tomato plants (Liu and Wang 2020). The near real-time object detection speeds make this approach the best approach for this research project.

Challenges/Opportunities with using YOLO

YOLO is built on the darknet architecture which is programmed in C. Models developed using darknet would require conversion into a format that works on a mobile device (i.e. TensorFlow Lite). The developers of YOLO version five built it on pytorch, which can be supported natively on mobile platforms.

Methodology

As stated in the introduction, this research project focuses on developing a machine learning model that can accurately and robustly detect poison ivy. Based on the research detailed in the preceding section, this model draws on images from the iNaturalist project and uses the YOLO family of object detection models. This section details:

- **Data Acquisition and Preparation**: Describing the decisions made on what data to include, and what transformations to make;
- **Training Platform**: Briefly describes the architecture used in training the models;
- **Model Variants**: Describes the experimentation of model type and size and quality of training data; and
- **Evaluation of Models**: Details the metrics used in evaluating the performance of the models both during the training process and with a simulated "field test."

Data Acquisition and Preparation

Since the data used in this project is a custom dataset, this section details its acquisition and preprocessing in preparation for training the object detector. It will also discuss the limitations of the study dataset.

Data Acquisition

I acquired data by scrapping the iNaturalist search API to get a listing of observations by plant species. Search results were processed to build an index of potential images. Once this index was produced, additional scrapping was done to collect the URL of the image and information about the record such as the location and the date the image was taken.

Plants Included in the Dataset

In order to robustly identify poison ivy, the model needed to be trained to differentiate from plants that look like poison ivy. I developed a list of thirteen plants commonly mistaken for poison ivy after consulting both academic (Cornell Cooperative Extension, Indiana State University Extension, Michigan State University Extension) and popular experts (Sunset magazine, the iNaturalist project, and gardening television show blogs). I ultimately selected three of the most frequently mentioned plants from this list. They are Virginia creeper, box elder seedlings, and brambles.

Figure 1. Example iNaturalist Images (Left to Right: Poison Ivy, Virginia Creeper, Bramble, Box Elder)

Virginia Creeper does not follow the "leaves of three" rule as it has five leaves. But like poison ivy, it is a vine plant prone to climb trees. Brambles terminate a stem with "leaves of three" and the two lateral leaves often have a mitten shape similar to poison ivy. They do tend to have thorns on their stems unlike poison ivy. Box elder seedlings also terminate in "leaves of three" and generally resemble poison ivy. All of these look-alikes and poison ivy are typically found in wooded areas.

Downloading images for this research project ran from in the end of September to the beginning of October 2020. In total over 32,000 images were downloaded for these four plant types. Not all of the images could be used. Some images lacked leaves because of the season they were taken in or they were not the subject in the photograph. Instead the photo features berries, or flowers, for vines, etc.

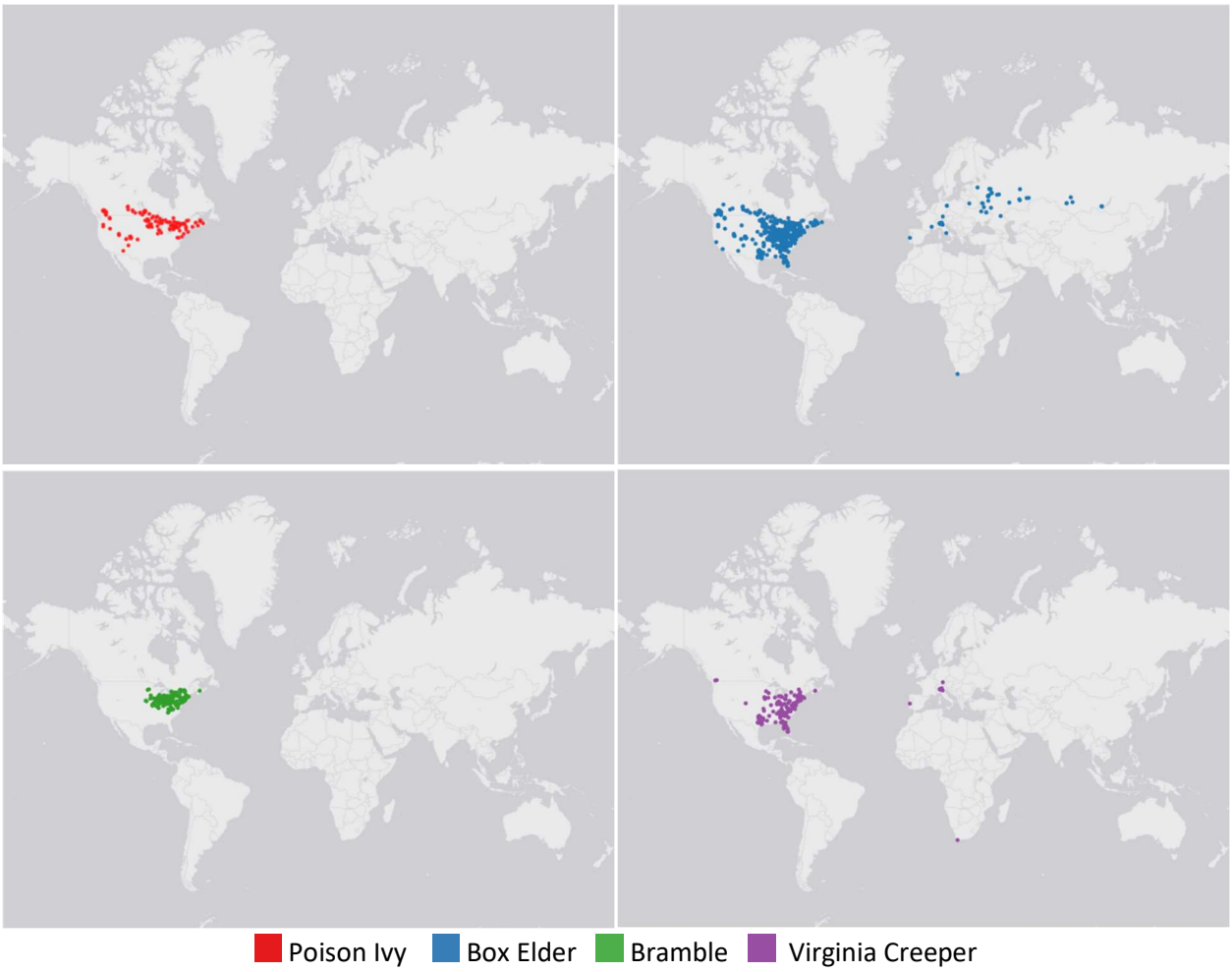
Limitations of the Dataset

As with any dataset, there are limitations with the dataset used in this project. First of all, this dataset assumes that the iNaturalist crowdsourcing process correctly classifies plants species.

Initially I applied my judgement, verifying that the image was truly poison ivy. I quickly became aware that by doing this I would bias the model to what I considered poison ivy. As I am not an expert, I abandoned this practice early in the process, and reviewed the images that I had passed over. Subsequently, I decided to wholly rely on the wisdom of the crowd. If they mistook a plant, the object detector trained on inaccurate data.

Another limitation of the study's dataset is it only represents certain geographical regions. Figure 2 illustrates the locations where the images were taken. While most of the observations occurred in the United States and Canada, there are some observations from other parts of the world. How well the model generalizes outside of these areas is unknown.

Figure 2. Location of Images by Plant Type

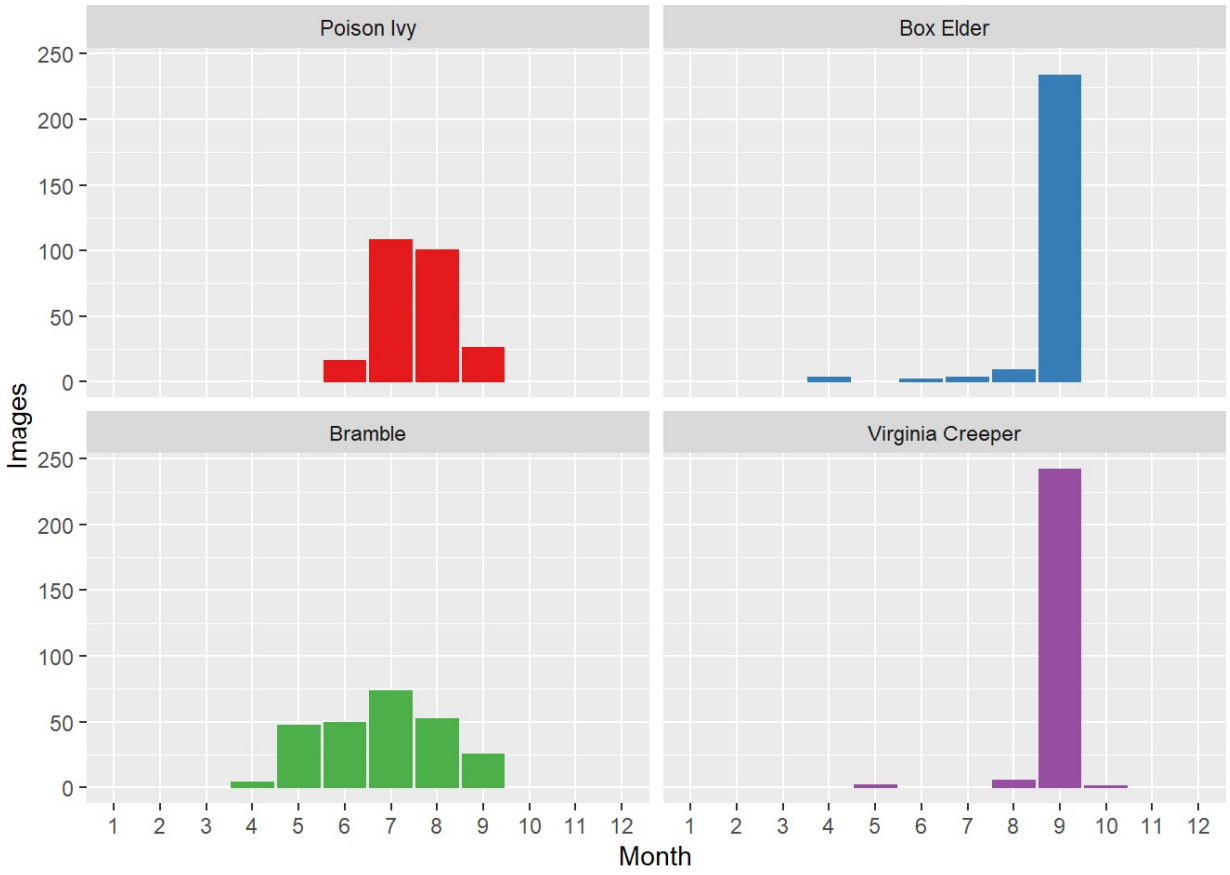


In addition to the geographical limitations, the images used in this dataset only capture a subset of all the months of the years. This is partly to be expected. Since the model is trained on the leaves and there are months where leaves are not present, some months would not be

represented in the dataset. One would not expect to see images from winter months, for example, especially given the temperate locations all the images came from.

Some months are overrepresented in the dataset. The box elder and Virginia creeper are dominated by photographs from September. This is probably due to the fact of when I collected the images from iNaturalist (which was September). It is unknown how well the model generalizes to months not covered by the datasets. Figure 3 illustrates the number of images by month.

Figure 3. Number of Images by Type of Plant and Month Photographed



Labeling the Objects

The iNaturalist images needed labeling prior to training. This is the process of drawing bounding boxes around the objects in each image. I labeled the images using the open-source tool LabelImg[1] using the YOLO specification. Building upon the "leaves of three" notion, the definition of an object used in this project is a cluster of leaves. Poison ivy is a vine plant that sends up leaf clusters along the length of the whole vine. By selecting a cluster of leaves as the

[1] Available online at: https://github.com/tzutalin/labelImg

object, the object detector would be trained to identify part of the plant and not the whole of it.

It also follows that a single image can have multiple objects, which was the case. Images in the dataset ranged from one to thirty-five objects per image. Certain plant species had more objects in their image than others as shown in Table 1. The images of brambles tended to have a lot more "plants" in the photograph than poison ivy.

Table 1. Objects in Images by Plant

Plant Type	Objects
Poison Ivy	1,069
Virginia Creeper	1,441
Bramble	2,063
Box Elder	1,232

Due to time-intensive nature of labeling data, I only labeled two hundred and fifty images of each plant type. Each plant had to have an equal number of images to prevent class imbalance. Class imbalances results in models biased towards the majority class, especially with deep learning models like an object detector (Johnson and Khoshgoftaar 2019). Consequently, efforts were made to ensure all classes were balanced.

Data Augmentation

Since object detection models perform best when trained on datasets with thousands of images, I decided to use data augmentation techniques to increase the size of the study's dataset. Researchers in the field commonly employ these techniques because labeled datasets are scarce and time-intensive to produce. In fact, commercial solutions, such as Roboflow, exist solely to help data scientists augment their labeled image data.

Common data augmentation techniques include transformations like rotating the image, performing horizontal or vertical flips, zooming, or cropping the image. Adding white noise (random black and white pixels) and varying the brightness are other techniques used to augment the data. By employing these techniques, researchers introduce variation into the data. Object detectors treat the augmented image as if it is a totally new image.

Through data augmentation the study produced trained on four sets data. The first set uses the original two hundred and fifty images per plant species with no augmentation. The second augments these images with a horizontal (mirrored) flip (resulting in five hundred images in total). The third preforms the horizontal flip, then creates duplicates of all the images with 5% noise, resulting in one thousand images. The final completes all the proceeding augmentations, duplicates all the images and adds 2% noise to the duplicates. This results in two thousand images per plant species. Figure 4 gives examples of the images after undergoing the augmentation techniques.

Figure 4. Examples of Study Image Augmentation Techniques

Original Image *Mirrored Image* *Salt and Pepper Noise Added*

The images did not undergo any rotations. This augmentation technique would simulate users rotating the orientation of their smartphone. It should be considered for future work.

YOLO Model Implementation

There are multiple implementations of the YOLO model. This project uses YOLO version five developed by Ultralytics. This study wanted to use this implementation because it is the latest iteration of the YOLO model. I wanted to compare the model against an earlier version to see how it stacked up against its predecessors.

However, Ultralytics implemented YOLO version five in Python. The earlier models developed by Redmon, and others building off of his work used the darknet architecture which is programed in C. In order to minimize the difference due to language I selected Ultralytics YOLO version three implementation which is also programmed in Python. All of the source code for the models are available on their respective GitHub pages[2].

Transfer Learning

There is one significant difference between the two YOLO models besides their version model. The YOLO version three models benefited from transfer learning. To understand what this means the reader must have a basic understanding of how an object classifier works.

Image data passes through the layers of a neural network. At each node of the network, the data passes through a filter. Each filter has a weight attached to it. When an object classifier starts from scratch, these weights are initialized with random values. Through the training process, the optimal values for these weights are learned using gradient descent to minimize regionalization and classification errors.

---

[2] See https://github.com/ultralytics/yolov3 and https://github.com/ultralytics/yolov5

Transfer learning initializes the weights of these filters with values from another object detector. It transfers the learning from one model to another. This method gives a model a jump on the training process.

As previously mentioned, the YOLO version three models benefited from transfer learning. They started with weight from a model trained on the famous COCO dataset (short for Common Objects in COntext), which trained on 80,000 training images and 40,000 validation images to identify eighty different classes of objects.

At the time of this research project, the YOLO version five model did not support transfer learning. The developers have since added this enhancement. But when these models trained the weights were initialized randomly.

Training Platform

All models were trained using Google Colab. Google Colab allows users access to cloud computing resources, including GPUs. Users work in a jupyter notebook style interface. The user has no control over what resources are provisioned to a Colab session. The user also has no control how other users are using shared computing resources. Initially training was attempted using the free tier, but would not complete successfully. Google Colab disconnects free tier users that use a lot of resources. A Google Colab Pro account provided the stability needed to train the object detection models. Colab provisioned either a NVIDIA Tesla V100-SXM2-16GB or P100-PCIE-16GB GPU for training the models in this study.

Model Variants

This research project trained eight different YOLO model variants. As explained earlier in this report, data augmentation resulted in datasets of four different sizes – two hundred and fifty, five hundred, one thousand, and two thousand images per plant species. YOLO version three and five models trained on each of these four datasets resulting in eight model variants. All version three models benefited from transfer learning, as previously noted. All other training parameters were identical (i.e. 80/20 training and test split, each model trained for three hundred epochs, etc.)[3]

Evaluation of Models

I employed standard performance metrics for object detectors in evaluating the models. This includes precision, recall, F_1, and mean average precision (mAP). These metrics indicate how well the object detector preforms its job. All of these metrics are for the test set images. In

[3] Training metrics are included in the appendix

addition to these metrics, I examined the training and test errors over the training epochs for evidence of overfitting.

Simulated Field Test

A field test would ideally measure the accuracy and robustness of the model. Unfortunately, the leaves had already fallen when I finalized training the models making a field test and impossibility. In lieu of an actual field test, I carried out a simulated field test. I selected one hundred images of plants unused in the training process. Twenty images per the four study plant species made up eighty of the images. Other plants commonly mistaken for poison ivy made up the remaining twenty. The simulated field test used the same performance metrics described in the preceding section. Although it is imperfect, it gives an opportunity to assess how well the models generalize.

Cost to Train the Model

I considered other factors besides the performance metrics during training and the simulated field test in evaluating the models. I estimated what the costs of training each model would be, using the current GPU pricing for Google's Cloud Engine. This cost contextualizes the performance metrics and can be used in evaluating the tradeoffs between models. As explained in the training platform section, the computing resources provisioned for the training process were outside of my control. The models were only trained once so the variation between training time is unknown.

Results and Discussion

Model Performance

Table 2. Evaluation Metrics on Test Data by Model

Method	Images	Precision	Recall	F_1	mAP@0.5
YOLO v3	250	79.2%	76.8%	78.0%	78.4%
YOLO v3	500	96.3%	96.7%	96.5%	97.4%
YOLO v3	1,000	99.3%	98.9%	99.1%	99.2%
YOLO v3	2,000	99.8%	99.7%	99.7%	99.5%
YOLO v5	250	59.2%	77.5%	67.1%	73.1%
YOLO v5	500	79.5%	96.8%	87.3%	97.6%
YOLO v5	1,000	93.8%	99.8%	96.7%	99.5%
YOLO v5	2,000	97.6%	99.9%	98.7%	99.5%

The performance metrics on the test data indicate all models performed well. The recall, precision, F_1 score, and mean average precision of different model variants at the end of the training process on the test data are shown in Table 2.

The mean average precision ranged from 73.1% to 99.5%. The precision ranges from 59.2% to 99.8% and recall ranges from 76.8% to 99.9%. As expected, the models trained on less data did not perform as well as those trained on more. The difference between the one thousand and two thousand image variants is negligible. The version three model appears to perform slightly

better than the version five model. This is probably because the version three models benefited from transfer learning, while the version five models did not.

There is some variation in performance between the plant species. Table 3 summarizes the mean average precision by the model variant and plant type. The model's ability to recognize poison ivy lags behind the other plant types until the model had a dataset of one thousand or more images. The fact that the model learned to identify Virginia creeper the fastest is not surprising. It is quite distinct from the other plants in the data set. These is not much difference in the performance of the YOLO version three and version five models.

Table 3. mAP@0.5 by Model and Plant Type

Class	v3 250	v3 500	v3 1,000	v3 2,000	v5 250	v5 500	v5 1,000	v5 2,000
All	78.4%	97.4%	99.2%	99.5%	73.1%	97.7%	99.5%	99.5%
Poison Ivy	68.0%	95.8%	99.5%	99.5%	61.5%	98.8%	99.5%	99.5%
Virginia Creeper	88.1%	98.9%	98.2%	99.5%	91.7%	98.8%	99.5%	99.5%
Bramble	82.9%	98.5%	99.5%	99.5%	75.2%	96.4%	99.5%	99.5%
Box Elder	74.6%	96.3%	99.5%	99.5%	63.9%	96.7%	99.5%	99.5%

Does the Models Overfit the Data?

Neural Networks have the tendency to overfit the data. By looking at the error on the training and test set over the training process it is possible to detect if the model is overfitting. If the error rates on the test data increase while it decreases on the training set, it indicates the object detector is overfitting the training data. The object detector tries to minimize two errors during training: the object detection and classification error. Figures 6 through 9 examine both these errors for the version three and version five YOLO models.

These figures indicate the models trained on two hundred fifty images overfit the data. This is not surprising. As previously discussed, two hundred fifty images are not enough to train a robust object detector. Both YOLO version three and five object detectors trained on the smallest dataset exhibit the same problem of overfitting.

Researchers often address overfitting by stopping the training process early. They tend to choose the point where the error on the test set is minimized. Figure 5 presents the mean average precision by the different model variants over the training epochs[4]. Mean average percentage did not improve much after the one hundred and fiftieth epoch. This suggests that training could be stopped around the one hundred and fiftieth epoch without much detriment to the model's performance. This would also decrease the amount of time needed to train the

[4] Additional visualizations of key performance metrics can be found in the appendix.

models. This may improve the model's performance. However, I believe more images would be more effective than stopping early.

Figure 5. mAP@0.5 by Method and Number of Images

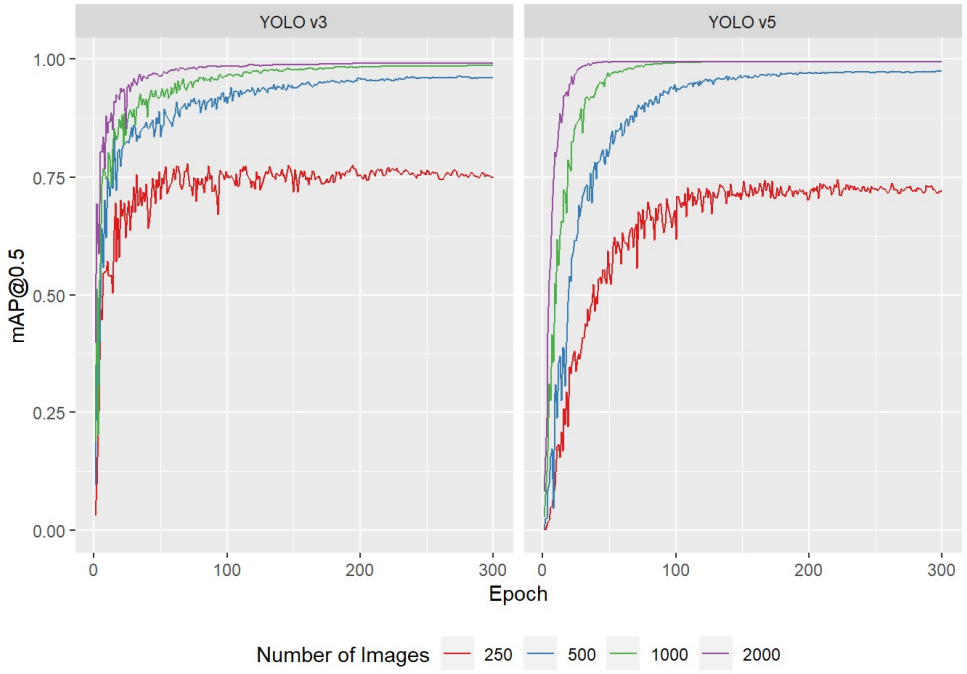


Figure 6. Object Detection Error of YOLO v3 Model Variants

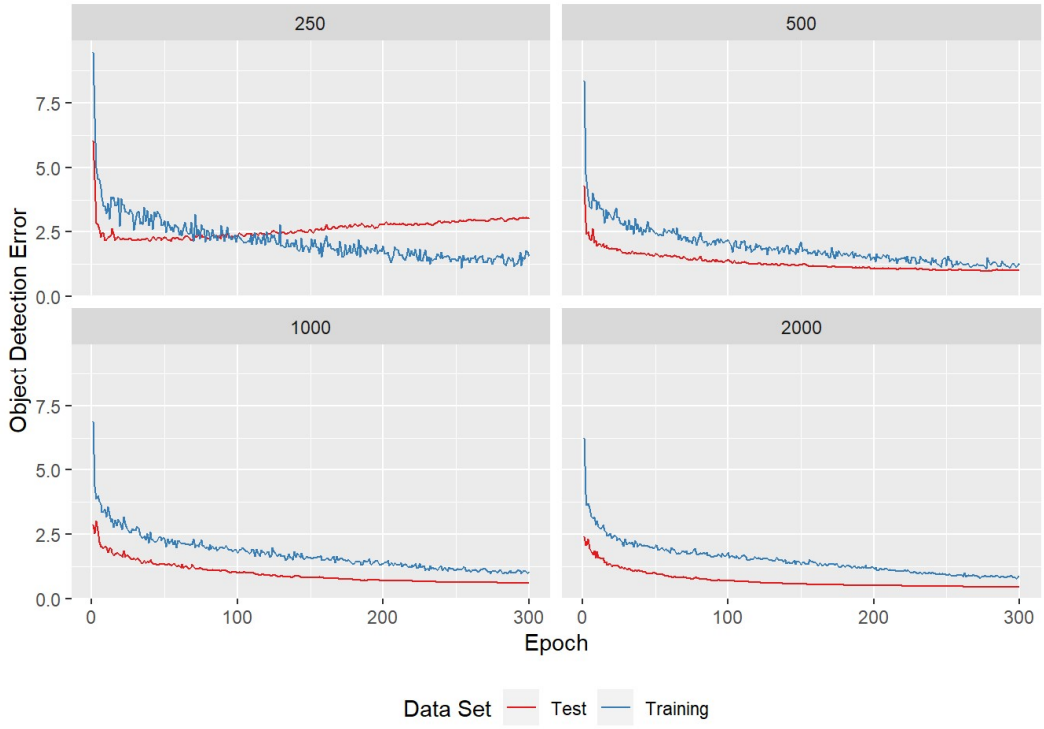


Figure 7. Classification Error of YOLO v3 Model Variants

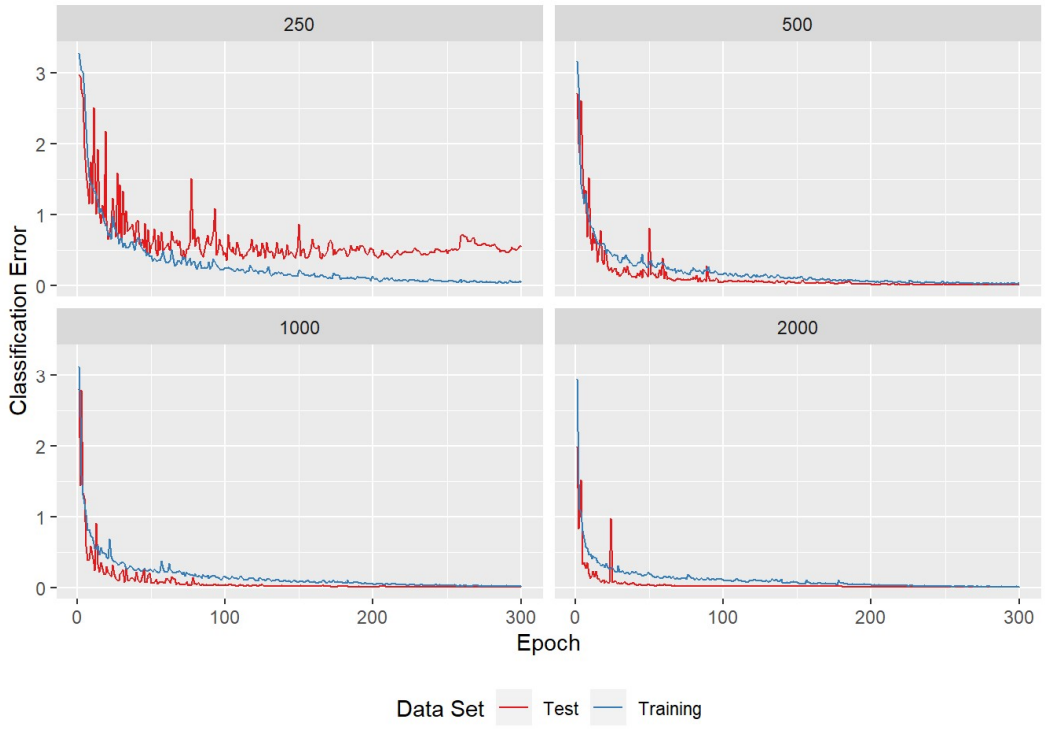


Figure 8. Object Detection Error of YOLO v5 Model Variants

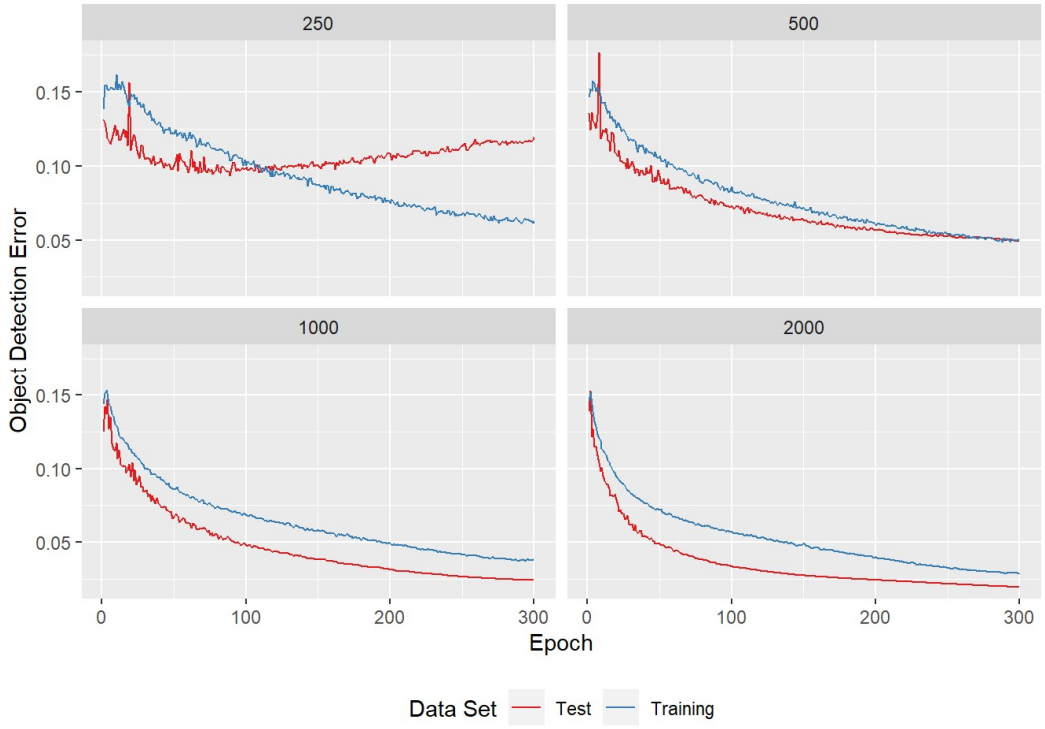
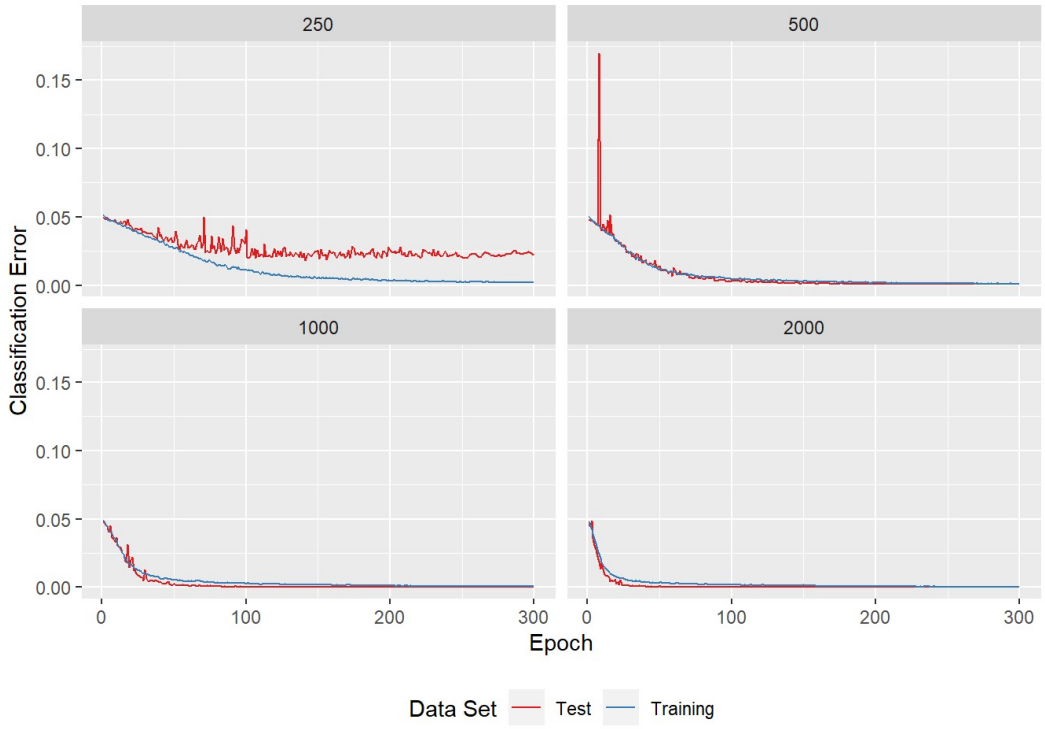


Figure 9. Classification Error of YOLO v5 Variants



Simulated Field Test

As explained in the methodology section of this paper, I performed a simulated field to gauge how well the object detectors generalize. Recall that this test involved preforming object detection on one hundred images outside of both the training and test set. These unseen images serve as a simulation of what a user of the app might experience. The simulated field test produced unexpected results.

The models did not perform as well on the field test images as they did on the test set. The mean average precision on these images ranged from 68.6% to 74.7%, compared to 73.1% to 99.5% on the test set. This indicates the models do not generalize as well as the training metrics suggest. It is also interesting to note that the evaluation metrics increase as the models are trained on more data, but then decreases for the variants trained on the most images. This unexpected result suggests the models overfit the data. Based on the metrics summarized in Table 4, the YOLO version five one-thousand image model preformed the best.

The mean average precision of the various plant species generally clustered around seventy percent. The models identify poison ivy better than other look-alikes for the variants trained on two and five hundred images. This performance advantage disappears for the models trained on one thousand plus images. The mean average percentage for these models is on part with

the overall mean average percentage. The models found box elder the most difficult plant to identify. Table 5 summarizes the mean average precision by plant species.

Table 4. Simulated Field Test Evaluation Metrics by Model

Method	Images	Precision	Recall	F_1	mAP@0.5
YOLO v3	250	69.5%	74.6%	70.2%	69.6%
YOLO v3	500	72.3%	74.2%	71.7%	69.4%
YOLO v3	1,000	74.5%	72.5%	72.6%	70.8%
YOLO v3	2,000	73.8%	70.8%	71.3%	68.6%
YOLO v5	250	52.9%	76.3%	62.5%	70.3%
YOLO v5	500	57.7%	77.8%	66.3%	71.7%
YOLO v5	1,000	61.5%	76.2%	68.1%	74.7%
YOLO v5	2,000	66.7%	71.1%	68.8%	69.0%

Table 5. mAP@0.5 by Model and Plant Type on Field Test Dataset

Class	v3 250	v3 500	v3 1,000	v3 2,000	v5 250	v5 500	v5 1,000	v5 2,000
All	69.6%	69.4%	70.8%	68.6%	70.3%	71.7%	74.7%	69.0%
Poison Ivy	72.1%	75.3%	70.6%	68.7%	72.1%	71.5%	74.2%	68.5%
Virginia Creeper	70.3%	65.5%	70.0%	61.7%	71.4%	73.8%	76.7%	72.1%
Bramble	70.5%	71.7%	72.3%	70.9%	74.4%	75.5%	73.5%	71.1%
Box Elder	65.5%	65.3%	70.5%	73.2%	63.1%	65.8%	74.5%	64.5%

Performance on Out of Sample Plants

Figure 10. Example of Misclassification

None of the models did a good job on the out of sample plants. They were almost always misclassified into one of the four in the training set. The object detector correctly saw no “plants” (meaning no poison ivy, brambles, box elder, or Virginia creeper) zero to three times out of twenty depending on the model.

Sometimes multiple misclassifications occurred as exemplified in figure 10 to the left. Table 6 below summarizes the plant species the models observed in the image. The misclassifications add up to more than the 20 images due to the misclassifications like those in figure 10.

Table 6. Misclassifications in the Out of Sample Plants

Class	v3 250	v3 500	v3 1,000	v3 2,000	v5 250	v5 500	v5 1,000	v5 2,000
Poison Ivy	10	8	12	12	12	12	13	12
Virginia Creeper	0	1	2	1	2	2	3	3
Bramble	5	4	4	6	3	2	3	3
Box Elder	7	9	8	6	12	8	10	8

Did the Model Miss Poison Ivy?

Table 7. Poison Ivy Detected Correctly

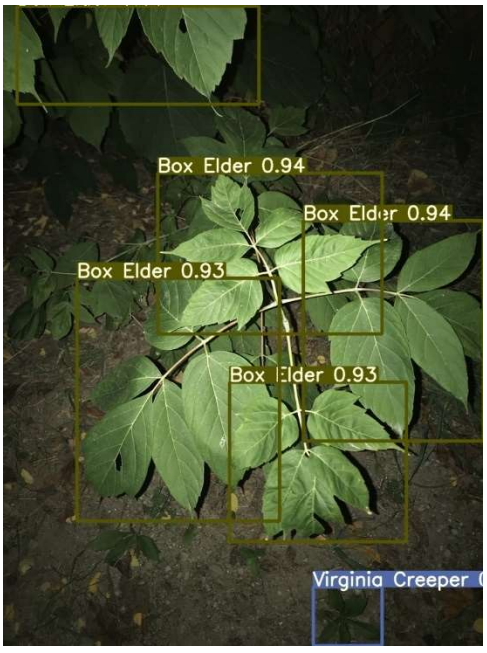
Images	v3	v5
250	19	20
500	19	20
1,000	20	20
2,000	20	19

The model did perform well on detecting poison ivy. Out of the twenty out of sample images the models found poison ivy in nineteen or all twenty of the images. While this performance seems encouraging, the model did see poison ivy when it was not present. Table 6 above shows that out of sample look-alike plants were most often classified as poison ivy.

The fact that the model seems to classify benign plants as poison ivy is not a bad thing. While it is not correct, the effect that this misclassification would have on the end user could be desirable. They would exercise caution, even when it is not needed.

Surprising Result from Simulated Field Test

Figure 11. Surprising Detection Result



There were some interesting results that came from the field test. One interesting result is seen in Figure 11 to the left. This is a picture of a box elder sapling. I selected this image for the field test due to the poor lighting conditions. I wanted to see how well the models would preform in this less than ideal condition.

The models preformed well with this image, detecting the box elder in the image with a high degree of confidence.

The interesting result was that it also detected Virginia creeper in the right-hand corner of the image. I, personally did not notice that was in the shadows when I labeled the image. The model detected a plant that I missed.

Learnings from the Simulated Field Test

The simulated field test provided a reasonably realistic view of how well the models would preform in production. The performance metrics differed significantly from those collected at the end of the training process. They suggest the data augmentation techniques I used did not add as much additional information as I had hoped. Even though the metrics are lower I am not disappointed with the object detector's performance. The YOLO version five has a mean average precision of less than 50% on the COCO dataset. The mean average precision of any of the plant species exceeds this level. Based on this the models created for this study are successes.

Cost to Train the Model

As discussed in the methodology, in addition to the tests focusing on the model accuracy metrics, some time would be devoted to the monetary costs of training the model. The models were only trained once and so it is unknown if the hours taken to train the model are representative. But even in their rough form they provide some information for consideration.

The object detectors trained on one of two GPU models. Half trained on NVIDA V100-SXM2 and half on NVIDA P100-PCIE. The current price for an hour of GPU time on Google's Cloud Engine is $0.74 for the V100, and $0.43 for the P100.[5] Table 8 summarizes the time it took to train the models. Generally, the more images, the more time is needed to train the model. Applying these times to the current prices give a rough estimate of what it would cost to train a model. The models cost anywhere from half a dollar to over fourteen depending on the hardware. The version five models took longer to train than the version three models. I cannot explain why the version five 500 image model took more time than the version five 1,000 image model. I don't know if the 12+ hours for the 500 image is long, or if the 8+ hours for the 1,000 images is short. These results are illustrative of my experience using the shared cloud computing resources, and others might have different results.

Table 8. Costs for Training YOLO Model

Method	Images	Hours to Train	GPU	P100 Cost	V100 Cost
YOLO v3	250	1.30	V100	$0.56	$0.96
YOLO v3	500	2.25	V100	$0.97	$1.66
YOLO v3	1,000	9.83	P100	$4.23	$7.28
YOLO v3	2,000	19.43	P100	$8.36	$14.38
YOLO v5	250	6.37	P100	$2.74	$4.71
YOLO v5	500	12.19	P100	$5.24	$9.02
YOLO v5	1,000	8.16	V100	$3.51	$6.04
YOLO v5	2,000	16.16	V100	$6.95	$11.96

[5] Source: https://cloud.google.com/compute/gpus-pricing

Conclusion and Future Work

This research project set out to develop a machine learning model that detects poison ivy, that could be embeded in a mobile app, and preliminary findings indicate success. Two-hundred and fifty images of poison ivy and three look-alikes plant species (box elder, brambles and Virginia creeper) from the iNaturalist project served as the base dataset for the model. I created additional training data using data augmentation techniques. This resulted in datasets of five hundred, one thousand, and two thousand images. YOLO version three and five object detection algorithms trained on these data.

Evaluation of these model using standard object detection evaluation metrics indicates strong performance by the models. Mean average precision on the test set ranged from seventy-three percent to virtually one hundred percent. While these results are encouraging, upon closer examination, evidence that the models overfit the data began to rise.

In addition to the standard evaluation metrics, I simulated a “field test.” I accomplished this by producing one hundred additional images and used them to evaluation the performance of the object detection models “in the wild”. Twenty images from each of the four plant species, and twenty images of other poison ivy look-alikes never “seen” by the model made up the field test data set.

Results from this simulated field test indicate that the models did not generalize as well as hoped. The models identified the plants in the images fairly well. Butut not as well as they did during training. Mean average precision ranged from roughly sixty-nine to seventy-five percent, quite lower from the seventy-three to one hundred percent range in training. The field test indicates the object detectors misclassify other look-alikes as poison ivy, in essence “seeing” it when it is not present. This, however, is not necessarily a bad outcome, as this would result in falsely classifying a benign plant as poison ivy. The field test indicated the models did a good job identifying poison ivy, but improvement is possible.

Future Work

Additional work should be done to improve model performance. The data from the training process indicate that the model preforms significantly better with more data. If the base dataset is composed of 1,000 unique images of each plant species, the model performance would certainly improve.

The YOLO version three models benefited from transfer learning. As noted earlier in this paper, the developers of YOLO version five has recent added this enhancement. Additional work should take advantage of this improvement. Evidence from this project indicates that the models would train faster and better than starting from scratch.

Future work might consider shortening the number of epochs the model trains. This project went with the default of three hundred epochs, but it appears that little was gained beyond the one hundred fiftieth epoch. Close observation of this and the difference between training and test set errors should prevent future researchers from creating models that overfit the data.

Other data augmentation strategies could improve performance. For example, the images could be rotated ninety degrees. This would make the object detector more robust to end users rotating their smartphones when using the app.

Beyond improving the performance of the object detector, future work could be done to create the app to use the model. This research project intentionally stopped short of this even though it is the end of this word. Developing an app would allow for an actual field test and ultimately the app that would help other successfully identify poison ivy in the wild.

Works Cited

Aakif, Aimen, and Muhammad Faisal Khan. 2015. "Automatic classification of plants based on their leaves." *Biosystems Engineering* p.66-75.

Al-masni, Mohammed A, Mugahed A Al-antari, Jeong-Min Park, Geon Gi, Tae-Yeon Kim, Patricio Rivera, Edwin Valarezo, Mun-Taek Choi, Seung-Moo Han, and Tae-Seong Kim. 2018. "Simultaneous detection and classification of breast masses in digital mammograms via a deep learning YOLO-based CAD system." *Computer methods and programs in biomedicine* Vol.157: p.85-94.

Bochkovskiy, Alexy, Chien-Yao Wang, and Hong-yuan Liao. 2020. "YOLOv4: Optimal Speed and Accuracy of Object Detection." *arXiv* 2004.10934.

Cui, Yin, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. 2018. "Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning." *Computer Vision and Pattern Recognition.* doi:10.1109/CVPR.2018.00432.

Das, Abhiram, Alexander Bucksch, Charles A Price, and Joshua S Weitz. 2014. "ClearedLeavesDB: an online database of cleared plant leaf images." *Plant Methods* Vol 10 (8). doi:10.1186/1746-4811-10-8.

Dollar, Piotr, Ron Appel, Serge Belongi, and Pietro Perona. 2014. "Fast feature pyramids for object." *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 36 (8): p. 1532-1545.

Du, Juan. 2018. "Understanding of Object Detection Based on CNN Family and YOLO." *Journal of physics. Conference series* Vol.1004: p.012029. doi::10.1088/1742-6596/1004/1/012029.

Felzenszwalb, P F, R B Girshick, D McAllester, and D Ramanan. 2010. "Object detection with discriminatively trained part-based models." *IEEE transactions on pattern analysis and machine intelligence* Vol. 32 (9): p. 1627-1645.

Fukushima, Kunihiko. 2005. "Restoring partly occluded patterns: a neural network model." *Neural Networks* Vol 18 (Issue 1): p. 33-43. doi:https://doi.org/10.1016/j.neunet.2004.05.001.

Gaurav, Agarwalk, Peter Belhumeur, Steven Feiner, David Jacobs, W. John Kress, Ravi Ramamoorthi, Norman A Bourg, et al. 2006-08. "First steps toward an electronic field guide for plants." *Taxon* Vol. 55 (3): p. 597-610.

Girshick, Ross. 2015. "Fast R-CNN." *IEEE International Conference on Computer Vision (ICCV).* p. 1440-1448.

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. "Rich feature hierarchies for accurate object detection and semantic segmentation." *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition.* p. 580-587.

Hendry, and Rung-Ching Chen. 2019. "Automatic License Plate Recognition via sliding-window darknet-YOLO deep learning." *Image and vision computing* Vol.87: p.47-56.

Hochmair, Hartwig H., Roudolf H. Scheffrahn, Mathieu Basille, and Matthew Boone. 2020. "Evaluating the data quality of iNaturalist termite records." *PLoS ONE* (Public Library of Science) Vol. 15 (Issue 5): p.e0226534.

Jin, Taisong, Xueliang Hou, Pifan Li, and Feifei Zhou. 2015. "A Novel Method of Automatic Plant Species Identification Using Sparse Representation of Leaf Tooth Features." *PLoS ONE* Vol. 10 (10): e0139482. doi:10.1371/journal.pone.0139482.

Johnson, Justin M., and Taghi M. Khoshgoftaar. 2019. "Survey on deep learning with class imbalance." *Journal of Big Data* vol. 6. doi:https://doi.org/10.1186/s40537-019-0192-5.

Keddy, Paul. 2007. *Plants and Vegetation: Origins, Processes, Consequences.* Cambridge UK: Cambridge University Press.

Kumar, Meeraj, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida C. Lopez, and João V. B. Soares. 2012. "Leafsnap: A Computer Vision System for Automatic Plant Species Identification." Edited by Andrew Ftizgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato and Cordelia Schmid. *Computer Vision - European Conference on Computer Vision 2012.* Berlin, Heidelberg: Springer Berlin Heidelberg. p.502-516. doi:10.1007/978-3-642-33709-3_36.

Liu, Guoxu, Joseph Christian Nouaze, Phillippe Lyonel Touko Mbouembe, and Jae Ho Kim. 2020. "YOLO-Tomato: A Robust Algorithm for Tomato Detection Based on YOLOv3." *Sensors* Vol. 20 (Issue 7): p. 2145. doi:10.3390/s20072145.

Liu, Jun, and Xuewei Wang. 2020. "Tomato Diseases and Pests Detection Based on Improved Yolo V3 Convolutional Neural Network." *Frontiers in plant science* Vol.11. doi:10.3389/fpls.2020.00898.

Noh, Junhyug, Soochan Lee, Beomsu Kim, and Gunhee Kim. 2018. "Improving Occlusion and Hard Negative Handling." *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* Salt LAke City, Utay: IEEE. p. 966-974. doi:1109/CVPR.2018.00107.

Nugent, Jill. 2018. "iNaturalist: Citizen science for 21st-century naturalists." *Science Scope* Vol. 41 (Issue 7): p.12-15.

Papageorgiou, Constantine, and Tomaso Poggio. 2000. "A Trainable System for Object Detection." *International Journal of Computer Vision* Vol. 38 (1): p. 15-33.

Redmon, Joseph, and Ali Farhadi. 2017. "YOLO9000: Beter, Faster, Stronger." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE. pp. 7263-7271.

Redmon, Joseph, and Ali Farhadi. 2018. "YOLOv3: An Incremental Improvement." *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol.15: p.1125–1131.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You Only Look Once: Unified, Real-Time Object Detection." *Conference on Computer Vision and Pattern Recognition (CVPR).* IEE. p.779-788.

Ren, Shaoquing, Kaiming He, Ross Girshick, and Jian Sun. 2015. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *Advances in neural information processing system* Vol. 28: p. 91-99.

Rossi, Ryann E. 2017. "Using iNaturalist observations to detect disease in Red Mangroves (Rhizophora mangle)." *PeerJ preprints.*

Rzanny, Michael, Marco Seeland, Jana Wäldchen, and Patrick Mäder. 2017. "Acquiring and preprocessing leaf images for automated plant identification: understanding the tradeoff between effort and information gain." *Plant Methods* Vol. 13 (97). doi:10.1186/s13007-017-0245-8.

Seltzer, Carrie. 2019. "Making Biodiversity Data Social, Shareable, and Scalable: Reflections on iNaturalist & citizen science." *Biodiversity Information Science and Standards* Vol. 3: p.e46670. doi:10.3897/biss.3.46670.

Shorten, Connor, and Taghi M. Khoshgoftaar. 2019. "A survey on Image Data Augmentation for Deep Learning." *Journal of Big Data* Vol. 6. doi:10.1186/s40537-019-0197-0.

Šulc, Milan, and Jiří Matas. 2017. "Fine-grained recognition of plants from images." *Plant Methods* Vol. 13 (115). doi:10.1186/s13007-017-0265-4.

Tian, Yunong, Guodong Yang, Zhe Wang, Hao Wang, En Li, and Zize Liang. 2019. "Apple detection during different growth stages in orchards using the improved YOLO-V3 model." *Computers and electronics in agriculture* Vol.157: p.417-426.

Wäldchen, Jana, and Patrick Mäder. 2017. "Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review." *Computational Methods in Engineering* p.1-37.

Wäldchen, Jana, Michael Rzanny, Marco Seeland, and Patrick Mäder. 2018. "Automated plant species identification—Trends and future directions." *PLoS Comput Biol* Vol. 14 (4): e1005993. doi:10.1371/journal.pcbi.1005993.

Wu, Stephen Gang, Forrest Sheng Bao, Eric You Xu, Yu-Xuan Wang, Yi-Fan Chang, and Qiao-Liang Xiang. 2007. "A Leaf Recognition Algorithm for Plant Classification." *IEEE International Symposium on Signal Processing and Information Technology.* p.11-16.

Appendix

Additional Key Metric Visualizations

At the end of training the YOLO models, the following figures are produced showing the change in the key performance metrics over the training process.

Figure 12. Key Metrics for the YOLO v3 250 Image Model

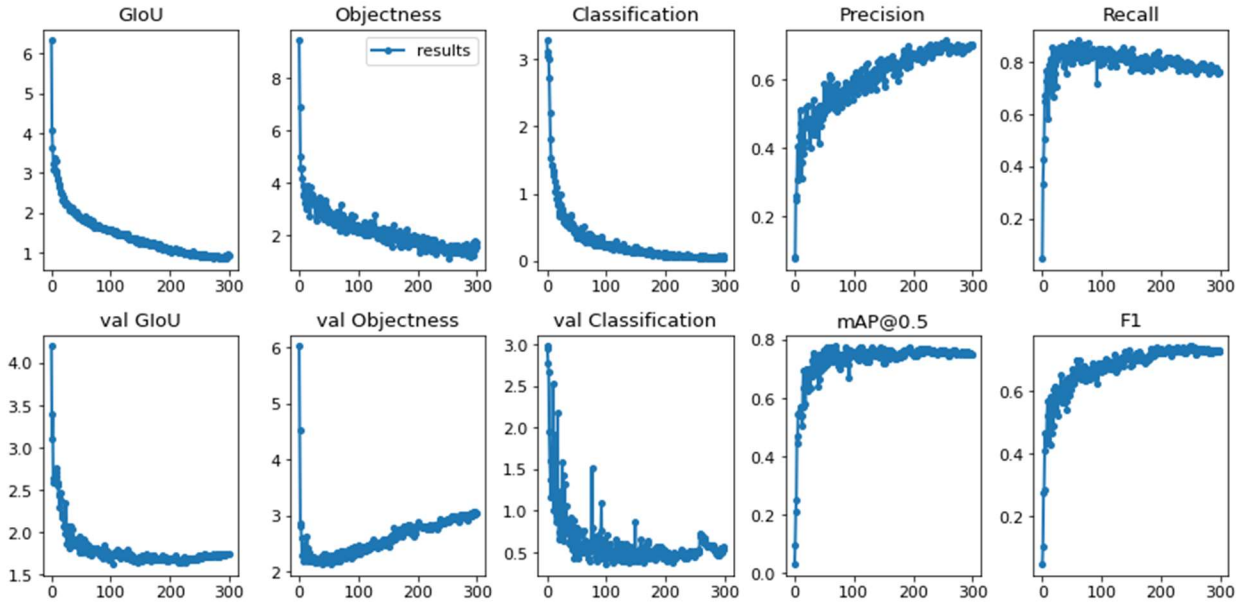


Figure 13. Key Metrics for the YOLO v3 500 Image Model

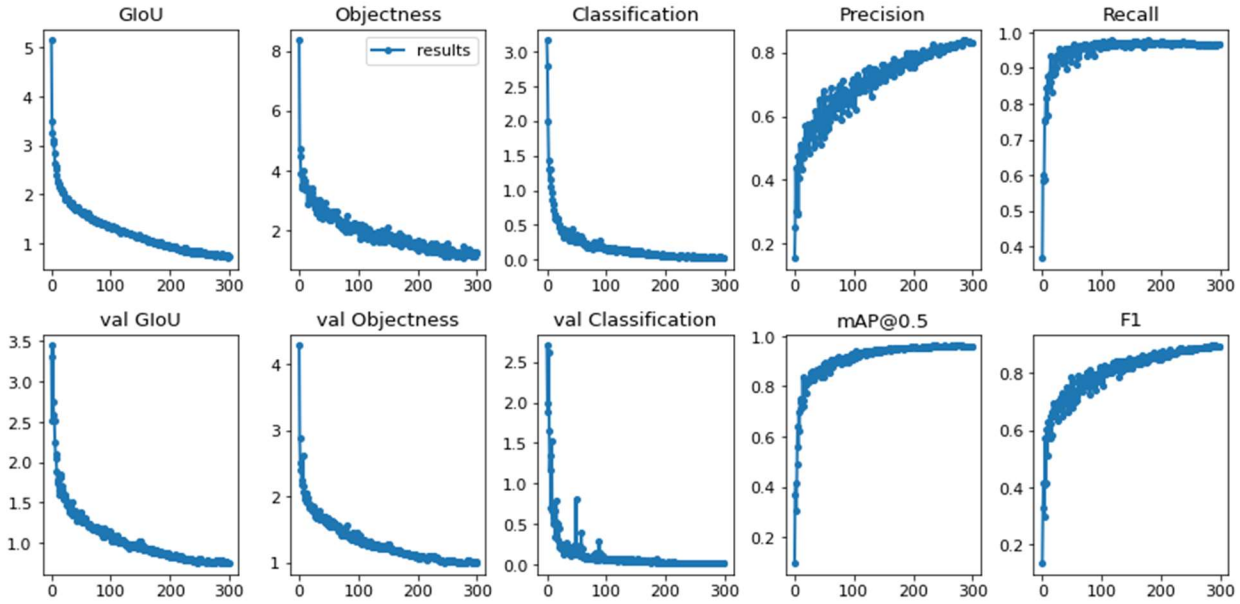


Figure 14. Key Metrics for the YOLO v3 1,000 Image Model

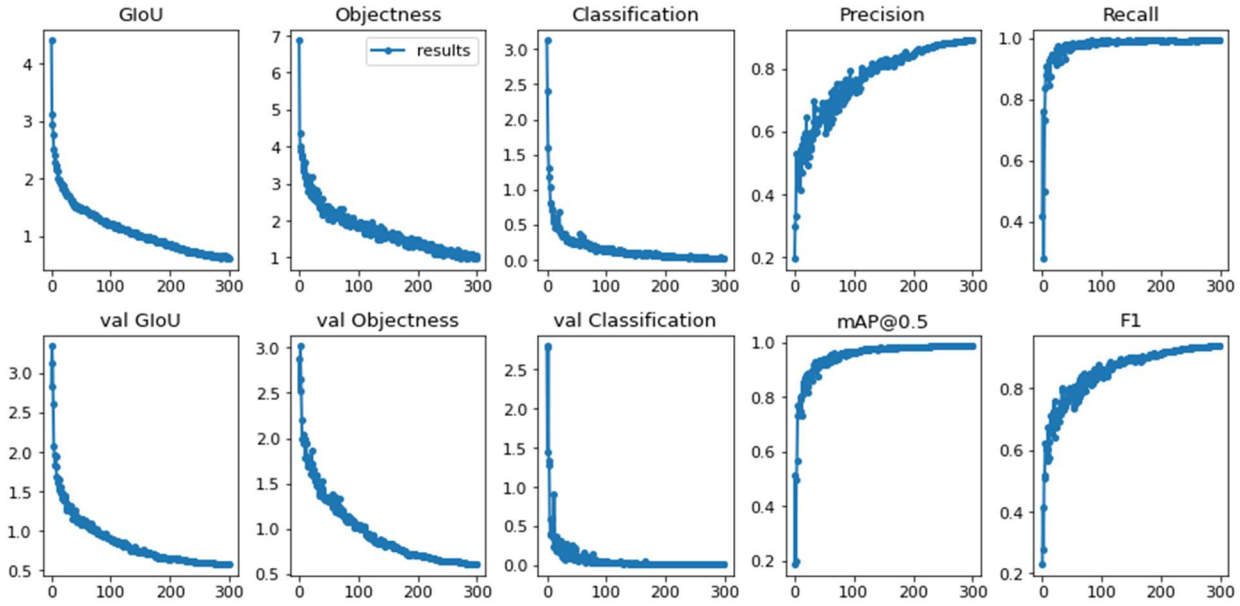


Figure 15. Key Metrics for the YOLO v3 2,000 Image Model

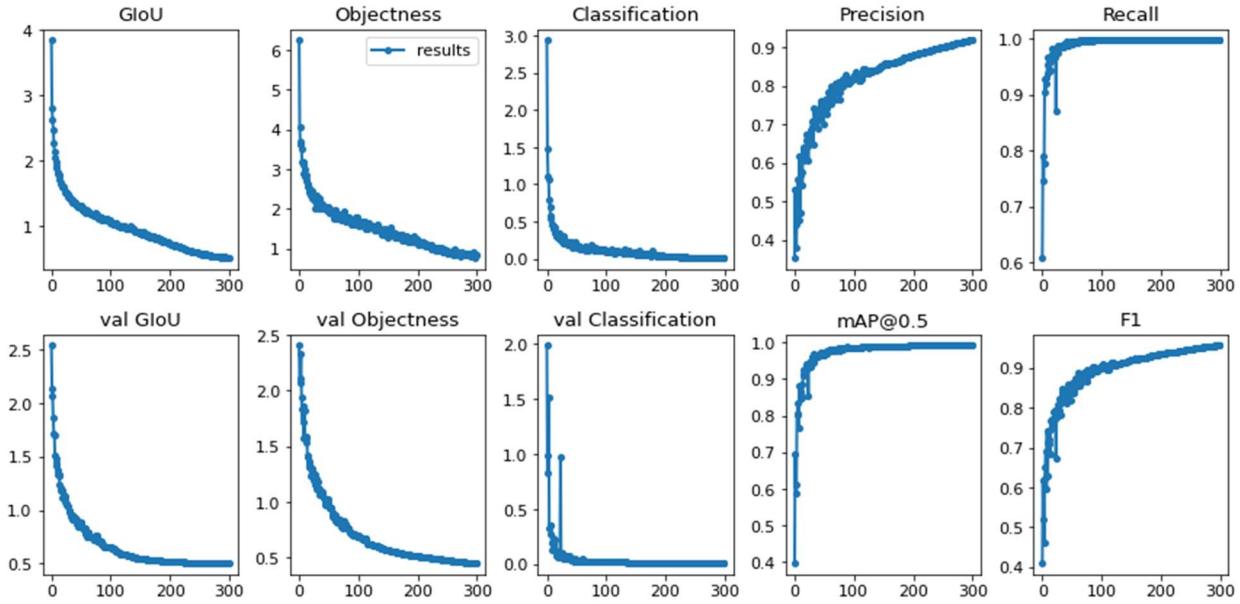


Figure 16. Key Metrics for the YOLO v5 250 Image Model

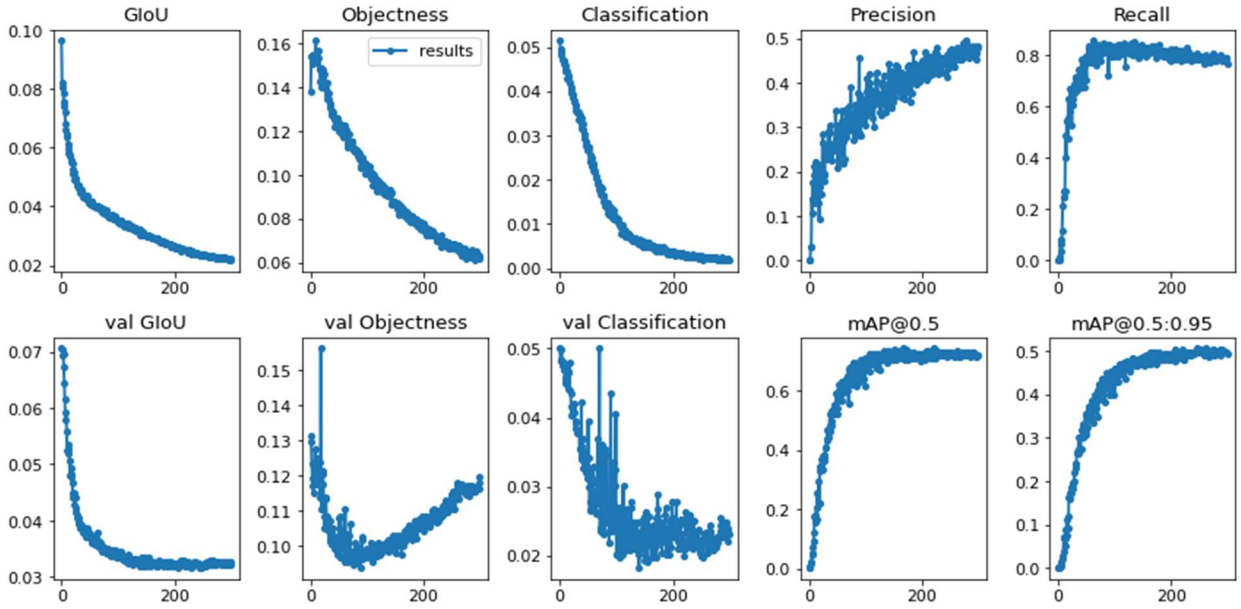


Figure 17. Key Metrics for the YOLO v5 500 Image Model

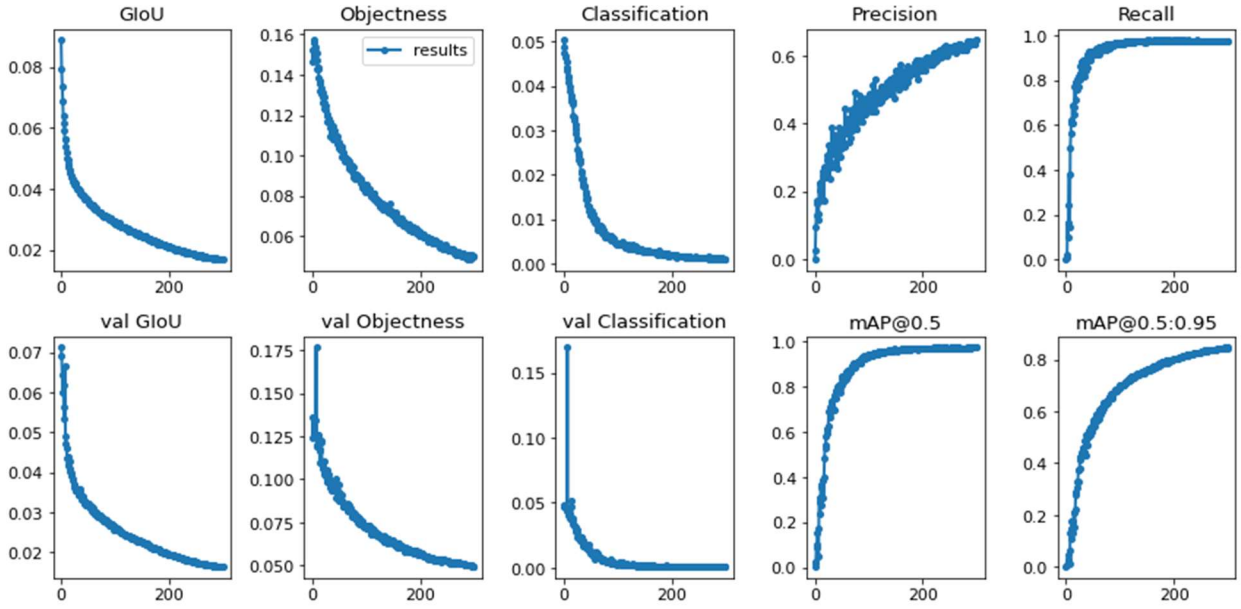


Figure 18. Key Metrics for the YOLO v5 1,000 Image Model

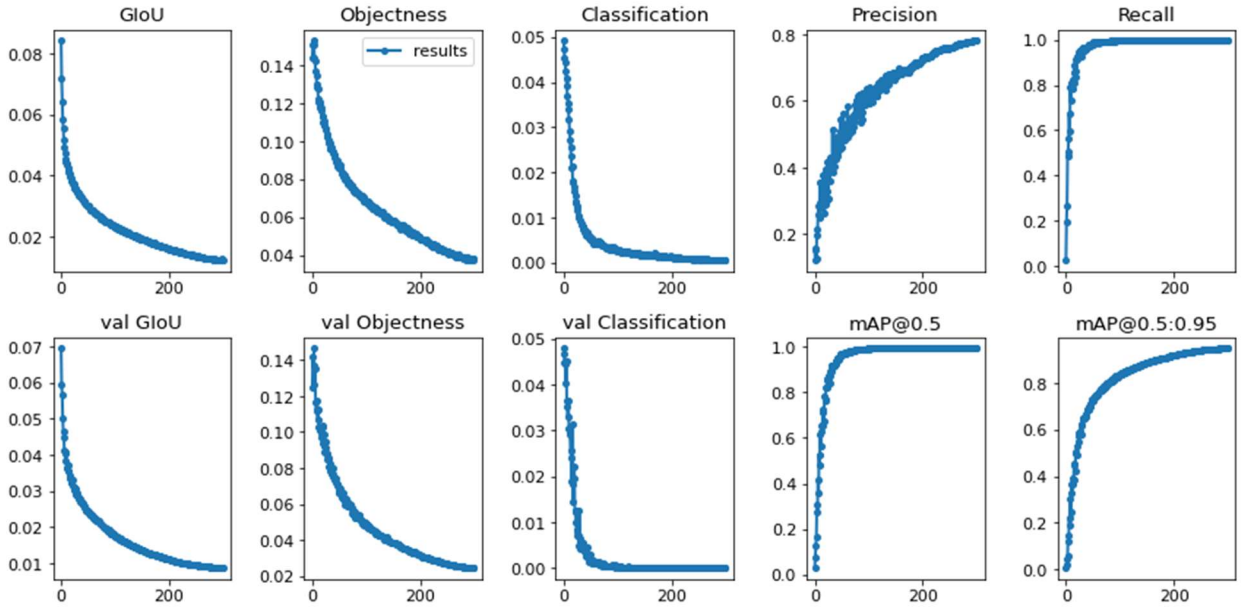
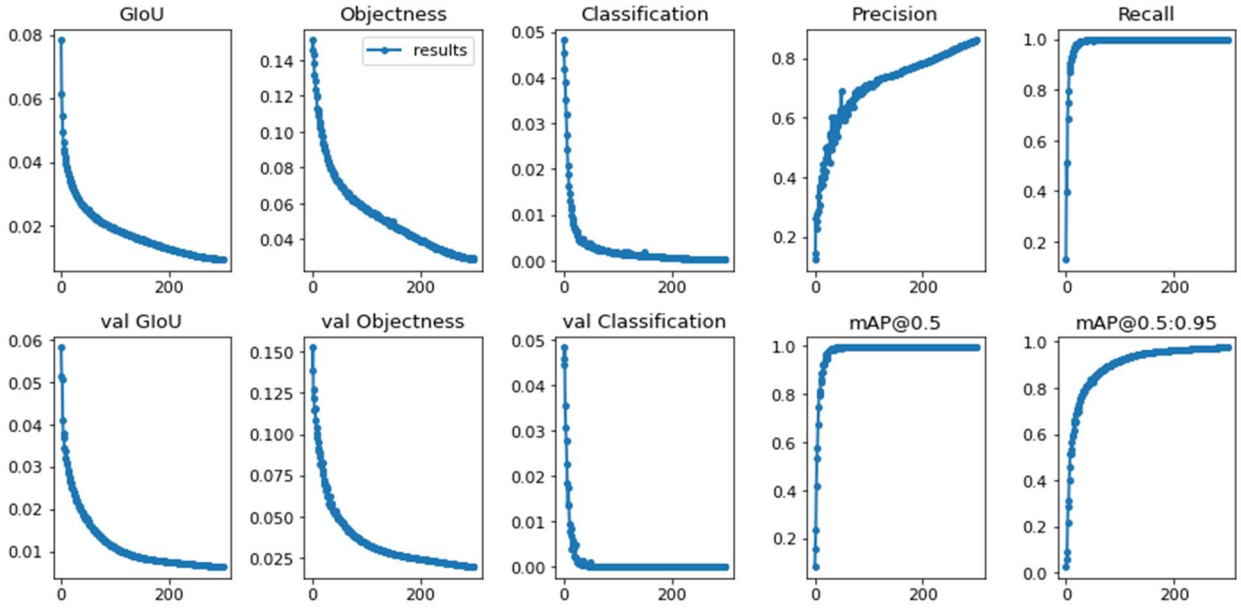


Figure 19. Key Metrics for the YOLO v5 2,000 Image Model



Additional Details on the Simulated Field Test

Table 9. Correct Identifications by Object Detector

Method	YOLO V3				YOLO V5			
Images	**250**	**500**	**1,000**	**2,000**	**250**	**500**	**1,000**	**2,000**
Poison Ivy	19	19	20	20	20	20	20	19
Box Elder	19	19	19	20	20	20	20	20
Brambles	19	20	19	20	20	19	19	19
Virginia Creeper	18	20	20	20	20	19	20	20
Null	1	3	0	2	1	1	0	1
Total	76	81	78	82	81	79	79	79

Source Code

All source code for this project is available at https://github.com/mikeasilva/CUNY-SPS/tree/master/DATA698. Due to GitHub's limit on file sizes, unfortunately the data set is not hosted there but is available through Google Drive[6]. The files that were primarily used in this project are included here. There are additional files that were used to explore the data, however they are not included in this appendix. The source code provided here is what is needed to replicate the research explained in the body of the report.

Data Acquisition, Wrangling and Visualization Scripts

01 - Gather iNaturalist JSON Data.py

The following script was used to build out the database of images that were under consideration for including in the project. It makes requests to the iNaturalist API, and saves the results to JSON files. These files are latter processed.

```
# -*- coding: utf-8 -*-
"""
Gather iNaturalist JSON Data

Created on Tue Sep 15 20:23:39 2020

@author: Michael Silva

This script gathers data on iNaturalist observations.  The data are organized
by iNaturalist taxons.  It will request the maximum allowed number of data and
save them as JSON files for further processing.
"""

from os import path
import json
import requests

# iNaturalist Taxon ID and the Label (does not have to be unique)
```

[6] https://drive.google.com/file/d/1_FcUcrjHjuAj5iMr8R4gLE70bUZKi1Hg/view?usp=sharing

```
taxon_id_and_label = {
    "58732": "Poison Ivy",
    "58729": "Poison Ivy",
    "50278": "Virginia Creeper",
    "47726": "Box Elder",
    "138680": "Thicket Bean",
    "62671": "Kudzu",
    "62666": "Kudzu",
    "82529": "Grapevine",
    "119936": "Grapevine",
    "50625": "Hoptree",
    "58738": "Sumac",
    "55911": "Bramble",
    "54436": "Bramble",
    "47544": "Bramble",
    "125489": "Bramble",
    "50298": "Strawberry",
    "243824": "Strawberry",
    "77835": "Japanese Honeysuckle",
    "50310": "Jack in the Pulpit",
    "166162": "Boston Ivy",
    "129021": "Honewort",
    "51435": "Trillium",
    "50855": "Trillium",
    "51741": "Goutweed",
    "85485": "Hog Peanut",
}
# Number of observations per API call
per_page = 100
# iNaturalist currently only allows 10,000 results
inaturalist_max = int(round(10000 / per_page, 0))


for taxon_id, label in taxon_id_and_label.items():
    print("Gathering " + taxon_id_and_label[taxon_id] + " Data...")
    scrape_and_save = True
    page = 1

    while scrape_and_save:
        # Determine if this has already been scrapped
        page_str = str(page)
        page_str = page_str.zfill(3)
        file_path = (
            "./iNaturalist/json/"
            + taxon_id_and_label[taxon_id]
            + "-"
            + taxon_id
            + "-"
            + page_str
            + ".json"
        )
        if not path.isfile(file_path):
            # File does not exist so scrape and save
            url = (
                "https://api.inaturalist.org/v1/observations?verifiable=true&order_by=observations.id&order=desc&page="
                + str(page)
                + "&spam=false&taxon_id="
                + taxon_id
                + "&locale=en-US&preferred_place_id=1&per_page="
                + str(per_page)
```

```
                + "&return_bounds=true&quality_grade=research"
            )
            response = requests.get(url)
            response_json = response.json()

            # Save the JSON data
            with open(file_path, "w") as f:
                json.dump(response_json, f)
        # Determine how much there is to scrape
        if "pages_to_scrape" not in locals():
            with open(file_path) as f:
                response_json = json.load(f)
            pages_to_scrape = int(round(response_json["total_results"] / per_page, 0))

            if pages_to_scrape > inaturalist_max:
                pages_to_scrape = inaturalist_max

        print("  " + str(page) + " of " + str(pages_to_scrape))
        # Check if we have more to scrape
        if page == pages_to_scrape or page > pages_to_scrape:
            scrape_and_save = False
            del pages_to_scrape
               else:
                   page += 1
```

02 - Process iNaturalist JSON Data.py

This script processes creates an SQLite database that holds some of the information in the JSON files. This database forms the index of images.

```
# -*- coding: utf-8 -*-
"""
Process iNaturalist JSON Data

Created on Fri Sep 18 16:24:40 2020

@author: Michael Silva

Having previously gathered iNaturalist observation records, and saved them to
JSON files, this script processess the data and builds out an index of API calls.
All data will be stored in an SQLite database.
"""

import json
from glob import glob
import sqlite3

create_tables = False

conn = sqlite3.connect("images.db")
c = conn.cursor()

if create_tables:
    c.execute("DROP TABLE IF EXISTS image;")
    c.execute("DROP TABLE IF EXISTS json_file;")
    c.execute(
        "CREATE TABLE images (id INTEGER PRIMARY KEY AUTOINCREMENT, label TEXT, url tex
t, img_url TEXT DEFAULT NULL, lat REAL DEFAULT NULL, lon REAL DEFAULT NULL, observed_on
 TEXT DEFAULT NULL, scrapped INTEGER default 0, assigned INTEGER DEFAULT 0, downloaded
INTEGER DEFAULT 0, error INTEGER DEFAULT 0);"
```

```
    )
    c.execute("CREATE INDEX IF NOT EXISTS idx_assigned ON images (assigned);")
    c.execute("CREATE INDEX IF NOT EXISTS idx_downloaded ON images (downloaded);")
    c.execute("CREATE INDEX IF NOT EXISTS idx_scrapped ON images (scrapped);")
    c.execute("CREATE INDEX IF NOT EXISTS idx_error ON images (error);")
    c.execute("CREATE TABLE json_file (file_name TEXT);")
    conn.commit()

for file in glob("./iNaturalist/json/*.json"):
        c.execute("SELECT COUNT(*) FROM json_file WHERE file_name = ?", (file,))
        result = c.fetchone()
        if result[0] == 0:
            print(file)
            with open(file) as json_file:
                label = file.split("\\")[1].split("-")[0]
                data = json.load(json_file)
                if "results" in data:
                    for result in data["results"]:
                        if "inaturalist.org" in result["uri"]:
                            url = result["uri"].replace(
                                "https://www.inaturalist.org/",
                                "https://api.inaturalist.org/v1/",
                            ).replace(
                                "http://www.inaturalist.org/",
                                "https://api.inaturalist.org/v1/",
                            )
                            c.execute("SELECT COUNT(*) FROM images WHERE url = ?", (url
,))
                            r = c.fetchone()
                            if r[0] == 0:
                                c.execute("INSERT INTO images (label, url) VALUES (?,?)
;", (label, url))
            c.execute("INSERT INTO json_file VALUES (?)", (file,))
            conn.commit()

conn.close()
```

03 - iNaturalist Work Controller.py

In order to maximize the extraction of the data from iNaturalist a small Flask App was developed to serve as a “work controller.” It would send out tasks to the workers, like scrape data or download images and managed. It also served has a dashboard functionality that was used to monitor the scrapping workload.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 19 07:15:13 2020

@author: Michael Silva
"""

import json
import sqlite3
from collections import OrderedDict
from flask import Flask, jsonify, request, g, render_template
from flask.views import View

DATABASE = "images.db"
debug = False
```

```
16. app = Flask(__name__)
17.
18.
19. def make_dicts(cursor, row):
20.     return dict((cursor.description[idx][0], value) for idx, value in enumerate(row))
21.
22.
23. def get_db():
24.     db = getattr(g, "_database", None)
25.     if db is None:
26.         db = g._database = sqlite3.connect(DATABASE)
27.         db.row_factory = make_dicts
28.     return db
29.
30.
31. def db_save():
32.     get_db().commit()
33.
34.
35. def get_viz_data():
36.     labels = ""
37.     series = {}
38.     cur = get_db().cursor()
39.
40.     for data in cur.execute("SELECT * FROM record_counts;").fetchall():
41.         label = data["label"]
42.         for k, v in data.items():
43.             if k != "label":
44.                 k = k.title()
45.                 series_k = series.get(k, list())
46.                 series_k.append(v)
47.                 series[k] = series_k
48.         labels += "'" + label + "', "
49.     labels = labels[: len(labels) - 2]
50.     return (labels, series)
51.
52.
53. @app.teardown_appcontext
54. def close_connection(exception):
55.     db = getattr(g, "_database", None)
56.     if db is not None:
57.         db.close()
58.
59.
60. @app.route("/")
61. def index():
62.     data = {}
63.     data["categories"], data["series"] = get_viz_data()
64.     return render_template("index.html", data=data)
65.
66.
67. @app.route("/image_downloaded")
68. def image_downloaded():
69.     cur = get_db().cursor()
70.     image_id = request.args.get("id")
71.     cur.execute(
72.         "UPDATE images SET assigned = 1, downloaded = 1 WHERE id = ?", (image_id,)
73.     )
74.     db_save()
75.     return json.dumps({"success": True}), 200, {"ContentType": "application/json"}
76.
```

```
77.
78. @app.route("/image_scrapped")
79. def image_scrapped():
80.     cur = get_db().cursor()
81.     image_id = request.args.get("id")
82.     cur.execute("UPDATE images SET scrapped = 1 WHERE id = ?", (image_id,))
83.     db_save()
84.     return json.dumps({"success": True}), 200, {"ContentType": "application/json"}
85.
86.
87. @app.route("/image_data", methods=["POST"])
88. def data_intake():
89.     cur = get_db().cursor()
90.     image_id = request.args.get("id")
91.     data = OrderedDict(request.get_json())
92.     val = tuple(data.values())
93.     sql = "UPDATE images SET "
94.     for k in data.keys():
95.         sql = sql + k + " = ?, "
96.     sql = sql[: len(sql) - 2] + " WHERE id = " + image_id + ";"
97.
98.     try:
99.         cur.execute(sql, val)
100.                db_save()
101.                return json.dumps({"success": True}), 200, {"ContentType": "application/
    json"}
102.            except:
103.                return json.dumps({"failure": True}), 200, {"ContentType": "application/
    json"}
104.
105.
106.        @app.route("/download_job")
107.        def job():
108.            cur = get_db().cursor()
109.            cur.execute(
110.                "SELECT images.* FROM images, (SELECT MIN(id) AS id FROM images WHERE as
    signed = 0 and img_url IS NOT NULL and error = 0) AS f WHERE images.id = f.id;"
111.            )
112.            try:
113.                task = cur.fetchall()[0]
114.                cur.execute("UPDATE images SET assigned = 1 WHERE id = ?", (task["id"],)
    )
115.                db_save()
116.            except:
117.                # We have assigned all the images
118.                task = {"id": "Done"}
119.            return jsonify(task)
120.
121.
122.        @app.route("/reset")
123.        def reset():
124.            cur = get_db().cursor()
125.            cur.execute("UPDATE images SET assigned = 0 WHERE downloaded = 0")
126.            cur.execute("UPDATE images SET scrapped = 0 WHERE scrapped = -1")
127.            db_save()
128.            return "Success"
129.
130.
131.        @app.route("/scrape_job")
132.        def scrape_job():
133.            cur = get_db().cursor()
```

```
            starts_with = request.args.get("starts_with")
            if starts_with is None:
                cur.execute(
                    "SELECT images.* FROM images, (SELECT MIN(id) AS id FROM images WHER
    E scrapped = 0 AND error = 0) AS f WHERE images.id = f.id;"
                )
            else:
                cur.execute(
                    "SELECT images.* FROM images, (SELECT MIN(id) AS id FROM images WHER
    E scrapped = 0 AND error = 0 AND label LIKE '"
                    + starts_with
                    + "%') AS f WHERE images.id = f.id;"
                )

            try:
                task = cur.fetchall()[0]
                cur.execute("UPDATE images SET scrapped = -
    1 WHERE id = ?", (task["id"],))
                db_save()
            except:
                # We have assigned all the images
                task = {"id": "Done"}
            return jsonify(task)


        if __name__ == "__main__":
            app.run(debug=debug)
```

04 - iNaturalist Worker.py

The following is the worker that executes order given by the work controller, and reports back. At the time that data was extracted from iNaturalist, I would have two to four of these workers running simultaneously. This was necessary to secure the data in a timely manner so I could have time to label the data.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 19 07:15:13 2020

@author: Michael Silva
"""

import requests
from os import path, mkdir
import json
import time

# Initial values
base_url = "http://127.0.0.1:5000/"
has_work = True
cooling_down = False
job_count = 0
starts_with = None

# How will this worker operate?  Which mode?
work_modes = {"1": "scrape_job", "2": "download_job", "3": "scrape_job", "4":"reset"}
work_mode = input("Enter work mode (1=scrape and download image, 2=download image ONLY,
     3=scrape ONLY, 4=reset): ")
endpoint = work_modes[work_mode]
```

```
# Ask about label starting with
if work_mode in ["1", "3"]:
    starts_with = input("What should the label start with (Press enter for nothing)? ")

    if starts_with == "":
        starts_with = None

# Helper functions
def relay_image_downloaded(id):
    url = base_url + "image_downloaded?id=" + id
    response = requests.get(url)
    response = response.json()
    if "success" not in response:
        print("Something bad happened > " + url)

def get_img_path(label):
    img_path = "./iNaturalist/images/" + label + "/"
    # Create directories if needed
    if not path.exists(img_path):
        mkdir(img_path)
    return(img_path)

def download_image(img_url, img_path, img_id_str):
    # Check if image path exists
    file_extension = img_url.split(".")[-1].split("?")[0]
    file_path = img_path + img_id_str.zfill(8) + "." + file_extension
    if not path.exists(file_path):
        print("Downloading " + label + " >> " + img_url)
        img = requests.get(img_url)
        with open(file_path, "wb") as f:
            f.write(img.content)

# Main workflow loop
while has_work:
    if cooling_down:
        # We pulled data too fast so we need to let the API cool down
        print(
            "========> COMPLETED " + str(job_count) + " JOBS BEFORE COOL DOWN <========
"
        )
        time.sleep(60)
        cooling_down = False
        job_count = 0

    if endpoint == work_modes["4"]:
        # Reset
        response = requests.get(base_url + "reset")
        print("Reset complete")
        has_work = False
    else:
        # Get a new job
        job_count += 1
        request_url = base_url + endpoint
        if starts_with is not None:
            request_url = request_url + "?starts_with=" + starts_with

        job = requests.get(request_url)
        job = job.json()

        if job["id"] == "Done":
            has_work = False
```

```
83.
84.         if endpoint == work_modes["1"] and has_work:
85.             # Scrape observation data
86.             label = job["label"]
87.             print(label + " >> Scrapping " + job["url"])
88.             response = requests.get(job["url"])
89.             if response.status_code == 200:
90.                 response_json = response.json()
91.                 try:
92.                     img_url = response_json["results"][0]["photos"][0]["url"].replace(
93.                         "square", "large"
94.                     )
95.                     try:
96.                         lat = response_json["results"][0]["geojson"]["coordinates"][1]
97.                         lon = response_json["results"][0]["geojson"]["coordinates"][0]
98.                     except:
99.                         lat = None
100.                             lon = None
101.                        observed_on = response_json["results"][0]["observed_on"]
102.                        data = {
103.                            "img_url": img_url,
104.                            "lat": lat,
105.                            "lon": lon,
106.                            "observed_on": observed_on,
107.                            "scrapped": 1,
108.                        }
109.                    except:
110.                        data = {"scrapped": 1, "error": 1}
111.                    json_data = json.dumps(data)
112.                    response = requests.post(
113.                        base_url + "image_data?id=" + str(job["id"]),
114.                        data=json_data,
115.                        headers={"Content-Type": "application/json"},
116.                    )
117.                    if work_mode == "1":
118.                        # Download the image too
119.                        img_id_str = str(job["id"])
120.                        img_path = get_img_path(label)
121.                        download_image(img_url, img_path, img_id_str)
122.                        relay_image_downloaded(img_id_str)
123.                else:
124.                    cooling_down = True
125.
126.            if work_mode == "2" and has_work:
127.                # Download the image ONLY
128.                img_url = job["img_url"]
129.                img_id_str = str(job["id"])
130.                label = job["label"]
131.                img_path = get_img_path(label)
132.                download_image(img_url, img_path, img_id_str)
133.                relay_image_downloaded(img_id_str)
```

05 - Exploratory Data Analysis.ipynb

Excluded from the appendix because it is not relevant, however it is available on GitHub.

06 - Prep Data for Upload to the Cloud.ipynb

While this file is a jupyter notebook, I have displayed it below as if it were a Python script. The markdown text and the output of the cells are in comments with two hash marks. *Note: Some of the indentation starting on line 100 is incorrect and is an artifact of the tool I used to give the syntax highlighting and line numbers. Readers are encouraged to see the notebooks on GitHub if they find this illegible.*

```
1.  ## Prep Data for Upload to the Cloud
2.  ## Having labeled the data, we can prep it for uploading it to Google drive for further
     processing.
3.
4.  from glob import glob
5.  import os
6.  import shutil
7.
8.  n_images_per_class = 250
9.
10. def init_clean_dir(path):
11.     if os.path.exists(path):
12.         shutil.rmtree(path)
13.     os.mkdir(path)
14.
15. dataset_name = str(n_images_per_class) + "_images"
16. dataset_path = "./dataset/" + dataset_name
17. init_clean_dir(dataset_path)
18.
19. objects_per_image = {}
20. object_counts = {}
21. image_counts = {}
22. all_labels = {}
23.
24. class_file_path = "./iNaturalist/images/Poison Ivy/classes.txt"
25. shutil.copyfile(class_file_path, dataset_path + "/obj.names")
26.
27. # Collect a dictionary of all the labels
28. i = 0
29. for line in open(class_file_path):
30.     all_labels[str(i)] = line.strip()
31.     i += 1
32.
33. # Let's build the data set!
34. for line in open(class_file_path):
35.     the_label = line.strip()
36.     save_data_to = dataset_path + "/" + the_label.replace(" ", "_")
37.     init_clean_dir(save_data_to)
38.     pattern = "./iNaturalist/images/" + the_label + "/*.txt"
39.     labels = glob(pattern)
40.     labels.remove("./iNaturalist/images/" + the_label + "\\classes.txt")
41.     i = 0
42.     for coords_path in labels:
43.         # Check to see we are under the threshold
44.         if i < n_images_per_class:
45.             # Add this image
46.             i += 1
47.             record_id = int(coords_path.split("\\")[-1].split(".txt")[0])
48.             pattern = "./iNaturalist/images/" + the_label + "/*" + str(record_id) + "*"
```

```
            files = glob(pattern)
            image_path = [x for x in files if x not in coords_path][0]
            new_image_path = save_data_to + "/" + str(record_id) + ".jpg"
            new_coords_path = save_data_to + "/" + str(record_id) + ".txt"
            shutil.copyfile(image_path, new_image_path)
            shutil.copyfile(coords_path, new_coords_path)
            # Count the number of objects in this image
            for line in open(coords_path):
                label_id = line.split(" ")[0]
                this_label = all_labels[label_id]
                object_counts[this_label] = object_counts.get(this_label, 0) + 1
                ic = image_counts.get(this_label, set())
                ic.add(coords_path)
                image_counts[this_label] = ic
                objects_per_image[this_label] = objects_per_image.get(this_label, dict(
))
                objects_per_image[this_label][record_id] = objects_per_image[this_label
].get(record_id, 0) + 1

## What is the maximum number of objects in an image?

objects_per_image_counts = {}
for k, v in objects_per_image.items():
    objects_per_image_counts[k] = {}
    for x, k2 in v.items():
        objects_per_image_counts[k][k2] = objects_per_image_counts[k].get(k2, 0) + 1

max_object_count = 0
for k, v in objects_per_image_counts.items():
    for k2, v2 in v.items():
        if k2 > max_object_count:
            max_object_count = k2

max_object_count

## 35

## YOLOv5 Config
## Generate the YAML used to train the YOLOv5 model

names = "names: ["
n_classes = 0
file = open(class_file_path, "r")
for line in file.readlines():
    n_classes += 1
    names += "'" + line.strip() + "', "
names = names[:len(names)-2] + "]"

f = open(dataset_path + "yolov5.yaml", "w")
f.write("train: ../data/images/train/\r")
f.write("val: ../data/images/val/\r")
f.write("\r")
f.write("nc: " + str(n_classes) + "\r")
        f.write("\r")
        f.write(names)
        f.close()

        ## YOLOv4 Config

        yolov4_config = dataset_path + "/yolov4-custom.cfg"
        shutil.copyfile("YOLO config files/yolov4-custom.cfg", yolov4_config)
```

```
        lines = open(yolov4_config, "r").readlines()
        for line_num in [970, 1058, 1146]:
            lines[line_num] = "classes="+str(n_classes)+"\n"
        for line_num in [963, 1051, 1139]:
            lines[line_num] = "filters=" + str((n_classes + 5) * 3)+"\n"
        out = open(yolov4_config, 'w')
        out.writelines(lines)
        out.close()

        ## YOLOv3 Config

        yolov3_config = dataset_path + "/yolov3-spp.cfg"
        shutil.copyfile("YOLO config files/yolov3-spp.cfg", yolov3_config)
        lines = open(yolov3_config, "r").readlines()
        for line_num in [643, 729, 816]:
            lines[line_num] = "classes="+str(n_classes)+"\n"
        for line_num in [636, 722, 809]:
            lines[line_num] = "filters=" + str((n_classes + 5) * 3)+"\n"
        out = open(yolov3_config, 'w')
        out.writelines(lines)
        out.close()

        ## Zip it up
        ## Finally compress everything so it can be uploaded to the cloud

        shutil.make_archive("./dataset/" + dataset_name, "zip", dataset_path)
        shutil.rmtree(dataset_path)
        print("Ready for Upload!")
        ## Ready for Upload!
```

07 - Create Final Report Visualizations and Data.ipynb

This is another piece of source code that created images that were latter produced using other tools. It is available on the GitHub repository.

08 - Report Visualizations.rmd

This R Markdown file produced some of the figures found in this report.

```
---
title: "08 - Report Viz"
author: "Mike Silva"
date: "Fall 2020"
output: html_document
---

```{r, message=FALSE}
library(ggplot2)
library(dplyr)
library(tidyr)

setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

yolov3_results <- read.csv("report/yolov3_results.csv") %>%
  rowwise() %>%
  mutate(Epoch = as.numeric(strsplit(Epoch, "/")[[1]][1])+1)

yolov5_results <- read.csv("report/yolov5_results.csv") %>%
  rowwise() %>%
```
```

```
  mutate(Epoch = as.numeric(strsplit(Epoch, "/")[[1]][1])+1)
```

Images by Month Chart

```{r}
months_df <- read.csv("report/all_month_counts.csv") %>%
  mutate(Images = ifelse(Images == 0, NA, Images)) %>%
  mutate(Plant = ifelse(Plant.Type == "Poison Ivy", "A", "ERROR")) %>%
  mutate(Plant = ifelse(Plant.Type == "Box Elder", "B", Plant)) %>%
  mutate(Plant = ifelse(Plant.Type == "Bramble", "C", Plant)) %>%
  mutate(Plant = ifelse(Plant.Type == "Virginia Creeper", "D", Plant)) %>%
  mutate(Plant = as.factor(Plant)) %>%
  mutate(Plant = recode(Plant, A = "Poison Ivy", B = "Box Elder", C = "Bramble", D = "Virginia Creeper")) %>%
  select(-Plant.Type)

ggplot(months_df, aes(x=Month, y=Images, color=Plant, fill=Plant)) +
  geom_col() +
  scale_color_brewer(palette = "Set1") +
  scale_fill_brewer(palette = "Set1") +
  facet_wrap(~Plant) +
  theme(legend.position = "none") +
  #xlim(1, 12)
  scale_x_discrete(name ="Month",  limits=c("1","2","3", "4", "5", "6", "7", "8", "9", "10", "11", "12"))
```

Model Performance Charts

```{r}
viz_data <- yolov3_results %>% select(model, n_images, Epoch, mAP.0.5)
viz_data <- yolov5_results %>% select(model, n_images, Epoch, mAP.0.5) %>% rbind(viz_data)
viz_data$model <- as.factor(viz_data$model)
viz_data$n_images <- as.factor(viz_data$n_images)

ggplot(viz_data) +
  geom_line(aes(x=Epoch, y=mAP.0.5, color=n_images, group=n_images)) +
  facet_wrap(~model) +
  scale_color_brewer(palette = "Set1") +
  labs(color = "Number of Images") +
  ylab("mAP@0.5") +
  theme(legend.position = "bottom")
```

YOLOv3

```{r}
viz_data <- yolov3_results %>% select(n_images, Epoch, obj) %>% mutate(set = "Training") %>% rename(measure = obj)
viz_data <- yolov3_results %>% select(n_images, Epoch, Val.objectness) %>% mutate(set = "Test") %>% rename(measure = Val.objectness) %>% rbind(viz_data)
viz_data$n_images <- as.factor(viz_data$n_images)

ggplot(viz_data) +
  geom_line(aes(x=Epoch, y=measure, color=set, group=set)) +
  facet_wrap(~n_images) +
  scale_color_brewer(palette = "Set1") +
  labs(color = "Data Set") +
  theme(legend.position = "bottom") +
```

```
  ylab("Object Detection Error")
```


```{r}
viz_data <- yolov3_results %>% select(n_images, Epoch, cls) %>% mutate(set = "Training") %>% rename(measure = cls)
viz_data <- yolov3_results %>% select(n_images, Epoch, Val.classification) %>% mutate(set = "Test") %>% rename(measure = Val.classification) %>% rbind(viz_data)
viz_data$n_images <- as.factor(viz_data$n_images)

ggplot(viz_data) +
  geom_line(aes(x=Epoch, y=measure, color=set, group=set)) +
  facet_wrap(~n_images) +
  scale_color_brewer(palette = "Set1") +
  labs(color = "Data Set") +
  theme(legend.position = "bottom") +
  ylab("Classification Error")
```

YOLOv5

```{r}
viz_data <- yolov5_results %>% select(n_images, Epoch, obj) %>% mutate(set = "Training") %>% rename(measure = obj)
viz_data <- yolov5_results %>% select(n_images, Epoch, Val.objectness) %>% mutate(set = "Test") %>% rename(measure = Val.objectness) %>% rbind(viz_data)
      viz_data$n_images <- as.factor(viz_data$n_images)

      ggplot(viz_data) +
        geom_line(aes(x=Epoch, y=measure, color=set, group=set)) +
        facet_wrap(~n_images) +
        scale_color_brewer(palette = "Set1") +
        labs(color = "Data Set") +
        theme(legend.position = "bottom") +
        ylab("Object Detection Error")
      ```


      ```{r}
      viz_data <- yolov5_results %>% select(n_images, Epoch, cls) %>% mutate(set = "Training") %>% rename(measure = cls)
      viz_data <- yolov5_results %>% select(n_images, Epoch, Val.classification) %>% mutate(set = "Test") %>% rename(measure = Val.classification) %>% rbind(viz_data)
      viz_data$n_images <- as.factor(viz_data$n_images)

      ggplot(viz_data) +
        geom_line(aes(x=Epoch, y=measure, color=set, group=set)) +
        facet_wrap(~n_images) +
        scale_color_brewer(palette = "Set1") +
        labs(color = "Data Set") +
        theme(legend.position = "bottom") +
        ylab("Classification Error")
      ```


      ```{r}
      yolo_results <- pivot_longer(yolov3_results, GIoU:Val.classification, names_to = "Measure")
      yolo_results <- pivot_longer(yolov5_results, GIoU:Val.classification, names_to = "Measure") %>%
```

```
        rbind(yolo_results)

      for(m in unique(yolo_results$Measure)){
        viz_data <- yolo_results %>%
          filter(Measure == m) %>%
          mutate(Model = as.factor(paste0(model,": ", n_images, " images")))
          p <- ggplot(viz_data, aes(x=Epoch, y=value, color=Model, group=Model)) +
            geom_line() +
            theme(axis.title.y = element_blank()) +
            ggtitle(m)
          print(p)
      }
      ```

      ```{r}
      ggplot(yolov3_results, aes(R, P, color=as.factor(paste(model, n_images)))) + geo
  m_point()
      ```
```

## *09 - Process Field Test Results.py*

This script extracts the results from the field test from the Google Colab notebook and saves it as a CSV.

```
-*- coding: utf-8 -*-
"""
Created on Fri Nov 20 09:35:33 2020

@author: Michael
"""
import pandas as pd
import json

yolov3_cells = {250: 19, 500: 22, 1000: 25, 2000: 28}
yolov5_cells = {250: 14, 500: 17, 1000: 20, 2000: 23}
classes = ["Virginia Creeper", "Poison Ivy", "Brambles", "Box Elder"]

def extract_data_from_cell(cell, n_images, model):
    data = []
    for c in cell:
        if "text" in c:
            for line in c["text"]:
                if "image 1/1" in line:
                    img = line.split(":")[0].split("/")[-1]
                    found = {"Virginia Creeper":0, "Poison Ivy": 0, "Brambles":0, "Box
  Elder":0, "Null":0}
                    findings = line.split(":")[1]
                    for cl in classes:
                        if cl in findings:
                            found[cl] = 1
                    if len(findings.split(",")) == 1:
                        found["Null"] = 1
                    found["Image"] = img
                    found["n_images"] = n_images
                    found["Model"] = model
                    data.append(found)
    return data

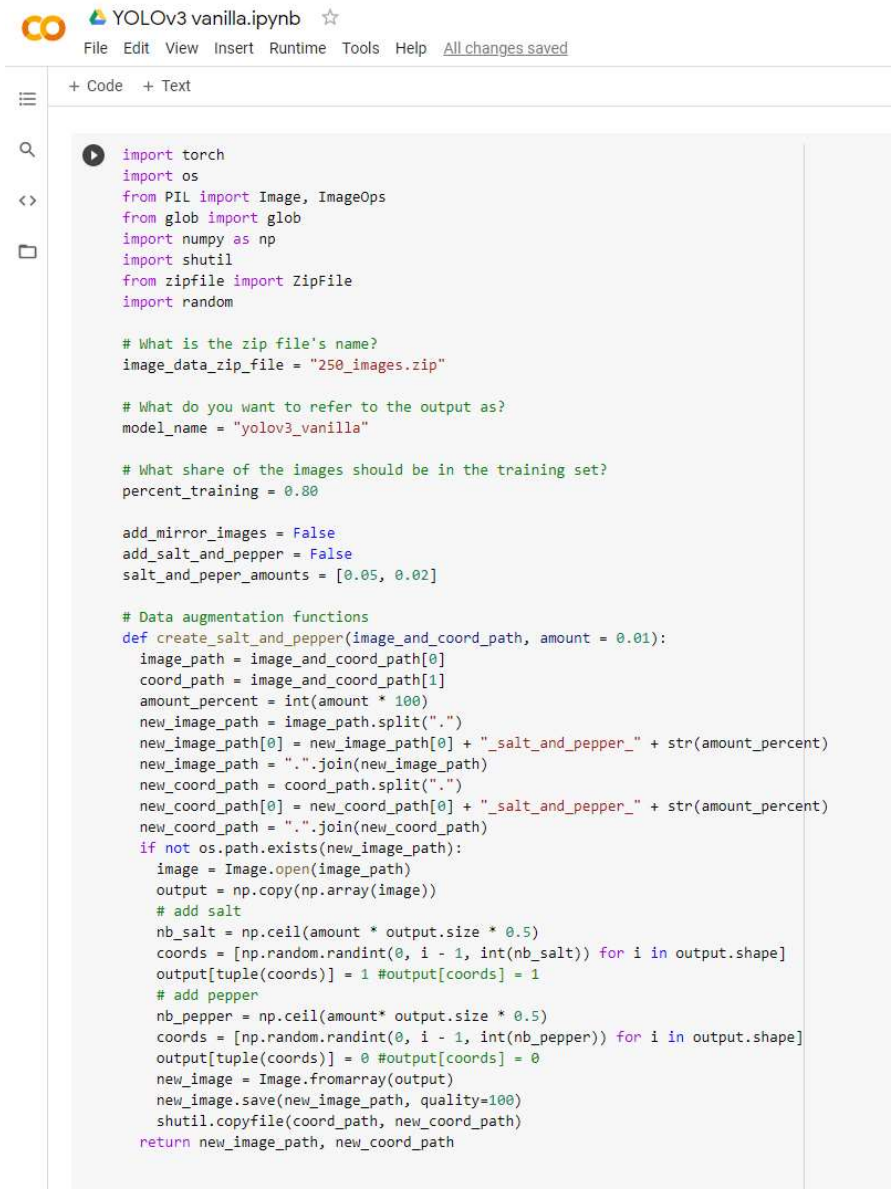
def extract_data_from_yolov5_cell(cell):
```
```

```
    data = {}
    return data


# Process the YOLOv3 Results
with open('./results/YOLOv3 Field Tests Results.ipynb') as json_file:
    yolov3 = json.load(json_file)["cells"]


for n_images, key in yolov3_cells.items():
    cell = yolov3[key]["outputs"]
    data = extract_data_from_cell(cell, n_images, "YOLOv3")
    try:
        yolo_data = yolo_data + data.copy()
    except:
        yolo_data = data.copy()


# Process the YOLOv5 Results
with open('./results/YOLOv5 Field Tests Results.ipynb') as json_file:
    yolov5 = json.load(json_file)["cells"]

for n_images, key in yolov5_cells.items():
    cell = yolov5[key]["outputs"]
    data = extract_data_from_cell(cell, n_images, "YOLOv5")
    yolo_data = yolo_data + data.copy()


df = pd.DataFrame(yolo_data)
df.to_csv("./report/field_test_results.csv", index = False)
```

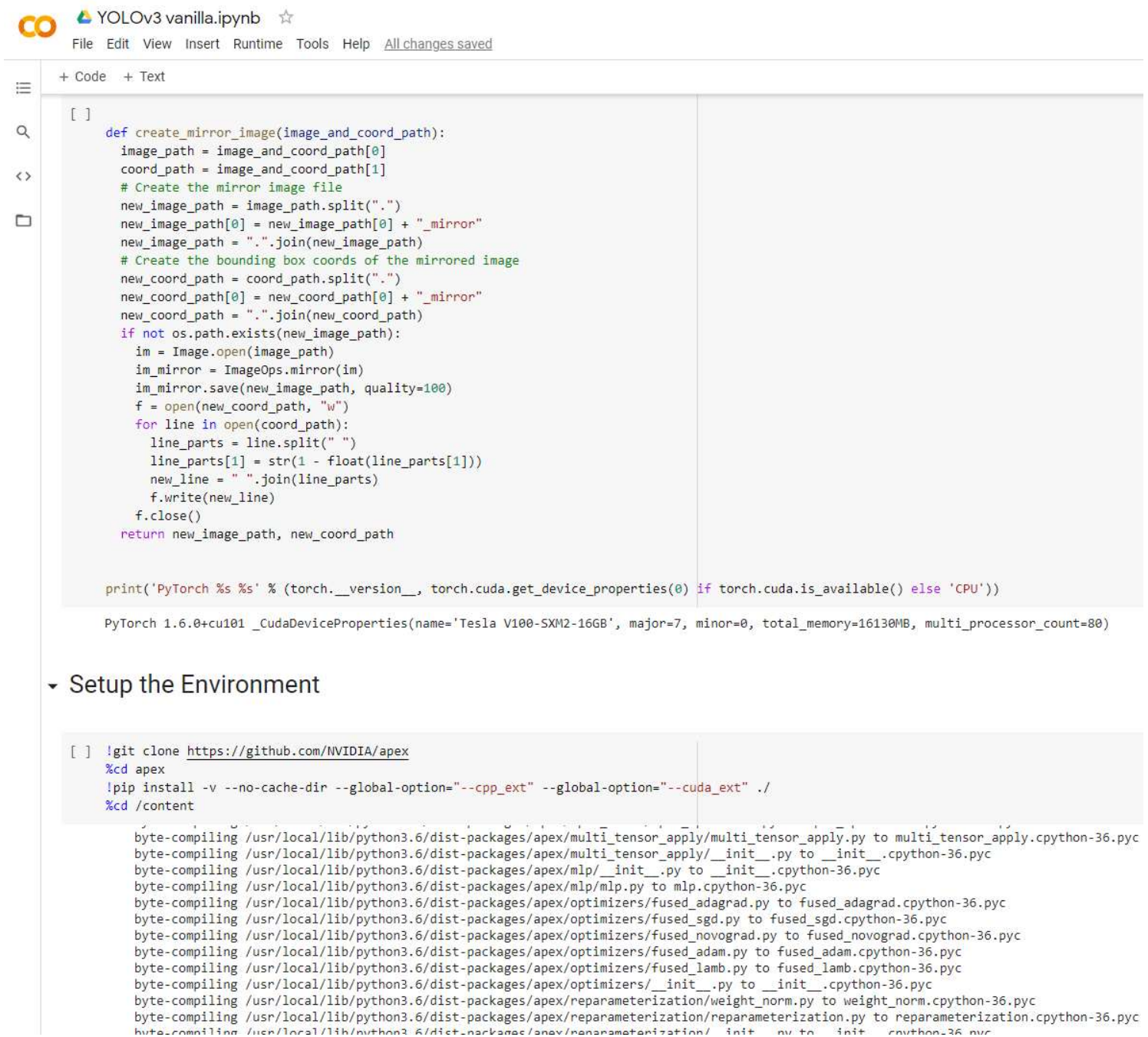
YOLO Models Notebooks

The Google Colab notebooks were set up so that the same notebook could be used to generate all model variants by changing a few variables in the script. Consequently, I am only including one instance of the YOLO v3 notebook and YOLOv5 notebook. All notebooks are on the GitHub repository in the “results” folder. These notebooks also contain commands to set up the environment. The “code” for these models are screenshots taken directly from Google Colab.

YOLO v3




```
import torch
import os
from PIL import Image, ImageOps
from glob import glob
import numpy as np
import shutil
from zipfile import ZipFile
import random

# What is the zip file's name?
image_data_zip_file = "250_images.zip"

# What do you want to refer to the output as?
model_name = "yolov3_vanilla"

# What share of the images should be in the training set?
percent_training = 0.80

add_mirror_images = False
add_salt_and_pepper = False
salt_and_peper_amounts = [0.05, 0.02]

# Data augmentation functions
def create_salt_and_pepper(image_and_coord_path, amount = 0.01):
  image_path = image_and_coord_path[0]
  coord_path = image_and_coord_path[1]
  amount_percent = int(amount * 100)
  new_image_path = image_path.split(".")
  new_image_path[0] = new_image_path[0] + "_salt_and_pepper_" + str(amount_percent)
  new_image_path = ".".join(new_image_path)
  new_coord_path = coord_path.split(".")
  new_coord_path[0] = new_coord_path[0] + "_salt_and_pepper_" + str(amount_percent)
  new_coord_path = ".".join(new_coord_path)
  if not os.path.exists(new_image_path):
    image = Image.open(image_path)
    output = np.copy(np.array(image))
    # add salt
    nb_salt = np.ceil(amount * output.size * 0.5)
    coords = [np.random.randint(0, i - 1, int(nb_salt)) for i in output.shape]
    output[tuple(coords)] = 1 #output[coords] = 1
    # add pepper
    nb_pepper = np.ceil(amount* output.size * 0.5)
    coords = [np.random.randint(0, i - 1, int(nb_pepper)) for i in output.shape]
    output[tuple(coords)] = 0 #output[coords] = 0
    new_image = Image.fromarray(output)
    new_image.save(new_image_path, quality=100)
    shutil.copyfile(coord_path, new_coord_path)
  return new_image_path, new_coord_path
```



YOLOv3 vanilla.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

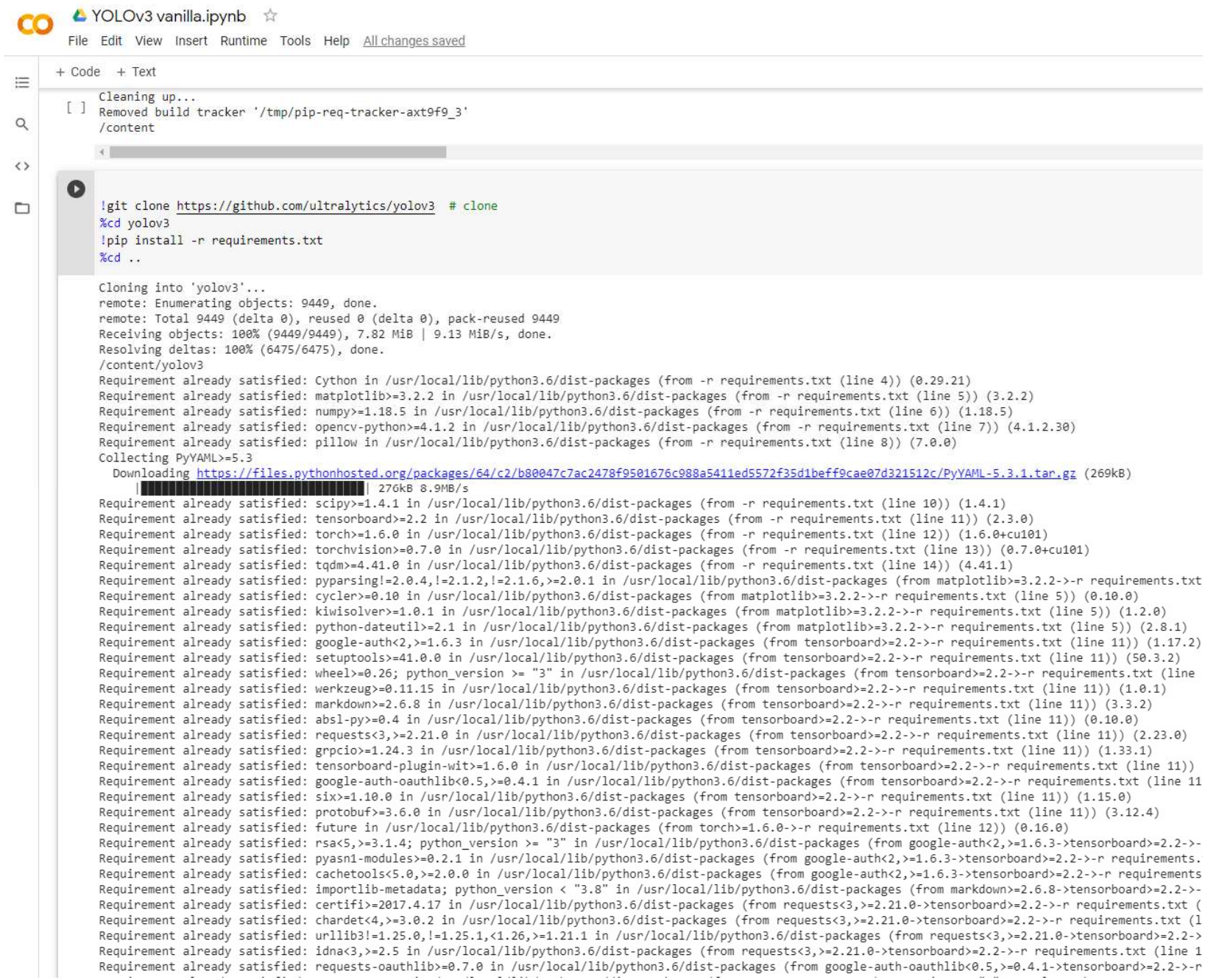
```
[ ]
    def create_mirror_image(image_and_coord_path):
      image_path = image_and_coord_path[0]
      coord_path = image_and_coord_path[1]
      # Create the mirror image file
      new_image_path = image_path.split(".")
      new_image_path[0] = new_image_path[0] + "_mirror"
      new_image_path = ".".join(new_image_path)
      # Create the bounding box coords of the mirrored image
      new_coord_path = coord_path.split(".")
      new_coord_path[0] = new_coord_path[0] + "_mirror"
      new_coord_path = ".".join(new_coord_path)
      if not os.path.exists(new_image_path):
        im = Image.open(image_path)
        im_mirror = ImageOps.mirror(im)
        im_mirror.save(new_image_path, quality=100)
        f = open(new_coord_path, "w")
        for line in open(coord_path):
          line_parts = line.split(" ")
          line_parts[1] = str(1 - float(line_parts[1]))
          new_line = " ".join(line_parts)
          f.write(new_line)
        f.close()
      return new_image_path, new_coord_path


    print('PyTorch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

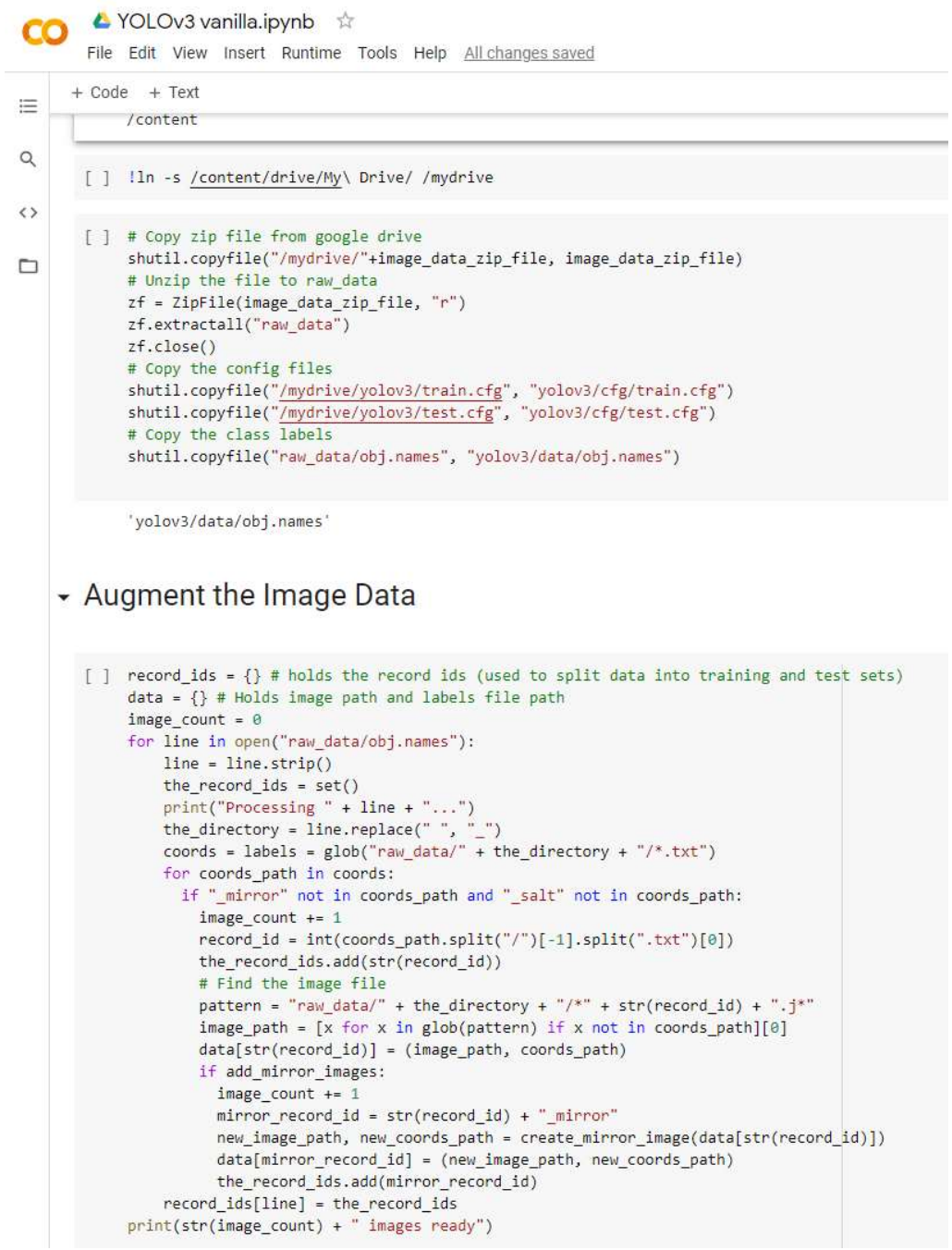
```
PyTorch 1.6.0+cu101 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', major=7, minor=0, total_memory=16130MB, multi_processor_count=80)
```

Setup the Environment

```
[ ] !git clone https://github.com/NVIDIA/apex
    %cd apex
    !pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext" ./
    %cd /content
```

```
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/multi_tensor_apply/multi_tensor_apply.py to multi_tensor_apply.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/multi_tensor_apply/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/mlp/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/mlp/mlp.py to mlp.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/optimizers/fused_adagrad.py to fused_adagrad.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/optimizers/fused_sgd.py to fused_sgd.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/optimizers/fused_novograd.py to fused_novograd.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/optimizers/fused_adam.py to fused_adam.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/optimizers/fused_lamb.py to fused_lamb.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/optimizers/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/reparameterization/weight_norm.py to weight_norm.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/reparameterization/reparameterization.py to reparameterization.cpython-36.pyc
```


YOLOv3 vanilla.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[ ] Cleaning up...
Removed build tracker '/tmp/pip-req-tracker-axt9f9_3'
/content
!git clone https://github.com/ultralytics/yolov3 # clone
%cd yolov3
!pip install -r requirements.txt
%cd ..
Cloning into 'yolov3'...
remote: Enumerating objects: 9449, done.
remote: Total 9449 (delta 0), reused 0 (delta 0), pack-reused 9449
Receiving objects: 100% (9449/9449), 7.82 MiB | 9.13 MiB/s, done.
Resolving deltas: 100% (6475/6475), done.
/content/yolov3
Requirement already satisfied: Cython in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 4)) (0.29.21)
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 5)) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 6)) (1.18.5)
Requirement already satisfied: opencv-python>=4.1.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 7)) (4.1.2.30)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 8)) (7.0.0)
Collecting PyYAML>=5.3
Downloading https://files.pythonhosted.org/packages/64/c2/b80047c7ac2478f9501676c988a5411ed5572f35d1beff9cae07d321512c/PyYAML-5.3.1.tar.gz (269kB)
276kB 8.9MB/s
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 10)) (1.4.1)
Requirement already satisfied: tensorboard>=2.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 11)) (2.3.0)
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 12)) (1.6.0+cu101)
Requirement already satisfied: torchvision>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 13)) (0.7.0+cu101)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 14)) (4.41.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (1.2.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (2.8.1)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.17.2)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (50.3.2)
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.0.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (3.3.2)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (0.10.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (2.23.0)
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.33.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11))
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.15.0)
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (3.12.4)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from torch>=1.6.0->-r requirements.txt (line 12)) (0.16.0)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard>=2.2->-
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard>=2.2->-r requirements.
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard>=2.2->-r requirements
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard>=2.2->-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (l
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (line 1
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard>=2.2->-r


YOLOv3 vanilla.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
/content
[ ] !ln -s /content/drive/My\ Drive/ /mydrive
[ ] # Copy zip file from google drive
shutil.copyfile("/mydrive/"+image_data_zip_file, image_data_zip_file)
# Unzip the file to raw_data
zf = ZipFile(image_data_zip_file, "r")
zf.extractall("raw_data")
zf.close()
# Copy the config files
shutil.copyfile("/mydrive/yolov3/train.cfg", "yolov3/cfg/train.cfg")
shutil.copyfile("/mydrive/yolov3/test.cfg", "yolov3/cfg/test.cfg")
# Copy the class labels
shutil.copyfile("raw_data/obj.names", "yolov3/data/obj.names")
'yolov3/data/obj.names'
Augment the Image Data
[ ] record_ids = {} # holds the record ids (used to split data into training and test sets)
data = {} # Holds image path and labels file path
image_count = 0
for line in open("raw_data/obj.names"):
line = line.strip()
the_record_ids = set()
print("Processing " + line + "...")
the_directory = line.replace(" ", "_")
coords = labels = glob("raw_data/" + the_directory + "/*.txt")
for coords_path in coords:
if "_mirror" not in coords_path and "_salt" not in coords_path:
image_count += 1
record_id = int(coords_path.split("/")[-1].split(".txt")[0])
the_record_ids.add(str(record_id))
# Find the image file
pattern = "raw_data/" + the_directory + "/*" + str(record_id) + ".j*"
image_path = [x for x in glob(pattern) if x not in coords_path][0]
data[str(record_id)] = (image_path, coords_path)
if add_mirror_images:
image_count += 1
mirror_record_id = str(record_id) + "_mirror"
new_image_path, new_coords_path = create_mirror_image(data[str(record_id)])
data[mirror_record_id] = (new_image_path, new_coords_path)
the_record_ids.add(mirror_record_id)
record_ids[line] = the_record_ids
print(str(image_count) + " images ready")

YOLOv3 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] Processing Poison Ivy...
    Processing Virginia Creeper...
    Processing Bramble...
    Processing Box Elder...
    1000 images ready
```

```
[ ] if add_salt_and_pepper:
      for amount in salt_and_peper_amounts:
        for line in open("raw_data/obj.names"):
          line = line.strip()
          print("Processing " + line + "...")
          the_directory = line.strip().replace(" ", "_")
          the_record_ids = record_ids[line]
          for record_id in the_record_ids.copy():
            salt_and_peper_record_id = record_id + "_salt_and_pepper_" + str(int(amount * 100))
            the_record_ids.add(salt_and_peper_record_id)
            data[salt_and_peper_record_id] = create_salt_and_pepper(data[record_id], amount)
            image_count += 1
          record_ids[line] = the_record_ids
    print(str(image_count) + " images ready")
```

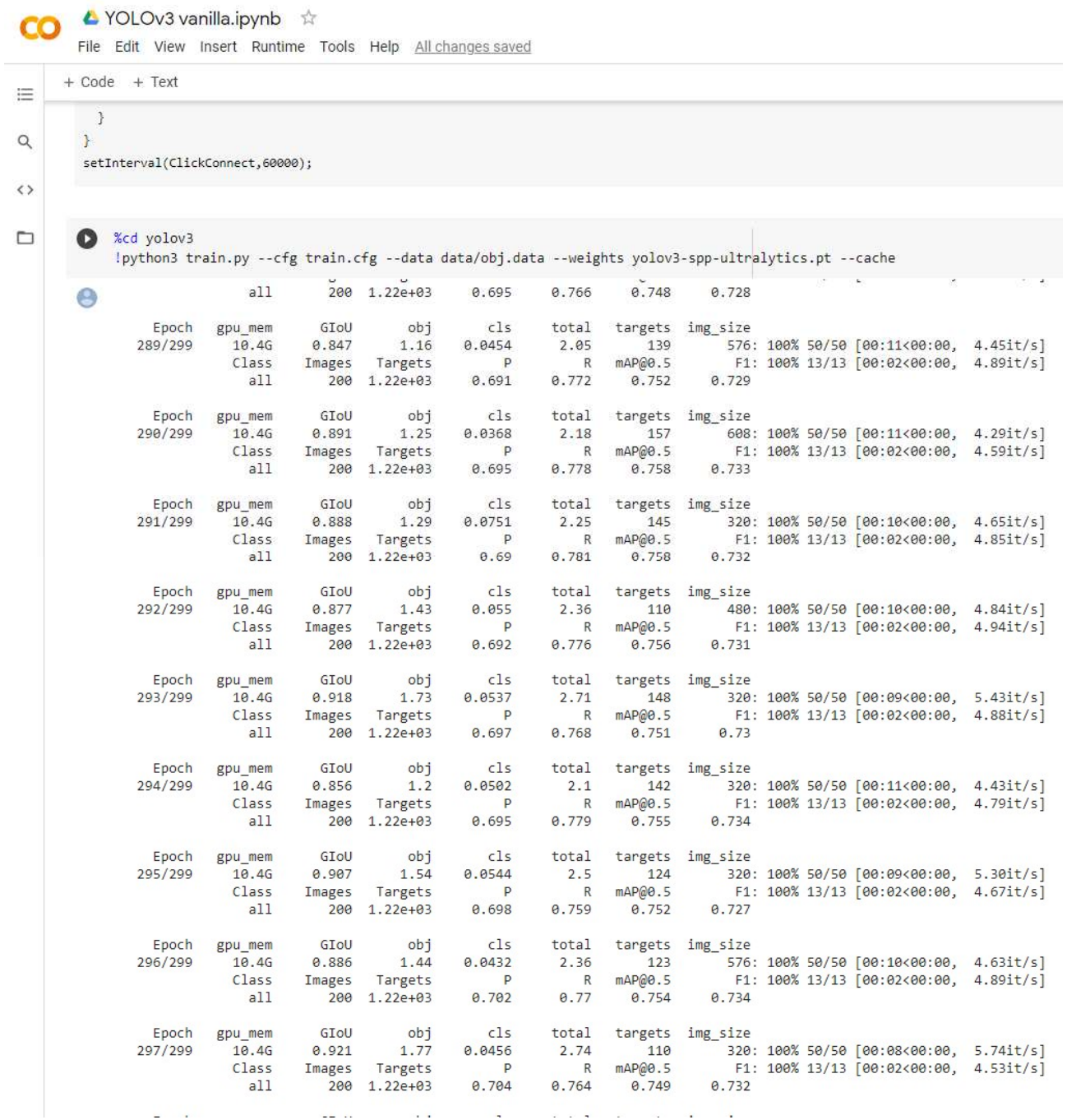
```
    1000 images ready
```

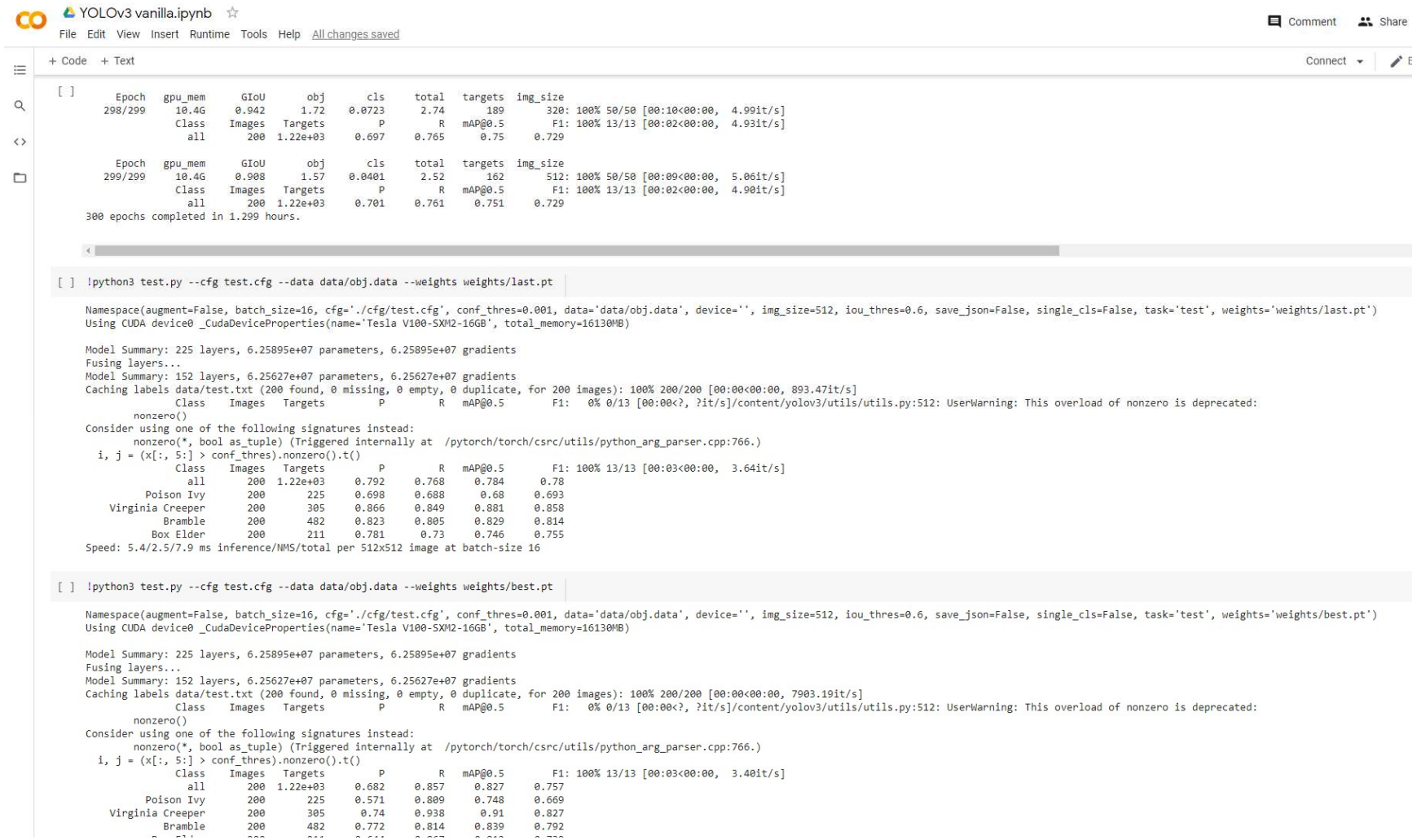
Train Test Split

```
[ ] !mkdir yolov3/data/training
    !mkdir yolov3/data/test
```

```
[ ] random.seed(42)
    training_image_paths = set()
    test_image_paths = set()
    for line in open("raw_data/obj.names"):
        line = line.strip()
        the_record_ids = record_ids[line]
        n_train = int(round(len(the_record_ids) * percent_training, 0))
        for_training = random.sample(list(the_record_ids), n_train)
        for_testing = np.setdiff1d(list(the_record_ids), for_training)
        for i in for_training:
          for file_path in data[i]:
            new_file_path = file_path.split("/")
            new_file_path = "yolov3/data/training/" + new_file_path[-1]
            shutil.move(file_path, new_file_path)
            if "jpg" in new_file_path:
              training_image_paths.add(new_file_path)
        for i in for_testing:
          for file_path in data[i]:
            new_file_path = file_path.split("/")
            new_file_path = "yolov3/data/test/" + new_file_path[-1]
```


YOLOv3 vanilla.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[ ]
new_file_path = "yolov3/data/test/" + new_file_path[-1]
shutil.move(file_path, new_file_path)
if "jpg" in new_file_path:
test_image_paths.add(new_file_path)
Finalize the Config Files
[ ] n_classes = 0
for line in open("raw_data/obj.names"):
n_classes += 1
f = open("yolov3/data/obj.data", "w")
f.write("classes=" + str(n_classes) + "\n")
f.write("train=data/training.txt\n")
f.write("valid=data/test.txt\n")
f.write("names=data/obj.names")
f.close()
[ ] f = open("/content/yolov3/data/training.txt", "w")
for training_image_path in training_image_paths:
training_image_path = training_image_path.replace("yolov3/data/", "./")
f.write(training_image_path + "\n")
f.close()
[ ] f = open("/content/yolov3/data/test.txt", "w")
for test_image_path in test_image_paths:
test_image_path = test_image_path.replace("yolov3/data", "./")
f.write(test_image_path + "\n")
f.close()
Train the Model
var keep_alive = false;
function ClickConnect(){
if (keep_alive){
console.log("Working");
document
.querySelector('#top-toolbar > colab-connect-button')
.shadowRoot.querySelector('#connect')
.click();


YOLOv3 vanilla.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
}
}
setInterval(ClickConnect,60000);
%cd yolov3
!python3 train.py --cfg train.cfg --data data/obj.data --weights yolov3-spp-ultralytics.pt --cache
all 200 1.22e+03 0.695 0.766 0.748 0.728
Epoch gpu_mem GIoU obj cls total targets img_size
289/299 10.4G 0.847 1.16 0.0454 2.05 139 576: 100% 50/50 [00:11<00:00, 4.45it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.89it/s]
all 200 1.22e+03 0.691 0.772 0.752 0.729
Epoch gpu_mem GIoU obj cls total targets img_size
290/299 10.4G 0.891 1.25 0.0368 2.18 157 608: 100% 50/50 [00:11<00:00, 4.29it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.59it/s]
all 200 1.22e+03 0.695 0.778 0.758 0.733
Epoch gpu_mem GIoU obj cls total targets img_size
291/299 10.4G 0.888 1.29 0.0751 2.25 145 320: 100% 50/50 [00:10<00:00, 4.65it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.85it/s]
all 200 1.22e+03 0.69 0.781 0.758 0.732
Epoch gpu_mem GIoU obj cls total targets img_size
292/299 10.4G 0.877 1.43 0.055 2.36 110 480: 100% 50/50 [00:10<00:00, 4.84it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.94it/s]
all 200 1.22e+03 0.692 0.776 0.756 0.731
Epoch gpu_mem GIoU obj cls total targets img_size
293/299 10.4G 0.918 1.73 0.0537 2.71 148 320: 100% 50/50 [00:09<00:00, 5.43it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.88it/s]
all 200 1.22e+03 0.697 0.768 0.751 0.73
Epoch gpu_mem GIoU obj cls total targets img_size
294/299 10.4G 0.856 1.2 0.0502 2.1 142 320: 100% 50/50 [00:11<00:00, 4.43it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.79it/s]
all 200 1.22e+03 0.695 0.779 0.755 0.734
Epoch gpu_mem GIoU obj cls total targets img_size
295/299 10.4G 0.907 1.54 0.0544 2.5 124 320: 100% 50/50 [00:09<00:00, 5.30it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.67it/s]
all 200 1.22e+03 0.698 0.759 0.752 0.727
Epoch gpu_mem GIoU obj cls total targets img_size
296/299 10.4G 0.886 1.44 0.0432 2.36 123 576: 100% 50/50 [00:10<00:00, 4.63it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.89it/s]
all 200 1.22e+03 0.702 0.77 0.754 0.734
Epoch gpu_mem GIoU obj cls total targets img_size
297/299 10.4G 0.921 1.77 0.0456 2.74 110 320: 100% 50/50 [00:08<00:00, 5.74it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.53it/s]
all 200 1.22e+03 0.704 0.764 0.749 0.732


YOLOv3 vanilla.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
[ ]
Epoch gpu_mem GIoU obj cls total targets img_size
298/299 10.4G 0.942 1.72 0.0723 2.74 189 320: 100% 50/50 [00:10<00:00, 4.99it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.93it/s]
all 200 1.22e+03 0.697 0.765 0.75 0.729
Epoch gpu_mem GIoU obj cls total targets img_size
299/299 10.4G 0.908 1.57 0.0401 2.52 162 512: 100% 50/50 [00:09<00:00, 5.06it/s]
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:02<00:00, 4.90it/s]
all 200 1.22e+03 0.701 0.761 0.751 0.729
300 epochs completed in 1.299 hours.
[ ] !python3 test.py --cfg test.cfg --data data/obj.data --weights weights/last.pt
Namespace(augment=False, batch_size=16, cfg='./cfg/test.cfg', conf_thres=0.001, data='data/obj.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls=False, task='test', weights='weights/last.pt')
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)
Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25627e+07 parameters, 6.25627e+07 gradients
Caching labels data/test.txt (200 found, 0 missing, 0 empty, 0 duplicate, for 200 images): 100% 200/200 [00:00<00:00, 893.47it/s]
Class Images Targets P R mAP@0.5 F1: 0% 0/13 [00:00<?, ?it/s]/content/yolov3/utils/utils.py:512: UserWarning: This overload of nonzero is deprecated:
nonzero()
Consider using one of the following signatures instead:
nonzero(*, bool as_tuple) (Triggered internally at /pytorch/torch/csrc/utils/python_arg_parser.cpp:766.)
i, j = (x[:, 5:] > conf_thres).nonzero().t()
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:03<00:00, 3.64it/s]
all 200 1.22e+03 0.792 0.768 0.784 0.78
Poison Ivy 200 225 0.698 0.688 0.68 0.693
Virginia Creeper 200 305 0.866 0.849 0.881 0.858
Bramble 200 482 0.823 0.805 0.829 0.814
Box Elder 200 211 0.781 0.73 0.746 0.755
Speed: 5.4/2.5/7.9 ms inference/NMS/total per 512x512 image at batch-size 16
[ ] !python3 test.py --cfg test.cfg --data data/obj.data --weights weights/best.pt
Namespace(augment=False, batch_size=16, cfg='./cfg/test.cfg', conf_thres=0.001, data='data/obj.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls=False, task='test', weights='weights/best.pt')
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)
Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25627e+07 parameters, 6.25627e+07 gradients
Caching labels data/test.txt (200 found, 0 missing, 0 empty, 0 duplicate, for 200 images): 100% 200/200 [00:00<00:00, 7903.19it/s]
Class Images Targets P R mAP@0.5 F1: 0% 0/13 [00:00<?, ?it/s]/content/yolov3/utils/utils.py:512: UserWarning: This overload of nonzero is deprecated:
nonzero()
Consider using one of the following signatures instead:
nonzero(*, bool as_tuple) (Triggered internally at /pytorch/torch/csrc/utils/python_arg_parser.cpp:766.)
i, j = (x[:, 5:] > conf_thres).nonzero().t()
Class Images Targets P R mAP@0.5 F1: 100% 13/13 [00:03<00:00, 3.40it/s]
all 200 1.22e+03 0.682 0.857 0.827 0.757
Poison Ivy 200 225 0.571 0.809 0.748 0.669
Virginia Creeper 200 305 0.74 0.938 0.91 0.827
Bramble 200 482 0.772 0.814 0.839 0.792

YOLOv3 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

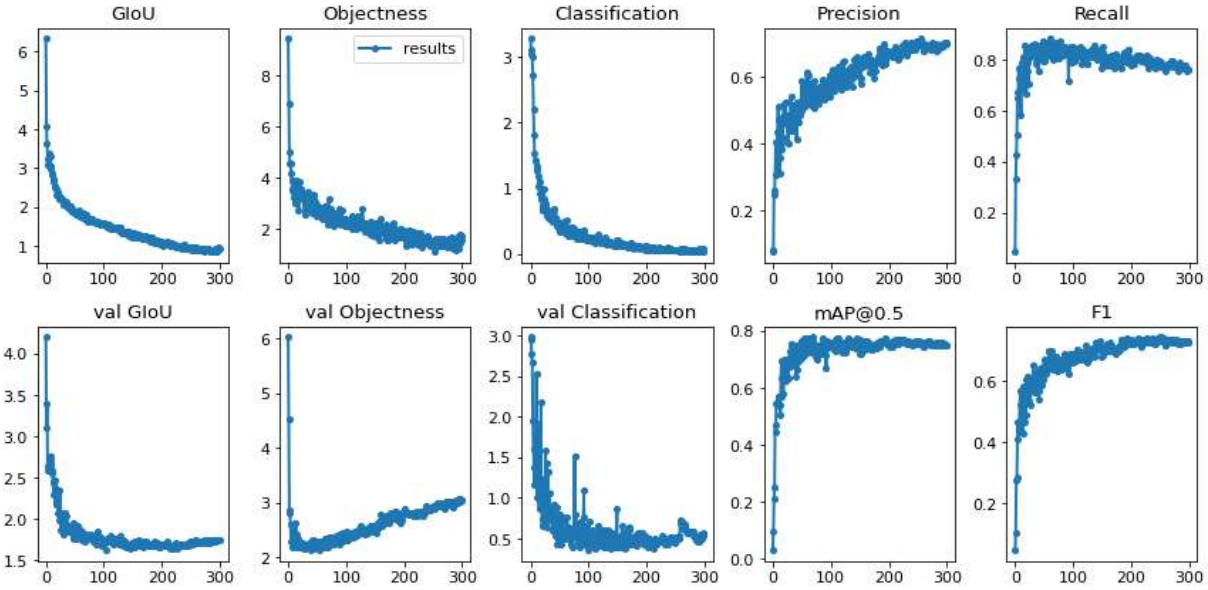
+ Code + Text

```
[ ]                Box Elder       200       211      0.644      0.867      0.812      0.739
Speed: 5.5/2.3/7.7 ms inference/NMS/total per 512x512 image at batch-size 16
```

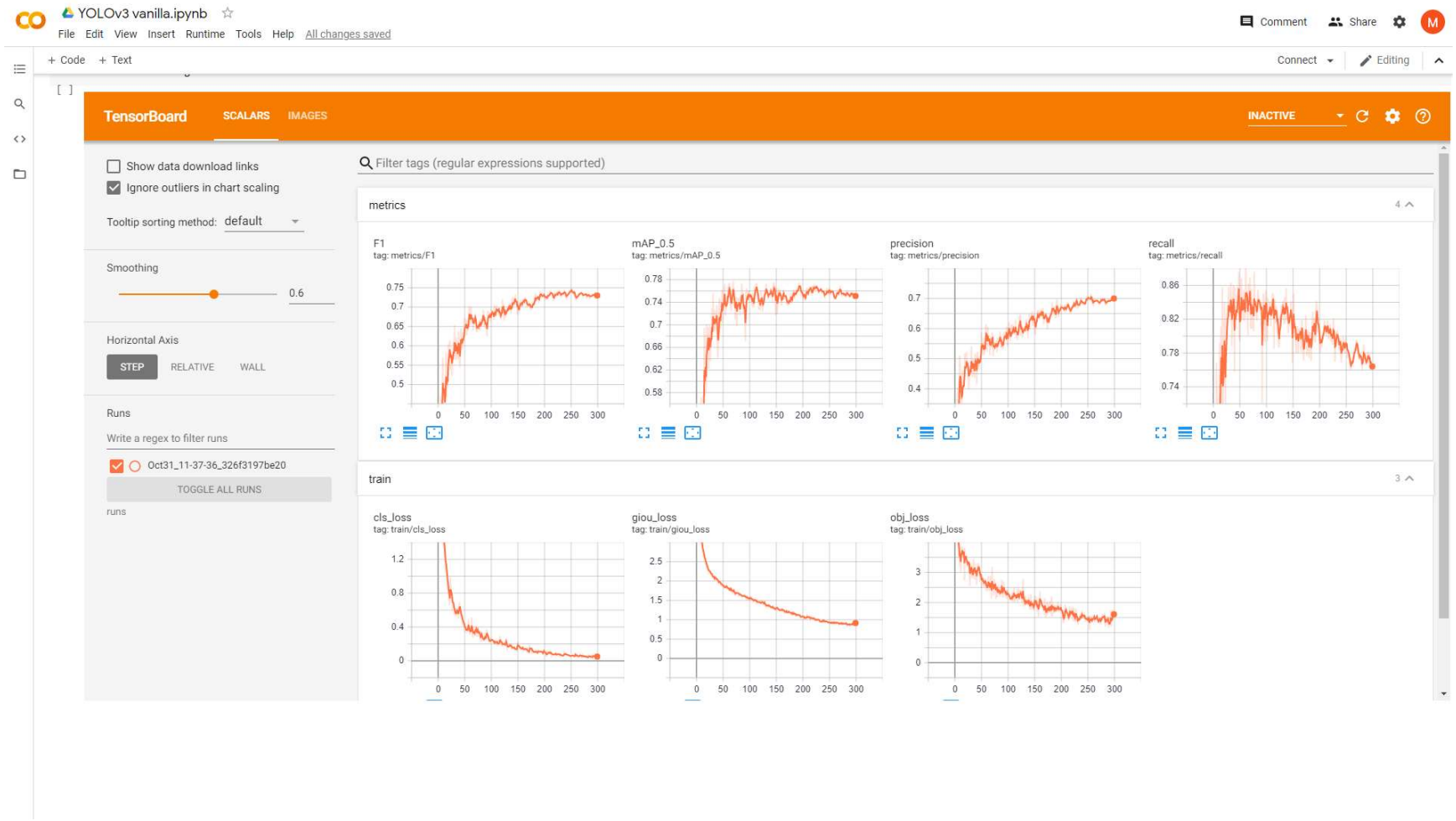
```
[ ] os.mkdir("/mydrive/yolov3/" + model_name)
shutil.copy("/content/yolov3/results.txt", "/mydrive/yolov3/" + model_name + "/results.txt")
shutil.copy("/content/yolov3/weights/last.pt", "/mydrive/yolov3/" + model_name + "/last.pt")
shutil.copy("/content/yolov3/weights/best.pt", "/mydrive/yolov3/" + model_name + "/best.pt")
```

```
'/mydrive/yolov3/yolov3_vanilla/best.pt'
```

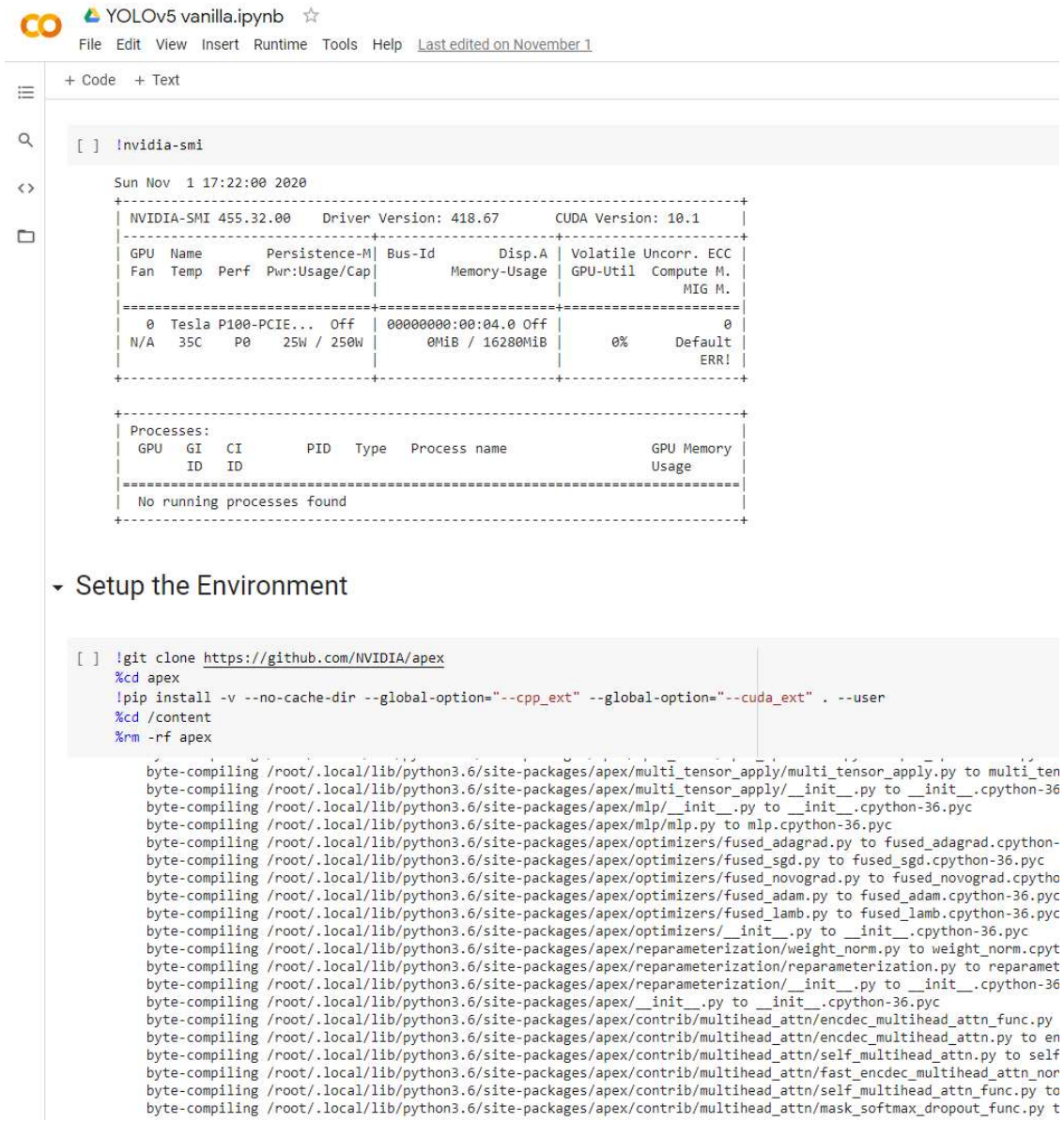
```
[ ] from utils import utils
utils.plot_results()
```




```
[ ] %load_ext tensorboard
%tensorboard --logdir runs
```


YOLOv3 vanilla.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share M
+ Code + Text
Connect Editing
TensorBoard SCALARS IMAGES
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP RELATIVE WALL
Runs
Write a regex to filter runs
Oct31_11-37-36_326f3197be20
TOGGLE ALL RUNS
runs
Filter tags (regular expressions supported)
metrics
F1
tag: metrics/F1
mAP_0.5
tag: metrics/mAP_0.5
precision
tag: metrics/precision
recall
tag: metrics/recall
train
cls_loss
tag: train/cls_loss
giou_loss
tag: train/giou_loss
obj_loss
tag: train/obj_loss

YOLO v5



YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

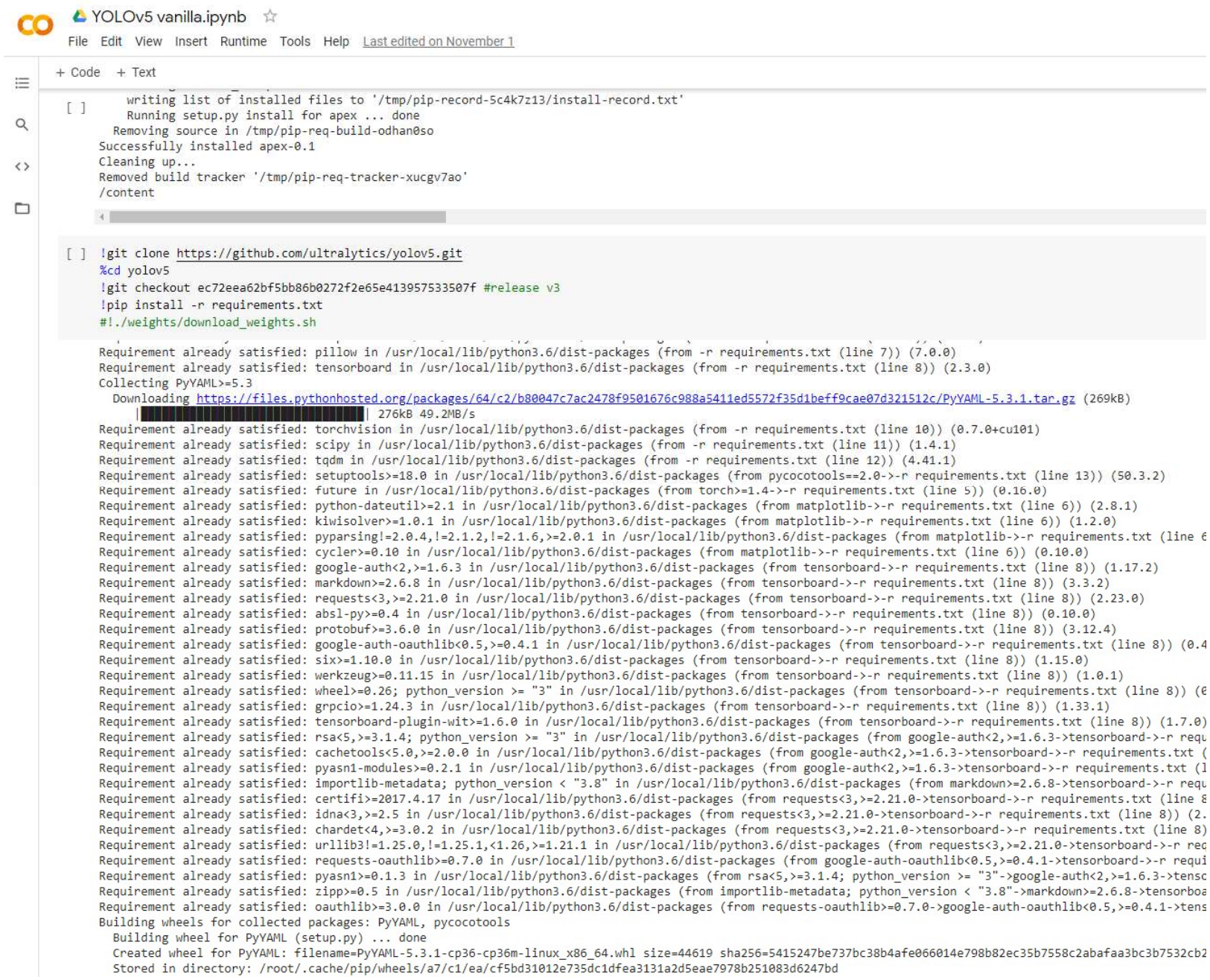
```
[ ] !nvidia-smi
```

```
Sun Nov  1 17:22:00 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.32.00    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   35C    P0    25W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Setup the Environment

```
[ ] !git clone https://github.com/NVIDIA/apex
    %cd apex
    !pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext" . --user
    %cd /content
    %rm -rf apex
```

```
byte-compiling /root/.local/lib/python3.6/site-packages/apex/multi_tensor_apply/multi_tensor_apply.py to multi_ten
byte-compiling /root/.local/lib/python3.6/site-packages/apex/multi_tensor_apply/__init__.py to __init__.cpython-36
byte-compiling /root/.local/lib/python3.6/site-packages/apex/mlp/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/mlp/mlp.py to mlp.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/optimizers/fused_adagrad.py to fused_adagrad.cpython-
byte-compiling /root/.local/lib/python3.6/site-packages/apex/optimizers/fused_sgd.py to fused_sgd.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/optimizers/fused_novograd.py to fused_novograd.cpytho
byte-compiling /root/.local/lib/python3.6/site-packages/apex/optimizers/fused_adam.py to fused_adam.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/optimizers/fused_lamb.py to fused_lamb.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/optimizers/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/reparameterization/weight_norm.py to weight_norm.cpyt
byte-compiling /root/.local/lib/python3.6/site-packages/apex/reparameterization/reparameterization.py to reparamet
byte-compiling /root/.local/lib/python3.6/site-packages/apex/reparameterization/__init__.py to __init__.cpython-36
byte-compiling /root/.local/lib/python3.6/site-packages/apex/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/encdec_multihead_attn_func.py
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/encdec_multihead_attn.py to en
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/self_multihead_attn.py to self
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/fast_encdec_multihead_attn_nor
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/self_multihead_attn_func.py to
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/mask_softmax_dropout_func.py t
```


YOLOv5 vanilla.ipynb
File Edit View Insert Runtime Tools Help Last edited on November 1
+ Code + Text
writing list of installed files to '/tmp/pip-record-5c4k7z13/install-record.txt'
Running setup.py install for apex ... done
Removing source in /tmp/pip-req-build-odhan0so
Successfully installed apex-0.1
Cleaning up...
Removed build tracker '/tmp/pip-req-tracker-xucgv7ao'
/content
!git clone https://github.com/ultralytics/yolov5.git
%cd yolov5
!git checkout ec72eea62bf5bb86b0272f2e65e413957533507f #release v3
!pip install -r requirements.txt
#!./weights/download_weights.sh
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 7)) (7.0.0)
Requirement already satisfied: tensorboard in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 8)) (2.3.0)
Collecting PyYAML>=5.3
Downloading https://files.pythonhosted.org/packages/64/c2/b80047c7ac2478f9501676c988a5411ed5572f35d1beff9cae07d321512c/PyYAML-5.3.1.tar.gz (269kB)
276kB 49.2MB/s
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 10)) (0.7.0+cu101)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 11)) (1.4.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 12)) (4.41.1)
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.6/dist-packages (from pycocotools==2.0->-r requirements.txt (line 13)) (50.3.2)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from torch>=1.4->-r requirements.txt (line 5)) (0.16.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6)) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6)) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6)) (0.10.0)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.17.2)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (3.3.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (2.23.0)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (0.10.0)
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (3.12.4)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (0.4
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.15.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.0.1)
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (0
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.33.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.7.0)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard->-r requ
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard->-r requirements.txt (
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard->-r requirements.txt (l
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard->-r requ
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r requirements.txt (line 8
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r requirements.txt (line 8)) (2.
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r requirements.txt (line 8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r req
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard->-r requi
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.6/dist-packages (from rsa<5,>=3.1.4; python_version >= "3"->google-auth<2,>=1.6.3->tensc
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from importlib-metadata; python_version < "3.8"->markdown>=2.6.8->tensorboa
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tens
Building wheels for collected packages: PyYAML, pycocotools
Building wheel for PyYAML (setup.py) ... done
Created wheel for PyYAML: filename=PyYAML-5.3.1-cp36-cp36m-linux_x86_64.whl size=44619 sha256=5415247be737bc38b4afe066014e798b82ec35b7558c2abafaa3bc3b7532cb2
Stored in directory: /root/.cache/pip/wheels/a7/c1/ea/cf5bd31012e735dc1dfea3131a2d5eae7978b251083d6247bd

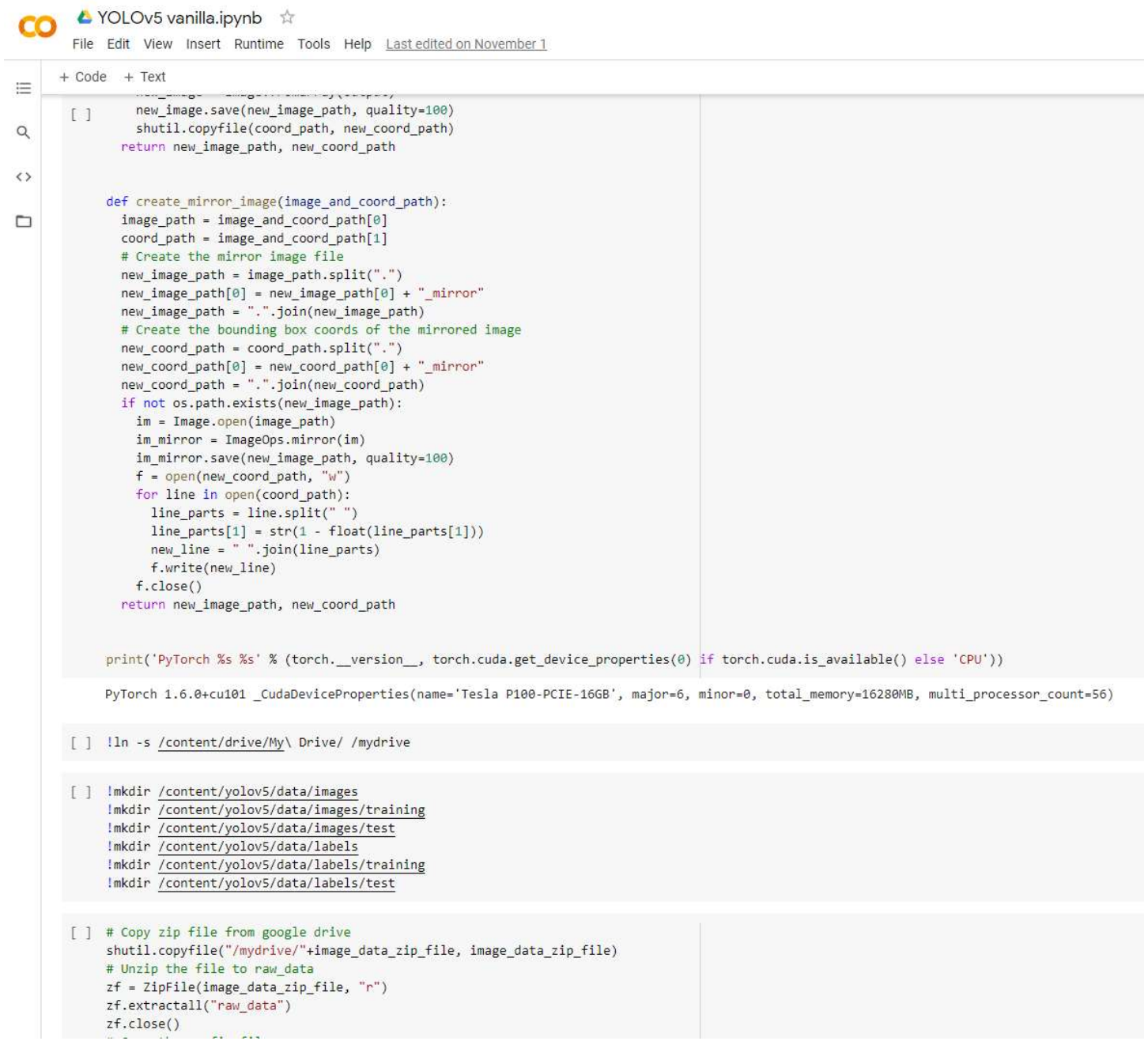
YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

```
[ ]   Found existing installation: pycocotools 2.0.2
        Uninstalling pycocotools-2.0.2:
          Successfully uninstalled pycocotools-2.0.2
    Successfully installed PyYAML-5.3.1 numpy-1.17.0 pycocotools-2.0
```

```
[ ] import torch
    import os
    from PIL import Image, ImageOps
    from glob import glob
    import numpy as np
    import shutil
    from zipfile import ZipFile
    import random

    # What is the zip file's name?
    image_data_zip_file = "250_images.zip"

    # What do you want to refer to the output as?
    model_name = "yolov5_vanilla"

    # What share of the images should be in the training set?
    percent_training = 0.80

    add_mirror_images = False
    add_salt_and_pepper = False
    salt_and_peper_amounts = [0.02, 0.03]
    salt_and_peper_amounts = [0.05]

    # Data augmentation functions
    def create_salt_and_pepper(image_and_coord_path, amount = 0.01):
      image_path = image_and_coord_path[0]
      coord_path = image_and_coord_path[1]
      amount_percent = int(amount * 100)
      new_image_path = image_path.split(".")
      new_image_path[0] = new_image_path[0] + "_salt_and_pepper_" + str(amount_percent)
      new_image_path = ".".join(new_image_path)
      new_coord_path = coord_path.split(".")
      new_coord_path[0] = new_coord_path[0] + "_salt_and_pepper_" + str(amount_percent)
      new_coord_path = ".".join(new_coord_path)
      if not os.path.exists(new_image_path):
        image = Image.open(image_path)
        output = np.copy(np.array(image))
        # add salt
        nb_salt = np.ceil(amount * output.size * 0.5)
        coords = [np.random.randint(0, i - 1, int(nb_salt)) for i in output.shape]
        output[tuple(coords)] = 1 #output[coords] = 1
        # add pepper
        nb_pepper = np.ceil(amount* output.size * 0.5)
        coords = [np.random.randint(0, i - 1, int(nb_pepper)) for i in output.shape]
        output[tuple(coords)] = 0 #output[coords] = 0
        new_image = Image.fromarray(output)
```



YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

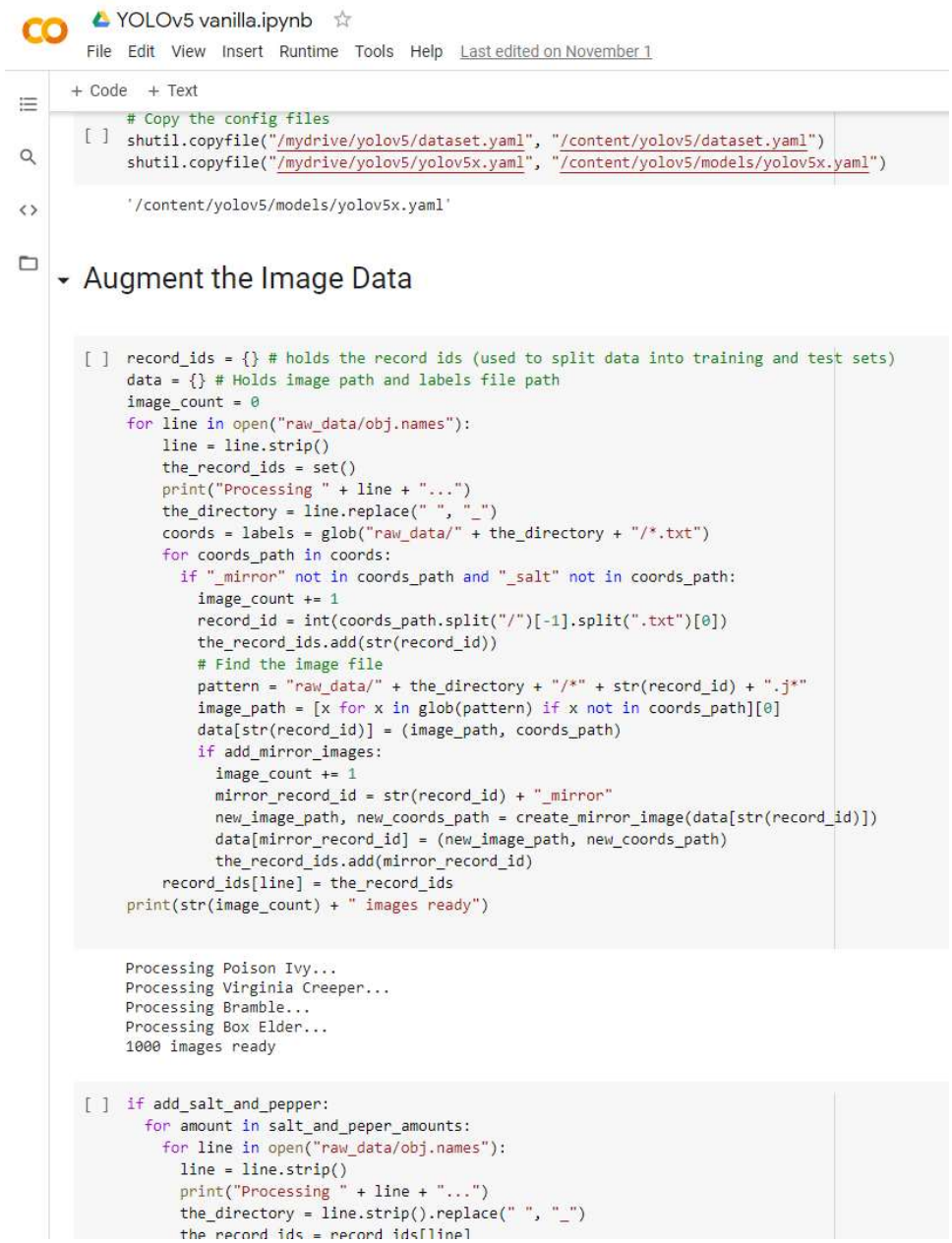
```
[ ]       new_image.save(new_image_path, quality=100)
          shutil.copyfile(coord_path, new_coord_path)
        return new_image_path, new_coord_path


      def create_mirror_image(image_and_coord_path):
        image_path = image_and_coord_path[0]
        coord_path = image_and_coord_path[1]
        # Create the mirror image file
        new_image_path = image_path.split(".")
        new_image_path[0] = new_image_path[0] + "_mirror"
        new_image_path = ".".join(new_image_path)
        # Create the bounding box coords of the mirrored image
        new_coord_path = coord_path.split(".")
        new_coord_path[0] = new_coord_path[0] + "_mirror"
        new_coord_path = ".".join(new_coord_path)
        if not os.path.exists(new_image_path):
          im = Image.open(image_path)
          im_mirror = ImageOps.mirror(im)
          im_mirror.save(new_image_path, quality=100)
          f = open(new_coord_path, "w")
          for line in open(coord_path):
            line_parts = line.split(" ")
            line_parts[1] = str(1 - float(line_parts[1]))
            new_line = " ".join(line_parts)
            f.write(new_line)
          f.close()
        return new_image_path, new_coord_path


      print('PyTorch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

```
PyTorch 1.6.0+cu101 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', major=6, minor=0, total_memory=16280MB, multi_processor_count=56)
```

```
[ ] !ln -s /content/drive/My\ Drive/ /mydrive
```

```
[ ] !mkdir /content/yolov5/data/images
    !mkdir /content/yolov5/data/images/training
    !mkdir /content/yolov5/data/images/test
    !mkdir /content/yolov5/data/labels
    !mkdir /content/yolov5/data/labels/training
    !mkdir /content/yolov5/data/labels/test
```

```
[ ] # Copy zip file from google drive
    shutil.copyfile("/mydrive/"+image_data_zip_file, image_data_zip_file)
    # Unzip the file to raw_data
    zf = ZipFile(image_data_zip_file, "r")
    zf.extractall("raw_data")
    zf.close()
```



YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

```
[ ] # Copy the config files
    shutil.copyfile("/mydrive/yolov5/dataset.yaml", "/content/yolov5/dataset.yaml")
    shutil.copyfile("/mydrive/yolov5/yolov5x.yaml", "/content/yolov5/models/yolov5x.yaml")
```

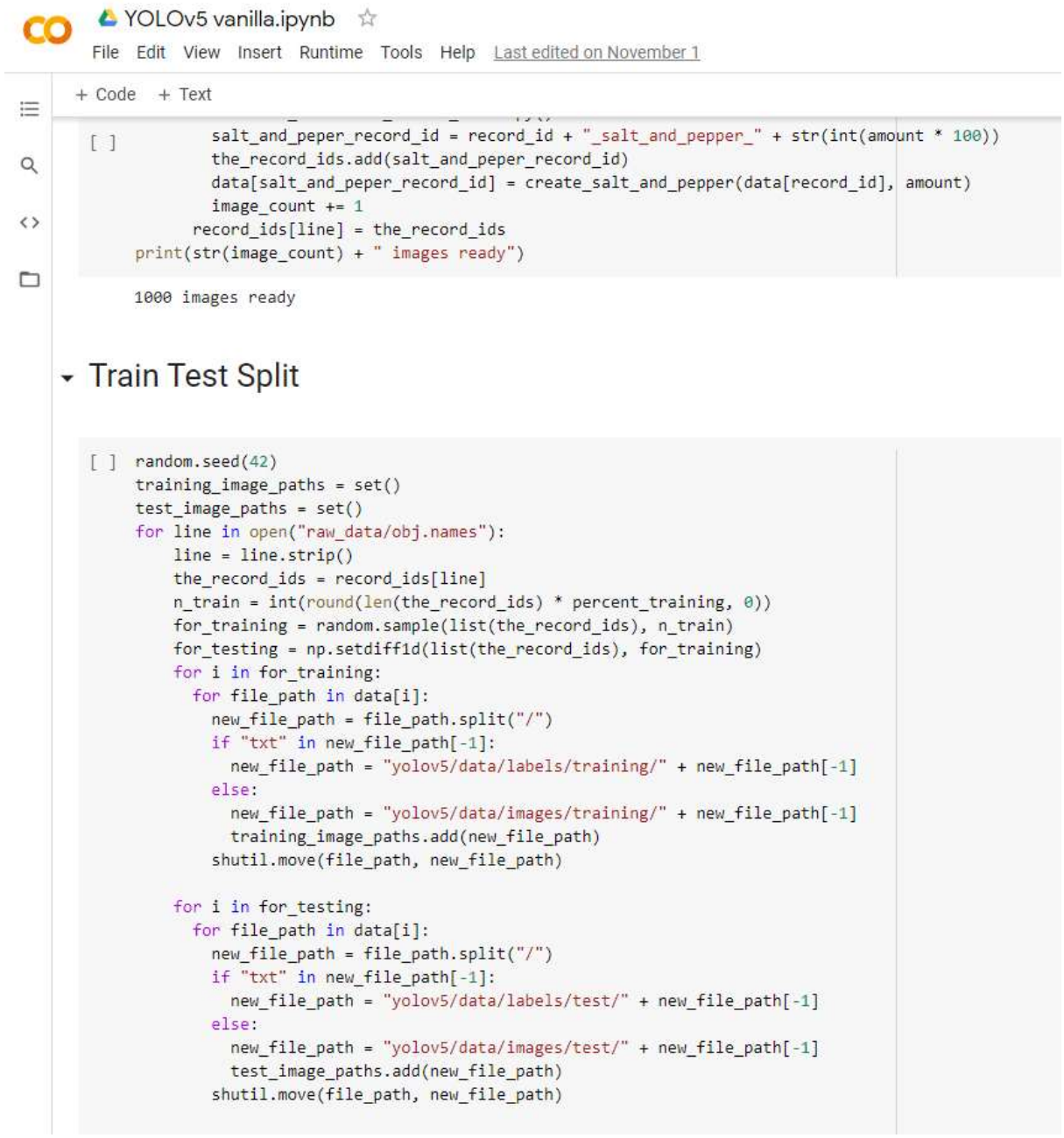
```
'/content/yolov5/models/yolov5x.yaml'
```

Augment the Image Data

```
[ ] record_ids = {} # holds the record ids (used to split data into training and test sets)
    data = {} # Holds image path and labels file path
    image_count = 0
    for line in open("raw_data/obj.names"):
        line = line.strip()
        the_record_ids = set()
        print("Processing " + line + "...")
        the_directory = line.replace(" ", "_")
        coords = labels = glob("raw_data/" + the_directory + "/*.txt")
        for coords_path in coords:
          if "_mirror" not in coords_path and "_salt" not in coords_path:
            image_count += 1
            record_id = int(coords_path.split("/")[-1].split(".txt")[0])
            the_record_ids.add(str(record_id))
            # Find the image file
            pattern = "raw_data/" + the_directory + "/*" + str(record_id) + ".j*"
            image_path = [x for x in glob(pattern) if x not in coords_path][0]
            data[str(record_id)] = (image_path, coords_path)
            if add_mirror_images:
              image_count += 1
              mirror_record_id = str(record_id) + "_mirror"
              new_image_path, new_coords_path = create_mirror_image(data[str(record_id)])
              data[mirror_record_id] = (new_image_path, new_coords_path)
              the_record_ids.add(mirror_record_id)
        record_ids[line] = the_record_ids
    print(str(image_count) + " images ready")
```

```
Processing Poison Ivy...
Processing Virginia Creeper...
Processing Bramble...
Processing Box Elder...
1000 images ready
```

```
[ ] if add_salt_and_pepper:
      for amount in salt_and_peper_amounts:
        for line in open("raw_data/obj.names"):
          line = line.strip()
          print("Processing " + line + "...")
          the_directory = line.strip().replace(" ", "_")
          the_record_ids = record_ids[line]
```



YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

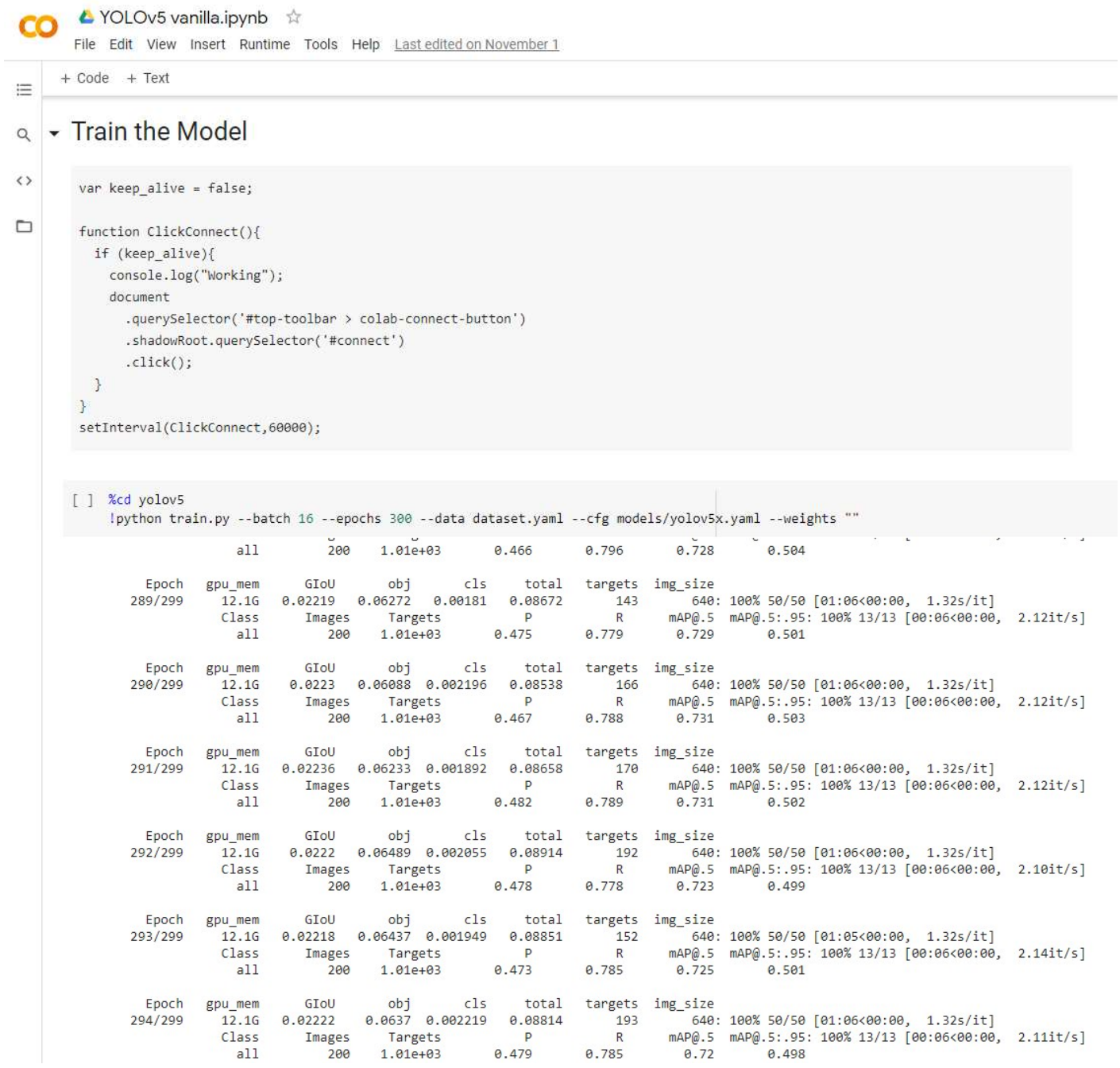
+ Code + Text

```
[ ]         salt_and_peper_record_id = record_id + "_salt_and_pepper_" + str(int(amount * 100))
            the_record_ids.add(salt_and_peper_record_id)
            data[salt_and_peper_record_id] = create_salt_and_pepper(data[record_id], amount)
            image_count += 1
        record_ids[line] = the_record_ids
    print(str(image_count) + " images ready")
```

```
1000 images ready
```

Train Test Split

```
[ ] random.seed(42)
    training_image_paths = set()
    test_image_paths = set()
    for line in open("raw_data/obj.names"):
        line = line.strip()
        the_record_ids = record_ids[line]
        n_train = int(round(len(the_record_ids) * percent_training, 0))
        for_training = random.sample(list(the_record_ids), n_train)
        for_testing = np.setdiff1d(list(the_record_ids), for_training)
        for i in for_training:
          for file_path in data[i]:
            new_file_path = file_path.split("/")
            if "txt" in new_file_path[-1]:
              new_file_path = "yolov5/data/labels/training/" + new_file_path[-1]
            else:
              new_file_path = "yolov5/data/images/training/" + new_file_path[-1]
              training_image_paths.add(new_file_path)
            shutil.move(file_path, new_file_path)

        for i in for_testing:
          for file_path in data[i]:
            new_file_path = file_path.split("/")
            if "txt" in new_file_path[-1]:
              new_file_path = "yolov5/data/labels/test/" + new_file_path[-1]
            else:
              new_file_path = "yolov5/data/images/test/" + new_file_path[-1]
              test_image_paths.add(new_file_path)
            shutil.move(file_path, new_file_path)
```



YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

Train the Model

```
var keep_alive = false;

function ClickConnect(){
  if (keep_alive){
    console.log("Working");
    document
      .querySelector('#top-toolbar > colab-connect-button')
      .shadowRoot.querySelector('#connect')
      .click();
  }
}
setInterval(ClickConnect,60000);
```

```
[ ] %cd yolov5
    !python train.py --batch 16 --epochs 300 --data dataset.yaml --cfg models/yolov5x.yaml --weights ""
```

```
                 all         200    1.01e+03       0.466       0.796       0.728       0.504

      Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
    289/299     12.1G   0.02219   0.06272   0.00181   0.08672       143       640: 100% 50/50 [01:06<00:00,  1.32s/it]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.12it/s]
                 all         200    1.01e+03       0.475       0.779       0.729       0.501

      Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
    290/299     12.1G    0.0223   0.06088  0.002196   0.08538       166       640: 100% 50/50 [01:06<00:00,  1.32s/it]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.12it/s]
                 all         200    1.01e+03       0.467       0.788       0.731       0.503

      Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
    291/299     12.1G   0.02236   0.06233  0.001892   0.08658       170       640: 100% 50/50 [01:06<00:00,  1.32s/it]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.12it/s]
                 all         200    1.01e+03       0.482       0.789       0.731       0.502

      Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
    292/299     12.1G    0.0222   0.06489  0.002055   0.08914       192       640: 100% 50/50 [01:06<00:00,  1.32s/it]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.10it/s]
                 all         200    1.01e+03       0.478       0.778       0.723       0.499

      Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
    293/299     12.1G   0.02218   0.06437  0.001949   0.08851       152       640: 100% 50/50 [01:05<00:00,  1.32s/it]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.14it/s]
                 all         200    1.01e+03       0.473       0.785       0.725       0.501

      Epoch   gpu_mem      GIoU       obj       cls     total   targets  img_size
    294/299     12.1G   0.02222    0.0637  0.002219   0.08814       193       640: 100% 50/50 [01:06<00:00,  1.32s/it]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 13/13 [00:06<00:00,  2.11it/s]
                 all         200    1.01e+03       0.479       0.785        0.72       0.498
```

YOLOv5 vanilla.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

```
[ ] 300 epochs completed in 6.370 hours.
```

```
[ ] !python test.py --data dataset.yaml --weights weights/last.pt --verbose

Namespace(augment=False, batch_size=32, conf_thres=0.001, data='dataset.yaml', device='', img_size=640, iou_thres=0.65, merge=False, save_json=False, single_cls=False, task='val', ve
Using CUDA device0 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', total_memory=16280MB)

Model Summary: 407 layers, 8.84538e+07 parameters, 0 gradients
Fusing layers...
Model Summary: 284 layers, 8.84108e+07 parameters, 8.45317e+07 gradients
Caching labels data/labels/test (200 found, 0 missing, 0 empty, 0 duplicate, for 200 images): 100% 200/200 [00:00<00:00, 1169.16it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0% 0/7 [00:00<?, ?it/s]/content/yolov5/utils/utils.py:580: UserWarning: This overload
       nonzero()
Consider using one of the following signatures instead:
       nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:766.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 7/7 [00:08<00:00,  1.16s/it]
                 all         200    1.01e+03       0.592       0.755       0.731         0.5
          Poison Ivy         200         225       0.564       0.653       0.615       0.413
    Virginia Creeper         200         216       0.667       0.917       0.917       0.702
             Bramble         200         369         0.6       0.756       0.752       0.495
           Box Elder         200         198       0.539       0.692       0.639       0.392
Speed: 23.8/1.0/24.8 ms inference/NMS/total per 640x640 image at batch-size 32
```

```
[ ] !python test.py --data dataset.yaml --weights weights/best.pt --verbose

Namespace(augment=False, batch_size=32, conf_thres=0.001, data='dataset.yaml', device='', img_size=640, iou_thres=0.65, merge=False, save_json=False, single_cls=False, task='val', ve
Using CUDA device0 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', total_memory=16280MB)

Model Summary: 407 layers, 8.84538e+07 parameters, 0 gradients
Fusing layers...
Model Summary: 284 layers, 8.84108e+07 parameters, 8.45317e+07 gradients
Caching labels data/labels/test (200 found, 0 missing, 0 empty, 0 duplicate, for 200 images): 100% 200/200 [00:00<00:00, 10534.35it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0% 0/7 [00:00<?, ?it/s]/content/yolov5/utils/utils.py:580: UserWarning: This overload
       nonzero()
Consider using one of the following signatures instead:
       nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:766.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 7/7 [00:08<00:00,  1.20s/it]
                 all         200    1.01e+03       0.527       0.784       0.733       0.504
          Poison Ivy         200         225        0.47       0.707       0.632       0.414
    Virginia Creeper         200         216       0.666       0.912       0.911       0.703
             Bramble         200         369        0.54        0.77       0.739       0.483
           Box Elder         200         198       0.433       0.747       0.651       0.417
Speed: 23.8/1.0/24.9 ms inference/NMS/total per 640x640 image at batch-size 32
```

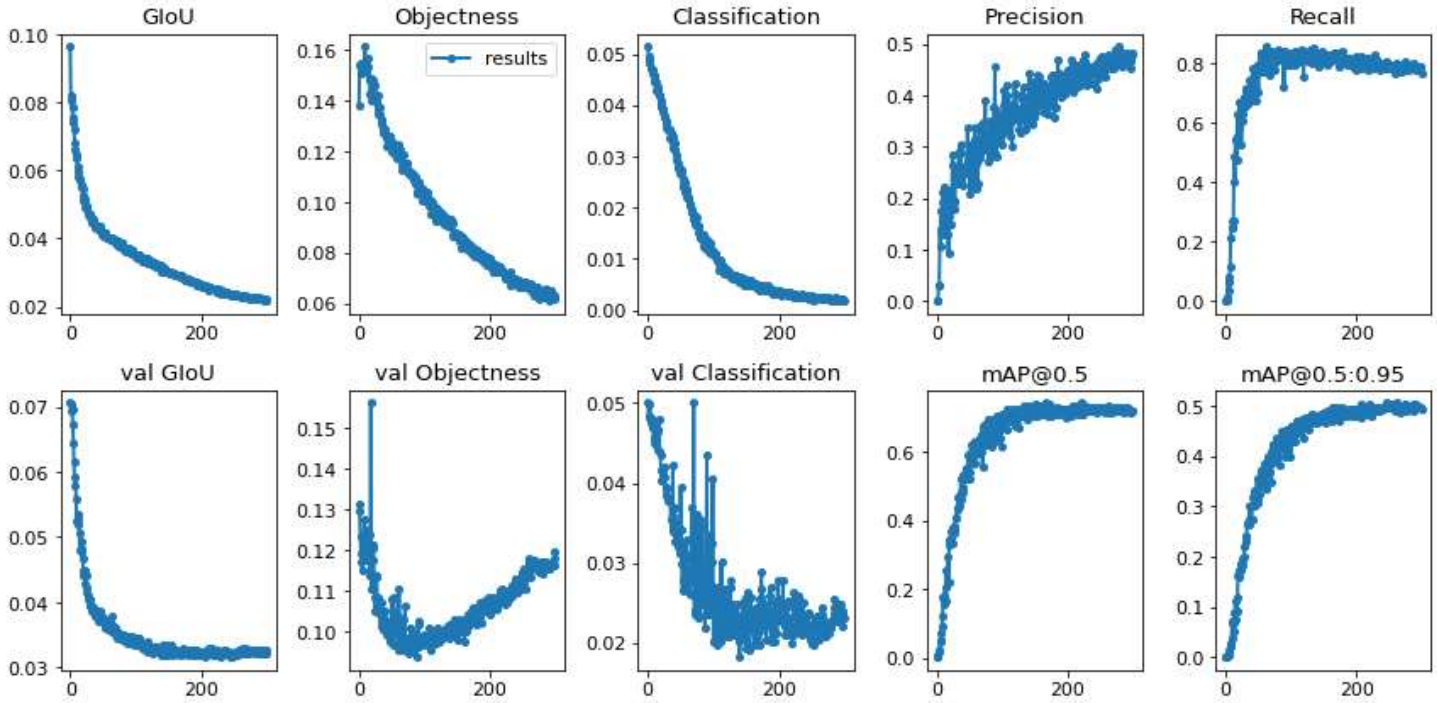
```
[ ] os.mkdir("/mydrive/yolov5/" + model_name)
    shutil.copy("/content/yolov5/results.txt", "/mydrive/yolov5/" + model_name + "/results.txt")
    shutil.copy("/content/yolov5/weights/last.pt", "/mydrive/yolov5/" + model_name + "/last.pt")
    shutil.copy("/content/yolov5/weights/best.pt", "/mydrive/yolov5/" + model_name + "/best.pt")
```

YOLOv5 vanilla.ipynb ☆

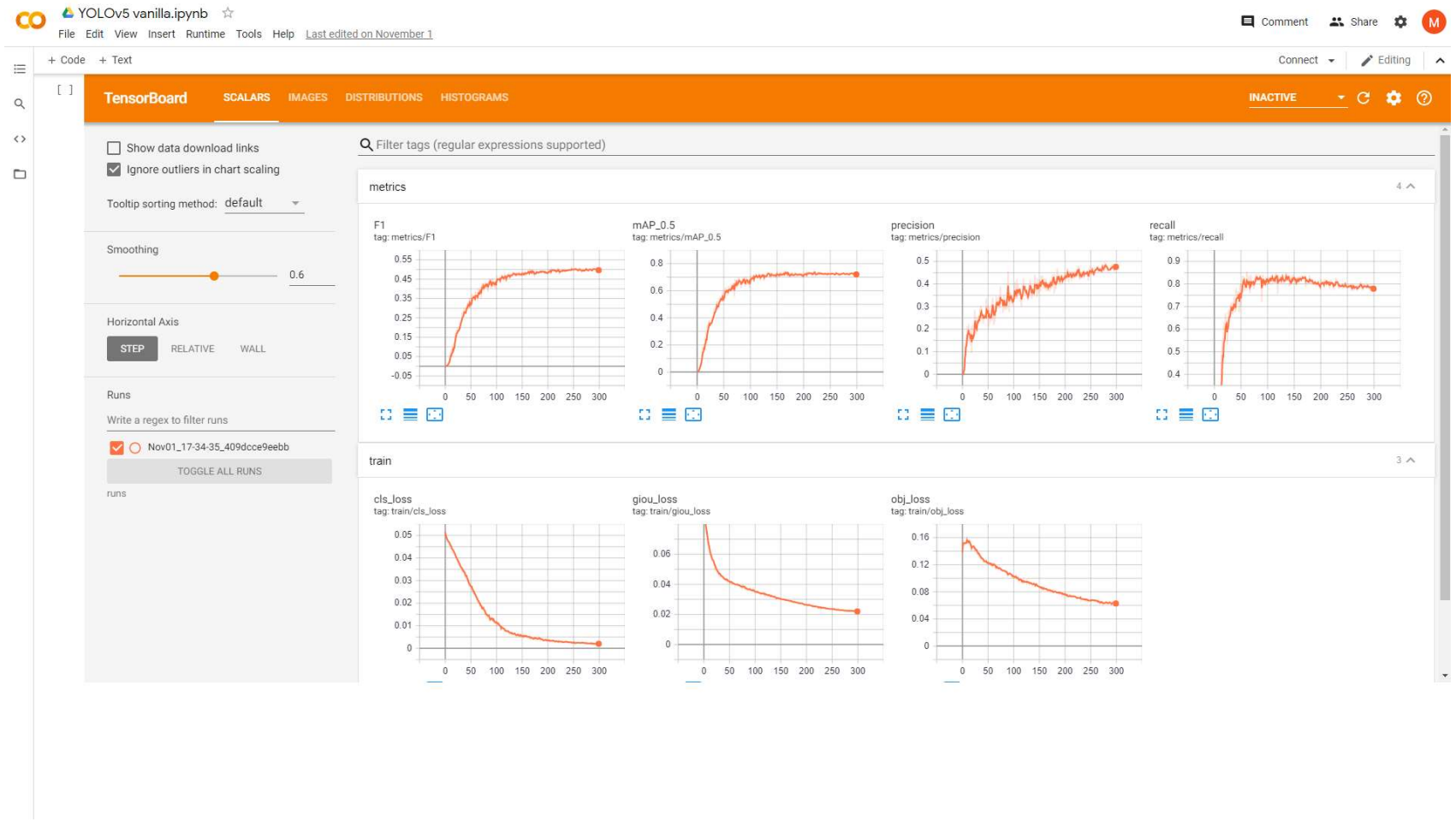
File Edit View Insert Runtime Tools Help Last edited on November 1

+ Code + Text

```
[ ] from utils import utils
    utils.plot_results()
```




```
[ ] %load_ext tensorboard
    %tensorboard --logdir runs
```



Simulated Field Test Notebooks

This last section details the code used to conduct the simulated field tests. Once again, the images from Google Colab are presented. Note: The variants are referred to using code names. Vanilla is the 250-image variant. Double Vision is the 500. Four Eyes is the 1,000 and 2K is the 2,000 image variant.

YOLO v3

YOLOv3 Field Tests.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

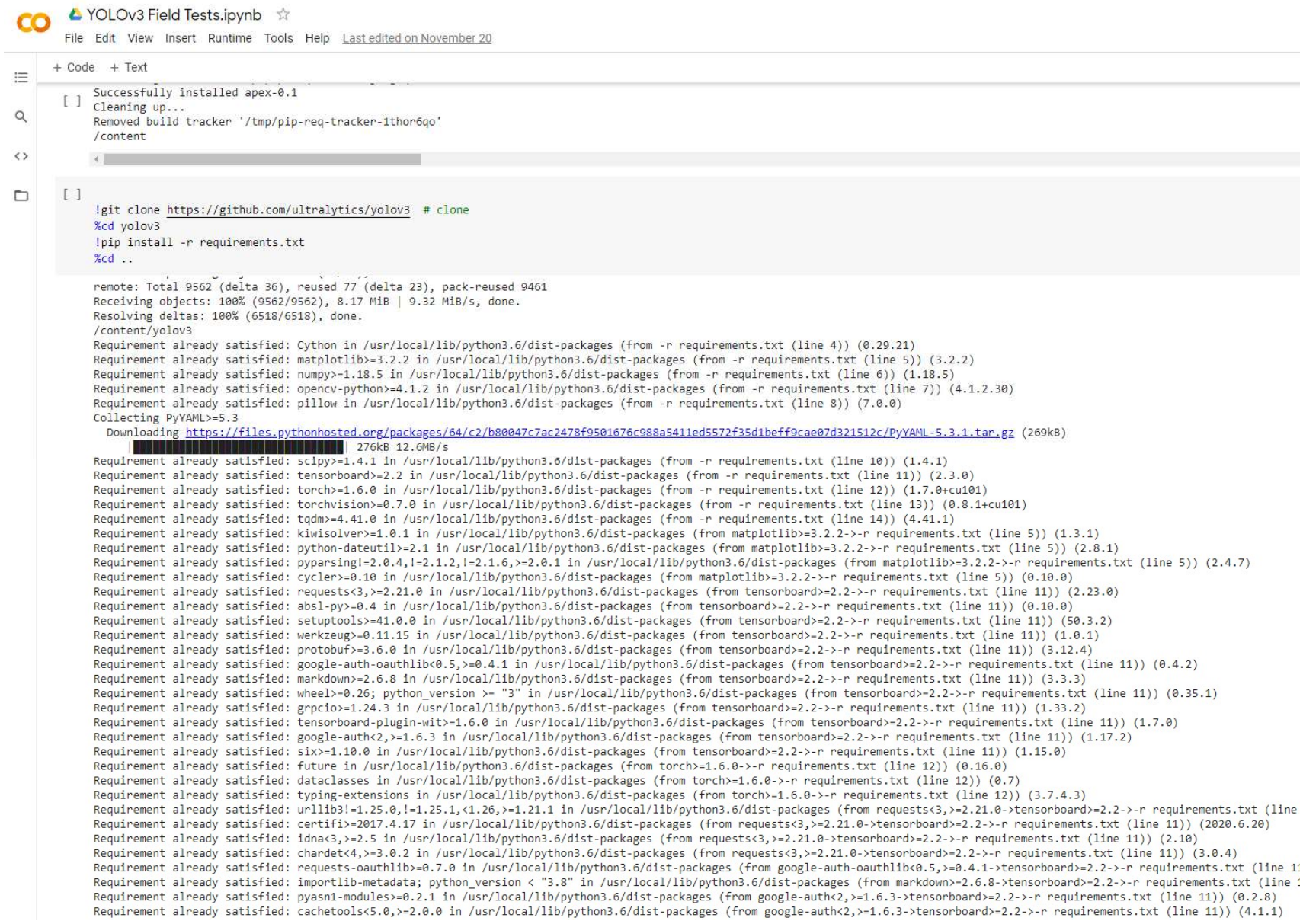
```
[ ] import torch
    import os
    from glob import glob
    import shutil
    from zipfile import ZipFile
    from IPython.display import Image, clear_output

    # What is the zip file's name?
    image_data_zip_file = "field_test.zip"

    print('PyTorch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

```
PyTorch 1.7.0+cu101 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', major=7, minor=0, total_memory=16130MB, multi_processor_count=80)
```

Set up Environment

```
[ ] !git clone https://github.com/NVIDIA/apex
    %cd apex
    !pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext" ./
    %cd /content
```

```
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/xentropy/softmax_xentropy.py to softmax_xentropy.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/sparsity/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/sparsity/asp.py to asp.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/sparsity/sparse_masklib.py to sparse_masklib.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/groupbn/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/groupbn/batch_norm.py to batch_norm.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/self_multihead_attn_func.py to self_multihead_attn_func.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/fast_encdec_multihead_attn_func.py to fast_encdec_multihead_attn_func.cpy
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/self_multihead_attn.py to self_multihead_attn.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/encdec_multihead_attn.py to encdec_multihead_attn.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/fast_self_multihead_attn_norm_add_func.py to fast_self_multihead_attn_no
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/mask_softmax_dropout_func.py to mask_softmax_dropout_func.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/encdec_multihead_attn_func.py to encdec_multihead_attn_func.cpython-36.py
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/fast_self_multihead_attn_func.py to fast_self_multihead_attn_func.cpytho
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/contrib/multihead_attn/fast_encdec_multihead_attn_norm_add_func.py to fast_encdec_multihead_att
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/wrap.py to wrap.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/lists/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/lists/torch_overrides.py to torch_overrides.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/lists/functional_overrides.py to functional_overrides.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/lists/tensor_overrides.py to tensor_overrides.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/scaler.py to scaler.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/amp.py to amp.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/__version__.py to __version__.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/_process_optimizer.py to _process_optimizer.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/frontend.py to frontend.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/_amp_state.py to _amp_state.cpython-36.pyc
byte-compiling /usr/local/lib/python3.6/dist-packages/apex/amp/rnn_compat.py to rnn_compat.cpython-36.pyc
```


YOLOv3 Field Tests.ipynb
File Edit View Insert Runtime Tools Help Last edited on November 20
+ Code + Text
Successfully installed apex-0.1
Cleaning up...
Removed build tracker '/tmp/pip-req-tracker-1thor6qo'
/content
!git clone https://github.com/ultralytics/yolov3 # clone
%cd yolov3
!pip install -r requirements.txt
%cd ..
remote: Total 9562 (delta 36), reused 77 (delta 23), pack-reused 9461
Receiving objects: 100% (9562/9562), 8.17 MiB | 9.32 MiB/s, done.
Resolving deltas: 100% (6518/6518), done.
/content/yolov3
Requirement already satisfied: Cython in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 4)) (0.29.21)
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 5)) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 6)) (1.18.5)
Requirement already satisfied: opencv-python>=4.1.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 7)) (4.1.2.30)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 8)) (7.0.0)
Collecting PyYAML>=5.3
Downloading https://files.pythonhosted.org/packages/64/c2/b80047c7ac2478f9501676c988a5411ed5572f35d1beff9cae07d321512c/PyYAML-5.3.1.tar.gz (269kB)
| 276kB 12.6MB/s
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 10)) (1.4.1)
Requirement already satisfied: tensorboard>=2.2 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 11)) (2.3.0)
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 12)) (1.7.0+cu101)
Requirement already satisfied: torchvision>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 13)) (0.8.1+cu101)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 14)) (4.41.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (1.3.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (0.10.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (2.23.0)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (0.10.0)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (50.3.2)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.0.1)
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (3.12.4)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (0.4.2)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (3.3.3)
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (0.35.1)
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.33.2)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.7.0)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.17.2)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard>=2.2->-r requirements.txt (line 11)) (1.15.0)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from torch>=1.6.0->-r requirements.txt (line 12)) (0.16.0)
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-packages (from torch>=1.6.0->-r requirements.txt (line 12)) (0.7)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.6/dist-packages (from torch>=1.6.0->-r requirements.txt (line 12)) (3.7.4.3)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (line
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (line 11)) (2020.6.20)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (line 11)) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard>=2.2->-r requirements.txt (line 11)) (3.0.4)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard>=2.2->-r requirements.txt (line 1
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard>=2.2->-r requirements.txt (line
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard>=2.2->-r requirements.txt (line 11)) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard>=2.2->-r requirements.txt (line 11)) (4.1.1)

YOLOv3 Field Tests.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

```
  Stored in directory: /root/.cache/pip/wheels/a7/c1/ea/cf5bd31012e735dc1dfea3131a2d5eae7978b251083d624
Successfully built PyYAML
Installing collected packages: PyYAML
  Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed PyYAML-5.3.1
/content
```

```
!ln -s /content/drive/My\ Drive/ /mydrive
```

Get Field Test Data

```
# Copy zip file from google drive
shutil.copyfile("/mydrive/"+image_data_zip_file, image_data_zip_file)
# Unzip the file to raw_data
zf = ZipFile(image_data_zip_file, "r")
zf.extractall("raw_data")
zf.close()
```

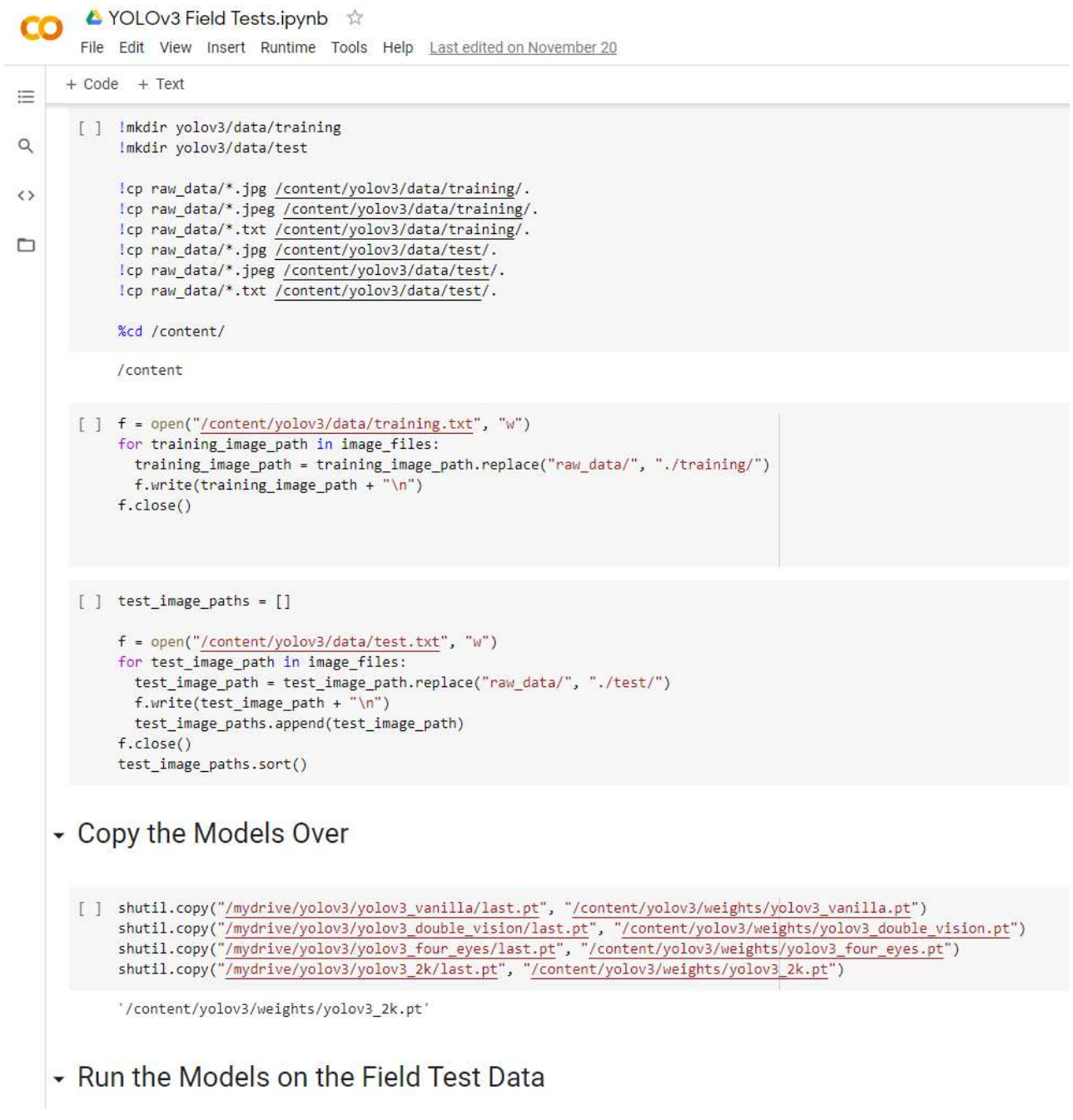
```
image_files = [f for f in glob("raw_data/*.jpg")] + [f for f in glob("raw_data/*.jpeg")]
image_files.sort(key=str.lower)
print(len(image_files))
```

```
100
```

Set Up The Config Files

```
# Copy the config files
shutil.copyfile("/mydrive/yolov3/test.cfg", "/content/yolov3/cfg/test.cfg")
# Copy the class labels
shutil.copyfile("raw_data/obj.names", "/content/yolov3/data/obj.names")

n_classes = 0
for line in open("raw_data/obj.names"):
  n_classes += 1

f = open("/content/yolov3/data/obj.data", "w")
f.write("classes=" + str(n_classes) + "\n")
f.write("train=data/training.txt\n")
f.write("valid=data/test.txt\n")
f.write("names=data/obj.names")
f.close()
```

```
!mkdir yolov3/data/training
```



YOLOv3 Field Tests.ipynb

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

```
[ ] !mkdir yolov3/data/training
    !mkdir yolov3/data/test

    !cp raw_data/*.jpg /content/yolov3/data/training/.
    !cp raw_data/*.jpeg /content/yolov3/data/training/.
    !cp raw_data/*.txt /content/yolov3/data/training/.
    !cp raw_data/*.jpg /content/yolov3/data/test/.
    !cp raw_data/*.jpeg /content/yolov3/data/test/.
    !cp raw_data/*.txt /content/yolov3/data/test/.

    %cd /content/
```

```
/content
```

```
[ ] f = open("/content/yolov3/data/training.txt", "w")
    for training_image_path in image_files:
      training_image_path = training_image_path.replace("raw_data/", "./training/")
      f.write(training_image_path + "\n")
    f.close()
```

```
[ ] test_image_paths = []

    f = open("/content/yolov3/data/test.txt", "w")
    for test_image_path in image_files:
      test_image_path = test_image_path.replace("raw_data/", "./test/")
      f.write(test_image_path + "\n")
      test_image_paths.append(test_image_path)
    f.close()
    test_image_paths.sort()
```

Copy the Models Over

```
[ ] shutil.copy("/mydrive/yolov3/yolov3_vanilla/last.pt", "/content/yolov3/weights/yolov3_vanilla.pt")
    shutil.copy("/mydrive/yolov3/yolov3_double_vision/last.pt", "/content/yolov3/weights/yolov3_double_vision.pt")
    shutil.copy("/mydrive/yolov3/yolov3_four_eyes/last.pt", "/content/yolov3/weights/yolov3_four_eyes.pt")
    shutil.copy("/mydrive/yolov3/yolov3_2k/last.pt", "/content/yolov3/weights/yolov3_2k.pt")
```

```
'/content/yolov3/weights/yolov3_2k.pt'
```

Run the Models on the Field Test Data

YOLOv3_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

```
[ ] %cd /content/yolov3
```

```
/content/yolov3
```

Vanilla

```
[ ] !python3 test.py --cfg test.cfg --data data/obj.data --weights weights/yolov3_vanilla.pt
```

```
Namespace(augment=False, batch_size=16, cfg='./cfg/test.cfg', conf_thres=0.001, data='data/obj.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25627e+07 parameters, 6.25627e+07 gradients
Reading image shapes: 100% 100/100 [00:00<00:00, 3394.52it/s]
Caching labels data/test.txt (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 7111.76it/s]
               Class    Images   Targets         P         R   mAP@0.5        F1:   0% 0/7 [00:00<?, ?it/s]/content/yolov3/utils/utils.py:512: UserWarning: This overload
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class    Images   Targets         P         R   mAP@0.5        F1: 100% 7/7 [00:02<00:00,  3.13it/s]
                 all       100       302     0.695     0.746     0.696     0.702
         Poison Ivy       100        61     0.553     0.836     0.721     0.665
   Virginia Creeper       100        61     0.903     0.656     0.703      0.76
            Bramble       100       108     0.757     0.676     0.705     0.714
           Box Elder       100        72     0.566     0.816     0.655     0.669
Speed: 5.7/2.1/7.7 ms inference/NMS/total per 512x512 image at batch-size 16
```

```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --cfg test.cfg --weights weights/yolov3_vanilla.pt --source data/test/{image_file} --names data/obj.names
      display(Image(filename='output/'+image_file, width=600))
      print("\n\n")
```

```
Namespace(agnostic_nms=False, augment=False, cfg='./cfg/test.cfg', classes=None, conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512, iou_thres=0.6, names
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
image 1/1 data/test/virginiacreeper_00113509.jpg: 512x384 1 Virginia Creepers, Done. (0.012s)
Results saved to /content/yolov3/output
Done. (0.071s)
```

YOLOv3_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

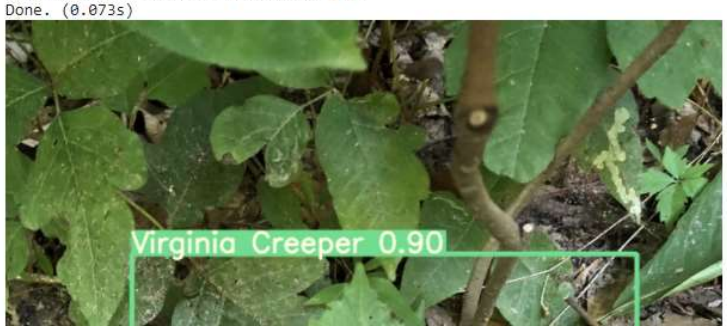
Double Vision

```
[ ] !python3 test.py --cfg test.cfg --data data/obj.data --weights weights/yolov3_double_vision.pt
```

```
Namespace(augment=False, batch_size=16, cfg='./cfg/test.cfg', conf_thres=0.001, data='data/obj.data', device='', img_size=512, iou_thres=0.6, save_j
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25627e+07 parameters, 6.25627e+07 gradients
Caching labels data/test.txt (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 9497.76it/s]
               Class    Images   Targets         P         R   mAP@0.5        F1:   0% 0/7 [00:00<?, ?it/s]/content/yolov3/utils/utils.py:512: UserW
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class    Images   Targets         P         R   mAP@0.5        F1: 100% 7/7 [00:02<00:00,  3.34it/s]
                 all       100       302     0.723     0.742     0.694     0.717
          Poison Ivy       100        61     0.599     0.834     0.753     0.697
    Virginia Creeper       100        61     0.904     0.621     0.655     0.736
             Bramble       100       108      0.82      0.75     0.717     0.783
           Box Elder       100        72     0.569     0.764     0.653     0.652
Speed: 5.6/2.2/7.7 ms inference/NMS/total per 512x512 image at batch-size 16
```

```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --cfg test.cfg --weights weights/yolov3_double_vision.pt --source data/test/{image_file} --names data/obj.names
      display(Image(filename='output/'+image_file, width=600))
      print("\n\n")
```

```
Namespace(agnostic_nms=False, augment=False, cfg='./cfg/test.cfg', classes=None, conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512,
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
image 1/1 data/test/virginiacreeper_00113509.jpg: 512x384 1 Virginia Creepers, Done. (0.012s)
Results saved to /content/yolov3/output
Done. (0.073s)
```



YOLOv3_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

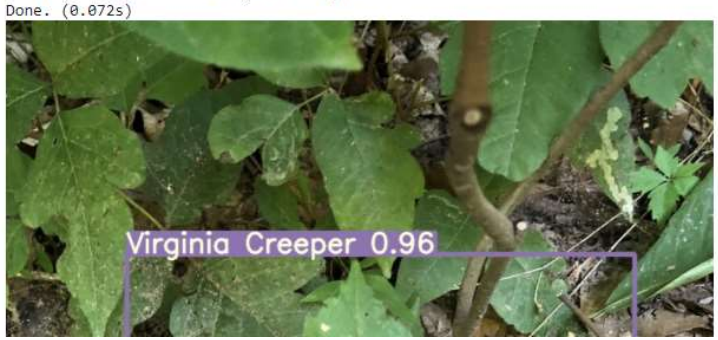
Four Eyes

```
[ ] !python3 test.py --cfg test.cfg --data data/obj.data --weights weights/yolov3_four_eyes.pt
```

```
Namespace(augment=False, batch_size=16, cfg='./cfg/test.cfg', conf_thres=0.001, data='data/obj.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls=False
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25627e+07 parameters, 6.25627e+07 gradients
Caching labels data/test.txt (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 8777.82it/s]
               Class    Images   Targets         P         R   mAP@0.5        F1:   0% 0/7 [00:00<?, ?it/s]/content/yolov3/utils/utils.py:512: UserWarning: This overload of no
       nonzero()
Consider using one of the following signatures instead:
       nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class    Images   Targets         P         R   mAP@0.5        F1: 100% 7/7 [00:02<00:00,  3.41it/s]
                 all       100       302     0.745     0.725     0.708     0.726
          Poison Ivy       100        61     0.627     0.754     0.706     0.685
    Virginia Creeper       100        61     0.888     0.652       0.7     0.752
             Bramble       100       108     0.849      0.73     0.723     0.785
           Box Elder       100        72     0.615     0.764     0.705     0.681
Speed: 5.6/2.3/7.9 ms inference/NMS/total per 512x512 image at batch-size 16
```

```
for test_image_path in image_files:
  image_file = test_image_path.split("/")[-1]
  !echo {image_file}
  !python3 detect.py --cfg test.cfg --weights weights/yolov3_four_eyes.pt --source  data/test/{image_file} --names data/obj.names
  display(Image(filename='output/'+image_file, width=600))
  print("\n\n")
```

```
Namespace(agnostic_nms=False, augment=False, cfg='./cfg/test.cfg', classes=None, conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512, iou_thres=0.6, names='data
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
image 1/1 data/test/virginiacreeper_00113509.jpg: 512x384 1 Virginia Creepers, Done. (0.013s)
Results saved to /content/yolov3/output
Done. (0.072s)
```



YOLOv3_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

2k

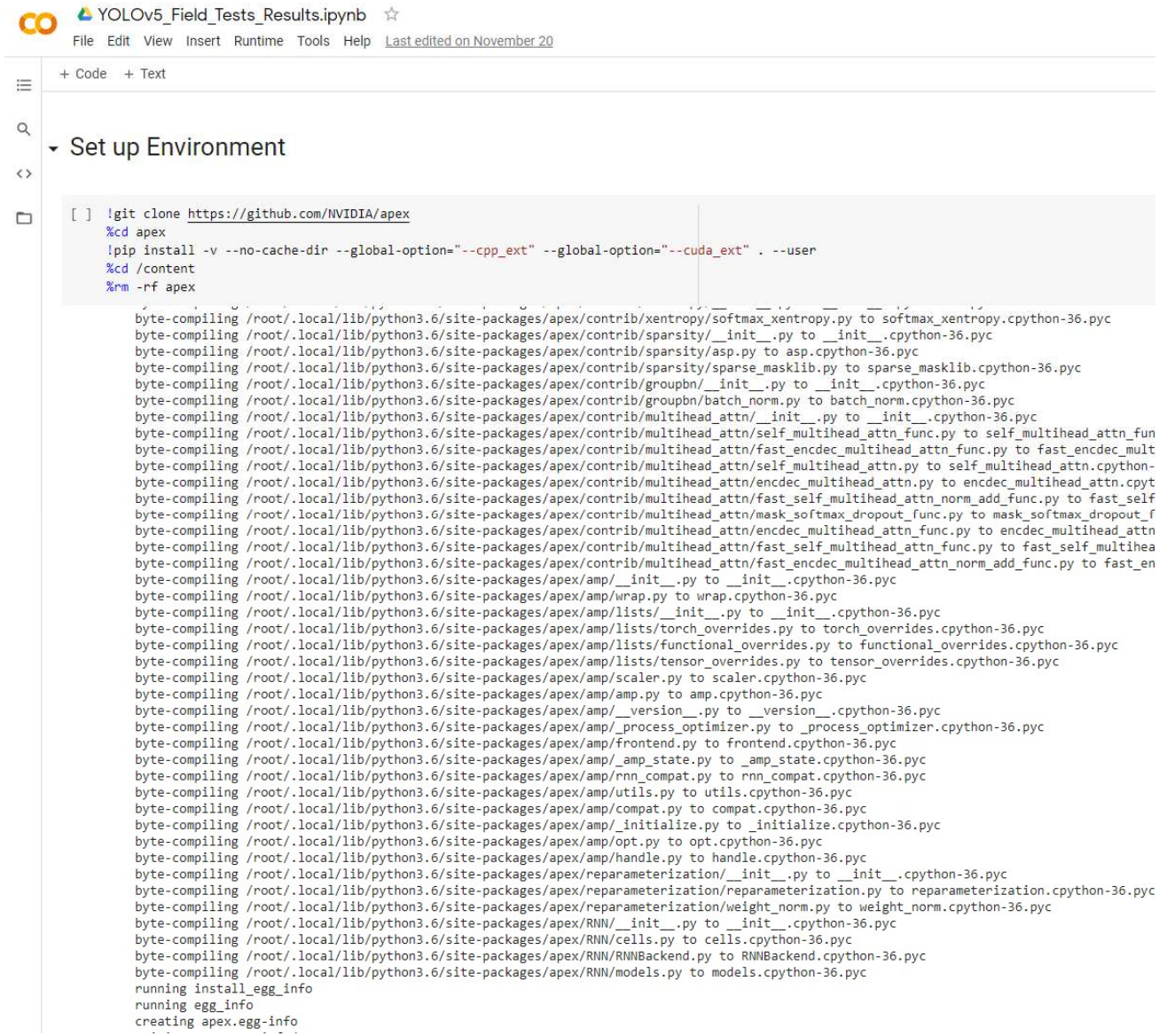
```
[ ] !python3 test.py --cfg test.cfg --data data/obj.data --weights weights/yolov3_2k.pt
```

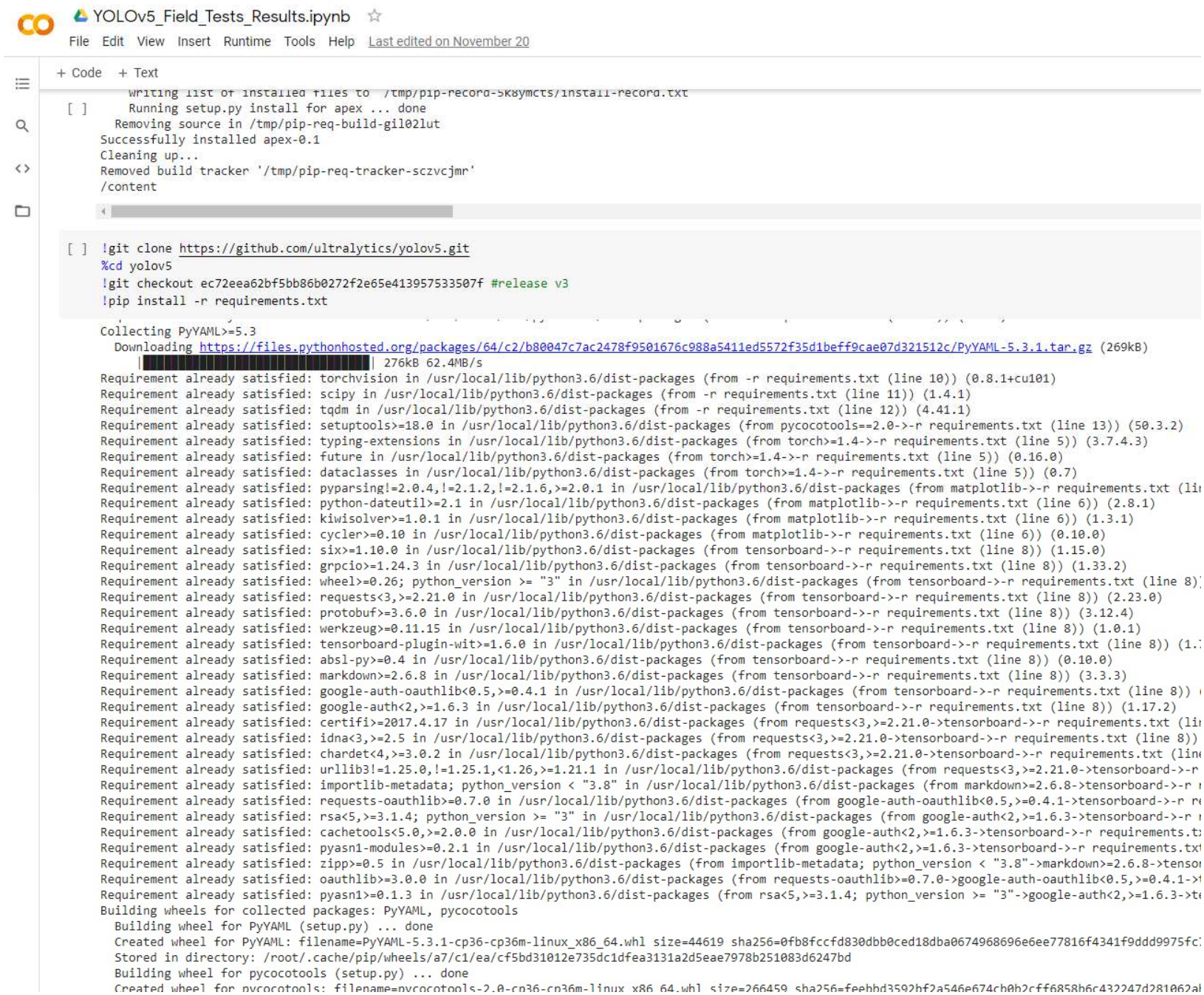
```
Namespace(augment=False, batch_size=16, cfg='./cfg/test.cfg', conf_thres=0.001, data='data/obj.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls=False
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25627e+07 parameters, 6.25627e+07 gradients
Caching labels data/test.txt (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 9020.78it/s]
               Class    Images   Targets         P         R   mAP@0.5        F1:   0% 0/7 [00:00<?, ?it/s]/content/yolov3/utils/utils.py:512: UserWarning: This overload of no
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class    Images   Targets         P         R   mAP@0.5        F1: 100% 7/7 [00:02<00:00,  3.37it/s]
                 all       100       302     0.738     0.708     0.686     0.713
          Poison Ivy       100        61     0.615     0.733     0.687     0.669
    Virginia Creeper       100        61     0.879     0.595     0.617      0.71
             Bramble       100       108     0.794     0.713     0.709     0.752
           Box Elder       100        72     0.663     0.792     0.732     0.722
Speed: 5.6/1.9/7.5 ms inference/NMS/total per 512x512 image at batch-size 16
```

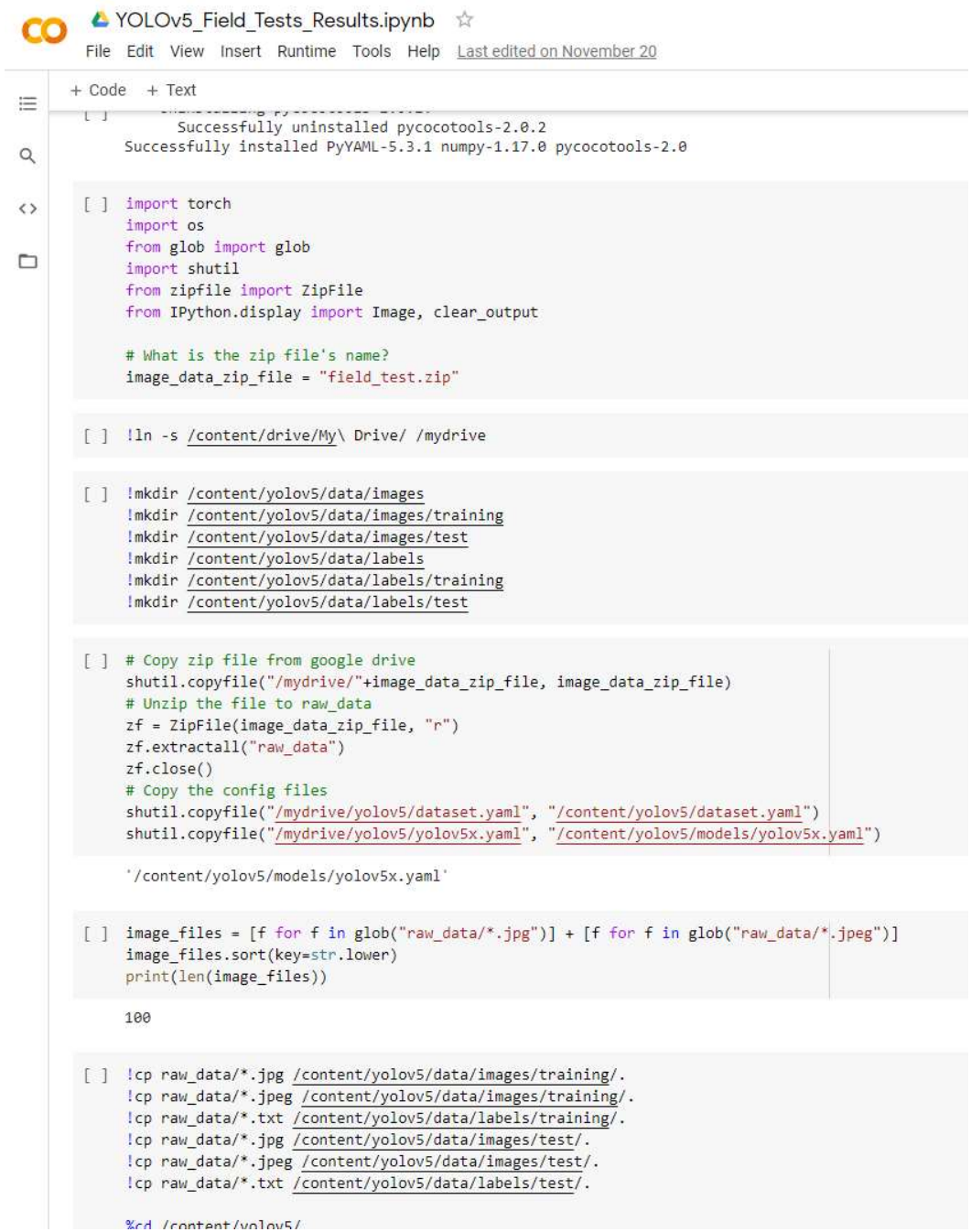
```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --cfg test.cfg --weights weights/yolov3_2k.pt --source data/test/{image_file} --names data/obj.names
      display(Image(filename='output/'+image_file, width=600))
      print("\n\n")
```

```
Namespace(agnostic_nms=False, augment=False, cfg='./cfg/test.cfg', classes=None, conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512, iou_thres=0.6, names='data
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 225 layers, 6.25895e+07 parameters, 6.25895e+07 gradients
image 1/1 data/test/virginiacreeper_00113509.jpg: 512x384 1 Virginia Creepers, Done. (0.012s)
Results saved to /content/yolov3/output
Done. (0.071s)
```

YOLO v5


YOLOv5_Field_Tests_Results.ipynb
File Edit View Insert Runtime Tools Help Last edited on November 20
+ Code + Text
Set up Environment
[ ] !git clone https://github.com/NVIDIA/apex
%cd apex
!pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--cuda_ext" . --user
%cd /content
%rm -rf apex
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/xentropy/softmax_xentropy.py to softmax_xentropy.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/sparsity/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/sparsity/asp.py to asp.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/sparsity/sparse_masklib.py to sparse_masklib.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/groupbn/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/groupbn/batch_norm.py to batch_norm.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/self_multihead_attn_func.py to self_multihead_attn_fun
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/fast_encdec_multihead_attn_func.py to fast_encdec_mult
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/self_multihead_attn.py to self_multihead_attn.cpython-
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/encdec_multihead_attn.py to encdec_multihead_attn.cpyt
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/fast_self_multihead_attn_norm_add_func.py to fast_self
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/mask_softmax_dropout_func.py to mask_softmax_dropout_f
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/encdec_multihead_attn_func.py to encdec_multihead_attn
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/fast_self_multihead_attn_func.py to fast_self_multihea
byte-compiling /root/.local/lib/python3.6/site-packages/apex/contrib/multihead_attn/fast_encdec_multihead_attn_norm_add_func.py to fast_en
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/wrap.py to wrap.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/lists/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/lists/torch_overrides.py to torch_overrides.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/lists/functional_overrides.py to functional_overrides.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/lists/tensor_overrides.py to tensor_overrides.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/scaler.py to scaler.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/amp.py to amp.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/__version__.py to __version__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/_process_optimizer.py to _process_optimizer.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/frontend.py to frontend.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/_amp_state.py to _amp_state.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/rnn_compat.py to rnn_compat.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/utils.py to utils.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/compat.py to compat.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/_initialize.py to _initialize.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/opt.py to opt.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/amp/handle.py to handle.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/reparameterization/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/reparameterization/reparameterization.py to reparameterization.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/reparameterization/weight_norm.py to weight_norm.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/RNN/__init__.py to __init__.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/RNN/cells.py to cells.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/RNN/RNNBackend.py to RNNBackend.cpython-36.pyc
byte-compiling /root/.local/lib/python3.6/site-packages/apex/RNN/models.py to models.cpython-36.pyc
running install_egg_info
running egg_info
creating apex.egg-info


YOLOv5_Field_Tests_Results.ipynb
File Edit View Insert Runtime Tools Help Last edited on November 20
+ Code + Text
Writing list of installed files to '/tmp/pip-record-5k8ymcts/install-record.txt
Running setup.py install for apex ... done
Removing source in /tmp/pip-req-build-gil02lut
Successfully installed apex-0.1
Cleaning up...
Removed build tracker '/tmp/pip-req-tracker-sczvcjmr'
/content
!git clone https://github.com/ultralytics/yolov5.git
%cd yolov5
!git checkout ec72eea62bf5bb86b0272f2e65e413957533507f #release v3
!pip install -r requirements.txt
Collecting PyYAML>=5.3
Downloading https://files.pythonhosted.org/packages/64/c2/b80047c7ac2478f9501676c988a5411ed5572f35d1beff9cae07d321512c/PyYAML-5.3.1.tar.gz (269kB)
276kB 62.4MB/s
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 10)) (0.8.1+cu101)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 11)) (1.4.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from -r requirements.txt (line 12)) (4.41.1)
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.6/dist-packages (from pycocotools==2.0->-r requirements.txt (line 13)) (50.3.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.6/dist-packages (from torch>=1.4->-r requirements.txt (line 5)) (3.7.4.3)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from torch>=1.4->-r requirements.txt (line 5)) (0.16.0)
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-packages (from torch>=1.4->-r requirements.txt (line 5)) (0.7)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (li
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6)) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6)) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->-r requirements.txt (line 6)) (0.10.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.15.0)
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.33.2)
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (2.23.0)
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (3.12.4)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.0.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (0.10.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (3.3.3)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8))
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard->-r requirements.txt (line 8)) (1.17.2)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r requirements.txt (li
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r requirements.txt (line 8))
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r requirements.txt (line
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard->-r
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/dist-packages (from markdown>=2.6.8->tensorboard->-r
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard->-r re
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard->-r
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard->-r requirements.t
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->tensorboard->-r requirements.tx
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from importlib-metadata; python_version < "3.8"->markdown>=2.6.8->tenso
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.6/dist-packages (from rsa<5,>=3.1.4; python_version >= "3"->google-auth<2,>=1.6.3->te
Building wheels for collected packages: PyYAML, pycocotools
Building wheel for PyYAML (setup.py) ... done
Created wheel for PyYAML: filename=PyYAML-5.3.1-cp36-cp36m-linux_x86_64.whl size=44619 sha256=0fb8fccfd830dbb0ced18dba0674968696e6ee77816f4341f9ddd9975fc
Stored in directory: /root/.cache/pip/wheels/a7/c1/ea/cf5bd31012e735dc1dfea3131a2d5eae7978b251083d6247bd
Building wheel for pycocotools (setup.py) ... done
Created wheel for pycocotools: filename=pycocotools-2.0-cp36-cp36m-linux_x86_64.whl size=266459 sha256=feebbd3592bf2a546e674cb0b2cff6858b6c432247d281062a



YOLOv5_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

```
      Successfully uninstalled pycocotools-2.0.2
Successfully installed PyYAML-5.3.1 numpy-1.17.0 pycocotools-2.0
```

```
[ ] import torch
    import os
    from glob import glob
    import shutil
    from zipfile import ZipFile
    from IPython.display import Image, clear_output

    # What is the zip file's name?
    image_data_zip_file = "field_test.zip"
```

```
[ ] !ln -s /content/drive/My\ Drive/ /mydrive
```

```
[ ] !mkdir /content/yolov5/data/images
    !mkdir /content/yolov5/data/images/training
    !mkdir /content/yolov5/data/images/test
    !mkdir /content/yolov5/data/labels
    !mkdir /content/yolov5/data/labels/training
    !mkdir /content/yolov5/data/labels/test
```

```
[ ] # Copy zip file from google drive
    shutil.copyfile("/mydrive/"+image_data_zip_file, image_data_zip_file)
    # Unzip the file to raw_data
    zf = ZipFile(image_data_zip_file, "r")
    zf.extractall("raw_data")
    zf.close()
    # Copy the config files
    shutil.copyfile("/mydrive/yolov5/dataset.yaml", "/content/yolov5/dataset.yaml")
    shutil.copyfile("/mydrive/yolov5/yolov5x.yaml", "/content/yolov5/models/yolov5x.yaml")
```

```
'/content/yolov5/models/yolov5x.yaml'
```

```
[ ] image_files = [f for f in glob("raw_data/*.jpg")] + [f for f in glob("raw_data/*.jpeg")]
    image_files.sort(key=str.lower)
    print(len(image_files))
```

```
100
```

```
[ ] !cp raw_data/*.jpg /content/yolov5/data/images/training/.
    !cp raw_data/*.jpeg /content/yolov5/data/images/training/.
    !cp raw_data/*.txt /content/yolov5/data/labels/training/.
    !cp raw_data/*.jpg /content/yolov5/data/images/test/.
    !cp raw_data/*.jpeg /content/yolov5/data/images/test/.
    !cp raw_data/*.txt /content/yolov5/data/labels/test/.

    %cd /content/yolov5/
```

YOLOv5_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

```
[ ] %cd /content/yolov5/
```

```
/content/yolov5
```

Copy Over Models

```
[ ] shutil.copy("/mydrive/yolov5/yolov5_vanilla/last.pt", "/content/yolov5/weights/yolov5_vanilla.pt")
    shutil.copy("/mydrive/yolov5/yolov5_double_vision/last.pt", "/content/yolov5/weights/yolov5_double_vision.pt")
    shutil.copy("/mydrive/yolov5/yolov5_four_eyes/last.pt", "/content/yolov5/weights/yolov5_four_eyes.pt")
    shutil.copy("/mydrive/yolov5/yolov5_2k/last.pt", "/content/yolov5/weights/yolov5_2k.pt")
```

```
'/content/yolov5/weights/yolov5_2k.pt'
```

Run the Models on the Field Test Data

```
var keep_alive = false;

function ClickConnect(){
  if (keep_alive){
    console.log("Working");
    document
      .querySelector('#top-toolbar > colab-connect-button')
      .shadowRoot.querySelector('#connect')
      .click();
  }
}
setInterval(ClickConnect,60000);
```

Vanilla

```
[ ] !python test.py --data dataset.yaml --weights weights/yolov5_vanilla.pt --verbose
```

```
Namespace(augment=False, batch_size=32, conf_thres=0.001, data='dataset.yaml', device='', img_size=640, iou_thres=0.65, merge=Fal
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 407 layers, 8.84538e+07 parameters, 0 gradients
Fusing layers...
Model Summary: 284 layers, 8.84108e+07 parameters, 8.45317e+07 gradients
Reading image shapes: 100% 100/100 [00:00<00:00, 2965.18it/s]
Caching labels data/labels/test (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 11328.61
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0% 0/4 [00:00<?, ?it/s]/content/y
```

YOLOv5_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

```
[ ] Consider using one of the following signatures instead:
          nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
      i, j = (x[:, 5:] > conf_thres).nonzero().t()
                 Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:03<00:00,  1.07it/s]
                   all         100         302       0.529       0.763       0.703       0.519
           Poison Ivy          100          61       0.504        0.82       0.721        0.52
     Virginia Creeper          100          61       0.508       0.754       0.714       0.539
              Bramble          100         108       0.656       0.758       0.744       0.563
            Box Elder          100          72       0.448       0.722       0.631       0.455
    Speed: 6.3/1.3/7.6 ms inference/NMS/total per 640x640 image at batch-size 32
```

```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --weights weights/yolov5_vanilla.pt --source ./data/images/test/{image_file}
      display(Image(filename='/content/yolov5/inference/output/'+image_file, width=600))
      print("\n\n")
```

```
          nonzero()
    Consider using one of the following signatures instead:
          nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
      i, j = (x[:, 5:] > conf_thres).nonzero().t()
    image 1/1 data/images/test/virginiacreeper_00113509.jpg: 640x512 1 Virginia Creepers, Done. (0.024s)
    Results saved to /content/yolov5/inference/output
    Done. (0.088s)
```

YOLOv5_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

Double Vision

```
[ ] !python test.py --data dataset.yaml --weights weights/yolov5_double_vision.pt --verbose
```

```
Namespace(augment=False, batch_size=32, conf_thres=0.001, data='dataset.yaml', device='', img_size=640, iou_thres=0.65, merge=False, save_jso
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 407 layers, 8.84538e+07 parameters, 0 gradients
Fusing layers...
Model Summary: 284 layers, 8.84108e+07 parameters, 8.45317e+07 gradients
Caching labels data/labels/test (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 11749.41it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0% 0/4 [00:00<?, ?it/s]/content/yolov5/utils/
       nonzero()
Consider using one of the following signatures instead:
       nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:03<00:00,  1.12it/s]
                 all         100         302       0.577       0.778       0.717       0.547
          Poison Ivy         100          61       0.501       0.852       0.715       0.532
    Virginia Creeper         100          61       0.617       0.754       0.738       0.552
             Bramble         100         108       0.747       0.713       0.755       0.611
           Box Elder         100          72       0.441       0.792       0.658       0.492
Speed: 6.3/1.0/7.3 ms inference/NMS/total per 640x640 image at batch-size 32
```

```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --weights weights/yolov5_double_vision.pt --source ./data/images/test/{image_file}
      display(Image(filename='/content/yolov5/inference/output/'+image_file, width=600))
      print("\n\n")
```

```
       nonzero()
Consider using one of the following signatures instead:
       nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
image 1/1 data/images/test/virginiacreeper_00113509.jpg: 640x512 1 Virginia Creepers, Done. (0.024s)
Results saved to /content/yolov5/inference/output
Done. (0.086s)
```

YOLOv5_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

Four Eyes

```
[ ] !python test.py --data dataset.yaml --weights weights/yolov5_four_eyes.pt --verbose
```

```
Namespace(augment=False, batch_size=32, conf_thres=0.001, data='dataset.yaml', device='', img_size=640, iou_thres=0.65, merge=False, s
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 407 layers, 8.84538e+07 parameters, 0 gradients
Fusing layers...
Model Summary: 284 layers, 8.84108e+07 parameters, 8.45317e+07 gradients
Caching labels data/labels/test (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 11636.30it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0% 0/4 [00:00<?, ?it/s]/content/yolov5
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:03<00:00,  1.21it/s]
                 all         100         302       0.615       0.762       0.747       0.571
          Poison Ivy         100          61       0.462       0.803       0.742       0.563
    Virginia Creeper         100          61       0.681       0.734       0.767        0.57
             Bramble         100         108       0.793       0.704       0.735       0.603
           Box Elder         100          72       0.523       0.806       0.745       0.547
Speed: 6.4/1.0/7.3 ms inference/NMS/total per 640x640 image at batch-size 32
```

```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --weights weights/yolov5_four_eyes.pt --source ./data/images/test/{image_file}
      display(Image(filename='/content/yolov5/inference/output/'+image_file, width=600))
      print("\n\n")
```

```
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
image 1/1 data/images/test/virginiacreeper_00113509.jpg: 640x512 1 Poison Ivys, 2 Virginia Creepers, Done. (0.026s)
Results saved to /content/yolov5/inference/output
Done. (0.093s)
```

YOLOv5_Field_Tests_Results.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 20

+ Code + Text

2K

```
[ ] !python test.py --data dataset.yaml --weights weights/yolov5_2k.pt --verbose
```

```
Namespace(augment=False, batch_size=32, conf_thres=0.001, data='dataset.yaml', device='', img_size=640, iou_thres=0.65, merge=False, save_json=False, single_c
Using CUDA device0 _CudaDeviceProperties(name='Tesla V100-SXM2-16GB', total_memory=16130MB)

Model Summary: 407 layers, 8.84538e+07 parameters, 0 gradients
Fusing layers...
Model Summary: 284 layers, 8.84108e+07 parameters, 8.45317e+07 gradients
Caching labels data/labels/test (80 found, 0 missing, 20 empty, 0 duplicate, for 100 images): 100% 100/100 [00:00<00:00, 9634.55it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:   0% 0/4 [00:00<?, ?it/s]/content/yolov5/utils/utils.py:580: Use
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:03<00:00,  1.21it/s]
                 all         100         302       0.667       0.711        0.69       0.544
          Poison Ivy         100          61       0.554       0.803       0.685       0.545
    Virginia Creeper         100          61       0.783       0.656       0.721       0.543
             Bramble         100         108       0.796       0.704       0.711        0.59
           Box Elder         100          72       0.535       0.681       0.645       0.497
Speed: 6.2/1.1/7.3 ms inference/NMS/total per 640x640 image at batch-size 32
```

```
[ ] for test_image_path in image_files:
      image_file = test_image_path.split("/")[-1]
      !echo {image_file}
      !python3 detect.py --weights weights/yolov5_2k.pt --source ./data/images/test/{image_file}
      display(Image(filename='/content/yolov5/inference/output/'+image_file, width=600))
      print("\n\n")
```

```
        nonzero()
Consider using one of the following signatures instead:
        nonzero(*, bool as_tuple) (Triggered internally at  /pytorch/torch/csrc/utils/python_arg_parser.cpp:882.)
  i, j = (x[:, 5:] > conf_thres).nonzero().t()
image 1/1 data/images/test/virginiacreeper_00113509.jpg: 640x512 1 Virginia Creepers, Done. (0.025s)
Results saved to /content/yolov5/inference/output
Done. (0.089s)
```